## Coherent System and Data Description

## 1. System Functionality Overview

## Your project is a start-to-finish as in sight manufacturing sensor monitoring platform that is made up of:

## - Real-time sensor data visualization

## - Anomaly detection with ML

## - User management and authentication

## - Responsive web interface

## 2. Sensor Data Handling

## Data Format (from CSV: line4.csv, line5.csv)\*\*

## The raw data returned from Line 4 and Line 5 sensor is in CSV format, for example:

## Line 4 CSV Structure:

## timestamp, r01, r02, ..., r08

## 2025-04-01 12:00:00.000, 15, 12, ..., 18

## Line 5 CSV Structure:

## timestamp, r01, r02, ..., r17

## 2025-04-01 12:00:00.000, 10, 13, ..., 19

## -These are batch loaded into the database (line4\_data, line5\_data).

## -Timestamps are cleaned, time zones are stripped off, and then parsed into datetime objects.

## -Both for speculative purposes in the past and seeding the ML prediction models.

## 3. Machine Learning: Predict & Status Classification

## Forecast Logic:

## -Prophet models predict expected values based on time series values.

## Downstream, the function forecast\_with\_prophet in model.py captures single point forecast.

## Status Logic:

## -All predicted values have been classified as:

## -Green: within threshold (10–20)

## -Red: out-of-threshold

## -Amber: edge/fallback

## Model Files:

## - 25 Prophet models (8 Line 4, 17 Line 5)

## - Loaded via at runtime.

## - The compiled .pyc file.

## 4. Environment & Dependencies

## Virtual Environment:

## -Created with Python 3.11.3

## -Probably has flask

## Greenlet Integration:

## -Submodule of greenlet package. Used for coroutine switching.

## -Supports async frameworks

## 5. Full-Stack Architecture

## Frontend: HTML, CSS, JS – responsive UI with forms, navbar, dashboards

## Backend API: Flask – routing, ML, CSV ingest.

## ML: Prophet – time-series anomaly detection

## Database: SQLite – sensor data storage

## User Auth: PHP – login/register system with session and admin panel

## Deployment: Localhost

## 6. User Roles & Experience

## Normal User:

## - Registers, logs in

## - Redirected to user home

## - Views dashboards

## Admin:

## - Access to admin page and admin users

## - Edit user accounts and authorize data from sensor

## Wrap-Up

## This system is a robust smart and well architecture monitoring platform for smart manufacture or industrial IoT. It offers:

## - Real-time + historical tracking

## - Machine learning-based anomaly alerts

## - Admin/user segregation

## - Modular, extensible backend

## - Clean, responsive frontend design

System Requirements and Specifications

1. Coherent Overview

This project is, industrial monitoring system, full stack, capture, and analyse data from 2 manufacturing lines. It provides real time, and historical data to view, using machine learning abstraction reporting anomalies, user identity authentication, role-based dashboards as well as administrative features of access control.

2. Functional Requirements

- Read sensor data from the CSV files of Line 4 and Line 5 to SQLite.

- Create restful API endpoints to retrieve pagination data with the optional filtering by date.

- Use existing records, increment times and generate synthetic, real-time sensor data by sampling.

- Use of Prophet-based ML for time-series forecast per sensor.

- Choice Predictions Vs Threshold to get status (Green, Red, Amber).

- Display sensor values and status in interactive dashboard.

- Allow user registration, user login, user logout and role redirection (user vs admin).

- Give admin view registered users and managing it.

3. Non-Functional Requirements

- Performance: The system should be able to recognize and make predictions very quickly.

- Scalability: Handles inflating data sets for the batch in addition to optimized queries.

- Reliability: Does retry on database locks and is nice to failed predictions.

- Usability: Clean, intuitive, mobile-responsive user interface with clear visual feedback.

- Security: Uses session-based authentication. Passwords should be password hashed.

- Maintainability: Separated role and function-based python and php code base.

- Compatibility: Built with Python 3.11, Flask, Prophet, PHP, and SQLite.

4. System Constraints

- SQLite is being used which is file based and might have limitation of superior performance for production environments

- ML models (Prophet) need the signature pre-trained files and enough historical data.

- The current sensor data is required to be conform to predefined CSV format.

- Frontend has only html/css/js compatibility issues.

- Real time simulation comes from the random data, not real hardware integration.

5. Specifications Summary

- Database: SQLite

- Backend: Flask, PHP

- ML Library: Prophecy for forecasting, for model save/ load

- UI: HTML, CSS, JS

- Data Input: Time stamped sensor read between CSV file.

External Admins

Interface

Admins

Operator

Interface Design and Functional Correlation

1. Login & Registration Interfaces

Functional Purpose:

- Authenticate users securely.

- Allow new user registration.

Interface Features:

-Clean Css form layout with input labels (username, email, password).

- Blue-based UI with rounded box styling and mouse on effects.

- Inline Non-Valid Submission alerts.

- Links login/register.

Functional Correlation:

- Automatically voted with PHP scripts (login\_form.php, register\_form.php).

- In submit, credentials are checked against the records in user\_db.sql

- Session creation and redirection logic (admin\_page.php or user\_home.php).

2. Dashboard Interface (User View)

Functional Purpose:

- Show real time sensor from Line 4 and Line 5.

- Display real-time anomality status (green, red, amber).

- Allow historical data retrieval.

Interface Features:

- Centrally located cards or charts.

- Color-coded indicators for the prediction status.

- Navigation bar with user greeting and logout button.

- Media queries using styles.css (responsive design is implemented on page zoom.

Functional Correlation:

- Fetch sensor data from Flask API /api/data.

- Data is processed and laid out via JavaScript/HTML.

- Response prediction status from model.py.

- Periodically or on user request refreshes.

3. Admin Panel Interface

Functional Purpose:

- Give access to user management.

- make view of all registered user, and account role

- Admin-only control panel interface.

A screenshot of a login form

AI-generated content may be incorrect.

A screenshot of a login form

AI-generated content may be incorrect.#A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a calendar

AI-generated content may be incorrect.