**Functional Requirements:**

1. **Dashboard Display** 
   * Organise the sensors logically for better readability and visualisation.
   * Display live sensor data and historic data
   * Display filtering options for the user to apply on the historic sensor data
2. **Display Statistics**
   * When specific sensors are clicked on, specific sensor data should be displayed to the user.
   * Statistics include sensor charts, pie charts, minimum, maximum, average values, and anomalies for each sensor.
3. **Implement Signup Process**
   * **Login**: Users should be able to log in using email and password.
   * **Password Security**: Implement a hashing technique to securely store passwords in the database.
   * **Reset Password**: Users should be provided with the option to reset their passwords
4. **Implement Login System**
   * **Login**: Users should be able to log in using email and password.
   * **Password Security**: Implement a hashing technique to securely store passwords in the database.
   * **Reset Password**: Users should be provided with the option to reset their passwords in the event that they have forgotten theirs
5. **Define User Roles and Access Privileges**:
   * **Production Operators**: Can only view and apply filters to the data by specific time ranges to refine data displayed to them
   * **Managers/Admins**:

* Can approve new user registrations
* Can optionally remove/add sensors
* Assign temporary passwords for registration
* Can reset passwords for users who have forgotten their passwords

1. **API Development**
   * **Interact with database**:
     + The system must be able to **read and process historical sensor data** from CSV files stored in the database.
     + The system will **authenticate user logins** by verifying the credentials and existence of the user stored in a separate database table, which includes emails and hashed passwords.
   * **Simulate Real-Time Data**: The API should simulate real-time data. This could be achieved either using synthetic data, replaying historical data or implementing a random number generator. The most suitable approach will be selected based on its effectiveness for the system.
   * **Update Data**: Simulated real-time data must be displayed and updated live on the dashboard, at specific time intervals.
   * **Filtering**: Allow filtering of data by specific time ranges to refine displayed data.
2. **Web Application Responsiveness**
   * **Device Compatibility**: The web application must be responsive and accessible on both desktop and tablet devices.
3. **Traffic-Light System**
   * **Anomaly Detection**: Flag sensor data points with a traffic light system (green, yellow, red) based on deviation from expected values.
   * Traffic light system should be integrated with the ML model, however as ML model integration is an optional task, the traffic light system could alternatively be implemented (e.g. to identify anomalies within the past hour)
4. **Machine Learning Model Integration (optional)**
   * **Predict Sensor Values**: The provided pre-trained ML model will predict the expected sensor values for a given timestamp, with these predictions being used internally within the traffic light system. They will not be presented to the user as actual sensor data but will instead serve as a reference for system operations.
   * **Integration**: The model should be integrated into the backend via an API or Python code.

**Non-Functional Requirements:**

1. **Light/Dark Mode (optional)**
   * **Mode Toggle**: Implement a toggle to allow users to switch between light and dark modes.
2. **Flexibility for Future Sensor Additions (optional)**
   * **Scalable Model**: The system must be able to accommodate the addition of new sensors without requiring major changes or experiencing a drop in performance.
   * **Dynamic Sensor Display**: The frontend must dynamically adjust to new sensor types without hardcoded changes.
3. **Performance**
   * **Fast Response Times**: The web application should load quickly and update sensor data in real-time without delays.
4. **Scalability**
   * **Expandable Architecture**: The system should be designed to easily add more sensors or integrate new features in the future.
5. **Reliability**
   * **Consistent Data**: Simulated real-time data displayed on the dashboard should be consistent with the actual sensor readings and historical data.
6. **Usability**
   * **User-Friendly Interface**: The application should have a clear, intuitive interface that is easy to navigate.
   * **Accessibility**: Ensure the application follows best practices for accessibility (e.g., colour contrast, font readability).
   * **Tooltips and Help Sections**: Provide user guidance through tooltips and a help section to support new or less-technical users.
7. **Maintainability**
   * **Modular Codebase**: The code should be modular and easily maintainable, with clear separation of concerns.
   * **Flexible Configuration**: The system should allow easy configuration and addition of new sensors or features without extensive rewrites.
   * **Adherence to Coding Best Practices:** Standard coding practices, such as using clear and descriptive variable names, , should be consistently applied to ensure ease of collaboration.
8. **Data Privacy**
   * **Anonymity of User Data**: Store only the necessary user data (e.g., hashed passwords) and avoid storing sensitive personal information.

**Actors Section with Reference to Functional and Non-Functional Requirements:**

**Product Operators**

Product Operators will login to the system (Functional Requirement 3) and have the least access privileges within the system as they can only view historical and real-time data, as well as access statistical analysis for the sensors. Additionally, they will have the ability to filter data based on specific time intervals (Functional Requirement 4).

**Manager**

Managers will login to the system (Functional Requirement 2) and can interact with the system in the same way as Production. However, they also have higher privileges as they are responsible for approving or declining user registrations, thereby controlling access to the system (Functional Requirement 4). They may also have additional capabilities, such as adding or deleting sensors, which enhances the system's flexibility and adaptability (Non-functional Requirement 2).

**Platform (Web Application)**

The platform consists of a visualisation dashboard which displays sensor data (including live and historic data) and subsequent statistics, whilst also providing filtering options for the user to apply onto historic data (Functional Requirement 1). The dashboard must also have a responsive design (Functional Requirement 6), with optional light/dark mode toggles and user help guides to assist accessibility (Non-Functional Requirements 1,6).

The API enables communication with the database for storing and retrieving data. This includes logging temperature data from sensors across two production lines and managing user details stored in separate database tables. It interacts with the database to retrieve historical data and user credentials, as well as simulating real-time data that’s updated at specific intervals. The API will also implement filtering logic for historical data (Functional Requirement 4) and incorporate a traffic light system for anomaly detection (Functional Requirement 7), which works in conjunction with an optional ML model (Functional Requirement 8).

A diagram of a company's process

AI-generated content may be incorrect.