**Functional Requirements:**

1. **Display sensors**
   * **Group Sensors**: Organise the sensors logically (e.g., by production line or sensor enumeration) for better interpretation and analysis.
2. **Display statistics**
   * When specific sensors are clicked on, specific sensor data should be displayed to the user
   * Statistic include sensor charts, pie charts, minimum, maximum, avg, anomalies for each sensor
3. **Implement Authentication System**
   * **Login**: Users should be able to log in using email and password.
   * **User Roles**:
     + **Production Operators**: View-only access to the data.
     + **Managers/Admins**: Approve/decline new registrations and manage user roles.
   * **Password Security**: Implement hashing and encryption to securely store passwords in the database
4. **API Development**
   * **Read Historical Data**: The system must be able to read and process historical sensor data from CSV files.
   * **View Historical Data**: Users should be able to access past historical sensor data.
   * **Simulate Real-Time Data**: The API should simulate real-time data, either using synthetic data or replaying historical data/random number generator
   * **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.
5. **Web Application Responsiveness**
   * **Device Compatibility**: The web application must be responsive and accessible on both desktop and tablet devices.
6. **Traffic-Light System**
   * **Anomaly Detection**: Use a machine learning model to flag sensor data points with a traffic light system (green, yellow, red) based on deviation from expected values.
   * As ML implementation is an optional task, traffic-light system could alternatively be implanted…
7. **Machine Learning Model Integration**
   * **Predict Sensor Values**: The pre-trained ML model should predict expected sensor values for a given timestamp, and e upper/lower bounds should be defined.
   * **Integration**: The model should be integrated into the backend via an API or Python code.

ACTORS:

The following section outlines the key actors within the system, describing their roles, responsibilities, and access privileges to ensure efficient operation and management of the platform.

**Product Operators**  
Product Operators will have the least access privileges within the system. They can 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 timestamps.

**Manager**  
Managers have higher privileges and are responsible for approving or declining user registrations, thereby controlling access to the system. They may also have additional capabilities, such as adding or deleting sensors, which enhances the system's flexibility and adaptability.

**API**  
The API acts as the intermediary between both the operators and managers. It reads historical data, generates real-time data, and integrates an ML model for anomaly detection. Additionally, the API facilitates communication with the database for data storage and retrieval. This includes storing temperature data from sensors across two production lines, as well as managing user details stored in separate tables within the database. It also handles the implementation of the traffic light system for monitoring.

NON-FUNCTIONAL REQUIREMENTS:

1. **Light/Dark Mode**
   * **Mode Toggle**: Implement a toggle to allow users to switch between light and dark modes.
2. **Flexibility for Future Sensor Additions**
   * **Scalable Model**: The system must be able to accommodate the addition of new sensors without requiring major changes.
   * **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**
   * **Scalable Data Handling**: The system should be able to handle an increasing number of sensors and users without a significant drop in performance.
   * **Expandable Architecture**: The system should be designed to easily add more sensors or integrate new features in the future.
5. **Reliability**
   * **Fault Tolerance**: The system should be robust and capable of recovering from failures without affecting critical operations.
   * **Consistent Data**: Data displayed on the dashboard should be consistent with the actual sensor readings and historical data.
   * **Backup and Recovery**: Backup mechanisms for storing data securely, and the ability to recover from failure scenarios.
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.
   * **Documentation**: Comprehensive documentation for the codebase, API, and user interface should be provided to support future maintenance and upgrades.
   * **Flexible Configuration**: The system should allow easy configuration and addition of new sensors or features without extensive rewrites.
8. **Compatibility**
   * **Cross-Browser Compatibility**: The web application should work across popular browsers (e.g., Chrome, Firefox, Safari).
   * **Mobile Responsiveness**: The system must be responsive and functional across both desktop and tablet devices.
9. **Data Integrity**
   * **Accurate Data Visualization**: Data displayed on the dashboard must be accurate, reflecting the true values of the sensor readings and calculations.
   * **Real-Time Accuracy**: The real-time data displayed must be consistently updated and reflect actual sensor behavior or synthetic simulation.
10. **Data Privacy**
    * **Anonymity of User Data**: Store only the necessary user data (e.g., hashed passwords) and avoid storing sensitive personal information.

**Functional Requirements:**

1. **Display Sensors**
   * Organise the sensors logically (by grouping sensors (e.g.1-4. etc) for better readability and organisation.
   * Display live sensor data and historic data
   * Display filtering functionality for the user to apply on the historic 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 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 able to reset their passwords
4. **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 additionally approve/decline new registrations and manage user roles.
5. **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 **grant user access** according to the login details stored in a separate table in the database (this include emails and hashed passwords)
   * **Simulate Real-Time Data**: The API should simulate real-time data, either using synthetic data or replaying historical data/random number generators.
   * **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.
6. **Web Application Responsiveness**
   * **Device Compatibility**: The web application must be responsive and accessible on both desktop and tablet devices.
7. **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)
8. **Machine Learning Model Integration (optional)**
   * **Predict Sensor Values**: The provided pre-trained ML model will predict expected sensor values for a given timestamp, and upper/lower bounds should be defined.
   * **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**
   * **Fault Tolerance**: The system should be robust and capable of recovering from failures without affecting critical operations.
   * **Consistent Data**: Data displayed on the dashboard should be consistent with the actual sensor readings and historical data.
   * **Backup and Recovery**: Backup mechanisms for storing data securely, and the ability to recover from failure scenarios.
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.
   * **Documentation**: Comprehensive documentation for the codebase, API, and user interface should be provided to support future maintenance and upgrades.
   * **Flexible Configuration**: The system should allow easy configuration and addition of new sensors or features without extensive rewrites.
8. **Compatibility**
   * **Cross-Browser Compatibility**: The web application should work across popular browsers (e.g., Chrome, Firefox, Safari).
9. **Data Integrity**
   * **Accurate Data Visualization**: Data displayed on the dashboard must be accurate, reflecting the true values of the sensor readings and calculations.
   * **Real-Time Accuracy**: The real-time data displayed must be consistently updated and reflect actual sensor behaviour or synthetic simulation.
10. **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 2) 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 3).

**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 areresponsible for approving or declining user registrations, thereby controlling access to the system (Functional Requirement 3). 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) ans 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 facilitates communication with the database for data storage and retrieval. This includes storing temperature data from sensors across two production lines, as well as managing user details stored in separate tables within the database. It interacts with the database to retrieve and read historical data and user credentials , as well as simulating real-time data(Functional Requirement 4). Additionally, a traffic light system is implemented here to facilitate anomaly detection (Functional Requirement 6) , in culmination with an optional ML model (Functional Requirement 7).

UML USE CASE DIAGRAM: