Software Requirements Specification (SRS)

1. Introduction

1.1 Purpose

The purpose of this document is to define the requirements for the Interactive Real-Time Visualisation Dashboard for Rakusens. The system will provide a web-based solution for monitoring temperature data from multiple sensors in real-time.

1.2 Scope

The system will include:

* Real-Time Sensor Data Monitoring – Live visualisation of temperature readings.
* Historical Data Retrieval – Ability to view past sensor data.
* Traffic-Light Anomaly Detection – AI-driven alerts for abnormal temperature readings.
* User Authentication & Roles – Admins, Operators, and General Users.
* Data Export & Reporting – Generate reports in CSV, PDF, and Excel formats.
* Integration with PostgreSQL Database – Store and retrieve sensor data efficiently.

1.3 Team Members

* Kian Vishnu Biswas
* Ehtasham Afzal
* Mahad Nazar
* Adam Rasheed
* Tayyib Khan
* Areesha Qazi
* Lina Khalid Naciri

1.4 References

* [Client-provided documentation] [To be filled]
* [Project proposal] [To be filled]

2. Functional Requirements

2.1 User Roles

* Administrator – Manages user accounts, configures system settings.
* Operator – Monitors temperature data, acknowledges alerts.
* General User – Views historical data and trends.

2.2 Features

User Authentication

* Login, registration, and password reset functionality.
* Role-based access control (RBAC).

Real-Time Sensor Data Monitoring

* Fetch live temperature readings via API.
* Display interactive graphs using Plotly.js.

Anomaly Detection & Alerts

* Machine Learning model flags abnormal readings.
* Notifications via email/web alerts.

Historical Data & Reports

* Query past temperature trends.
* Export reports (CSV/PDF/Excel).

System Performance & Security

* Secure encryption for data storage.
* Scalable backend to handle increasing data volumes.

3. Data Description

| Field | Type | Description |
| --- | --- | --- |
| sensor\_id | String | Unique identifier for a sensor |
| line | Integer | Production line number (4 or 5) |
| temperature | Float | Temperature reading |
| timestamp | DateTime | Data recording time |

4. Interface Design

Main Pages:

* Login Page – User authentication.
* Dashboard – Real-time sensor readings.
* Historical Data Page – Filter & view past data.
* Admin Panel – Manage users & settings.

5. Risks & Ethical Considerations

| Risk | Mitigation |
| --- | --- |
| Data privacy concerns | Ensure GDPR compliance |
| System downtime | Implement redundant backup strategies |
| ML inaccuracies | Clearly state ML limitations |

Software Design Document (SDD)

1. System Overview

The system follows a three-tier architecture:

1. Frontend: React.js (UI & user interaction).
2. Backend: Node.js with Express (API handling).
3. Database: PostgreSQL (data storage & retrieval).

2. Architectural Design

Components:

* API Layer – Handles requests between frontend & database.
* Database Layer – Stores sensor readings & user data.
* Machine Learning Module – Detects anomalies in sensor readings.

3. API Design

| Endpoint | Method | Description |
| --- | --- | --- |
| /api/login | POST | User authentication |
| /api/register | POST | Create new user |
| /api/historical-data | GET | Fetch past sensor readings |
| /api/real-time-data | GET | Retrieve live sensor readings |
| /api/anomaly-detection | POST | ML-based anomaly detection |

4. Database Schema

CREATE TABLE users (

id SERIAL PRIMARY KEY,

username VARCHAR(255) UNIQUE NOT NULL,

password VARCHAR(255) NOT NULL,

role VARCHAR(50) CHECK (role IN ('admin', 'operator', 'user'))

);

CREATE TABLE sensor\_data (

id SERIAL PRIMARY KEY,

sensor\_id VARCHAR(50),

line INTEGER,

temperature FLOAT,

timestamp TIMESTAMP

);

5. Use Case Diagram

[To be drawn]

6. Testing Plan

| Test Case | Expected Outcome | Pass/Fail |
| --- | --- | --- |
| User login with incorrect credentials | Displays error message | [To be filled] |
| Fetching real-time sensor data | Returns live sensor readings | [To be filled] |
| Anomaly detection API | Flags out-of-bound values | [To be filled] |

7. Deployment Strategy

* Development: Local PostgreSQL & Node.js environment.
* Production: Deploy on AWS / Heroku / Docker.

8. Next Steps

* Confirm API framework with client.
* Develop frontend wireframes.
* Set up GitHub repository for code management.