

**DEVELOPING AN IOT-BASED INVENTORY MANAGEMENT SYSTEM USING
NODE-RED: ENHANCING SMART FACTORY ENVIRONMENTS**

BY
TAN YI FEI

A REPORT
SUBMITTED TO
Universiti Tunku Abdul Rahman
in partial fulfillment of the requirements
for the degree of
BACHELOR OF INFORMATION TECHNOLOGY (HONOURS) COMMUNICATIONS
AND NETWORKING
Faculty of Information and Communication Technology
(Kampar Campus)

JUNE 2025

COPYRIGHT STATEMENT

© 2025 Tan Yi Fei. All rights reserved.

This Final Year Project report is submitted in partial fulfillment of the requirements for the degree of **Bachelor of Information Technology (Honours) Communications and Networking** at Universiti Tunku Abdul Rahman (UTAR). This Final Year Project report represents the work of the author, except where due acknowledgment has been made in the text. No part of this Final Year Project report may be reproduced, stored, or transmitted in any form or by any means, whether electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the author or UTAR, in accordance with UTAR's Intellectual Property Policy.

ACKNOWLEDGEMENTS

I would like to express my sincere thanks and appreciation to my supervisor, Ts. Dr. Goh Hock Guan, for his invaluable guidance, support, and encouragement throughout the development of my final year project. His expertise and insightful feedback have played a crucial role in shaping the direction and success of this project. Thank you for providing me with this opportunity to explore and grow in the field of IoT and smart factory inventory systems.

To my dearest friends, whose strong determination and resilience in overcoming challenges have been my source of inspiration. Their strengths and perseverance have encouraged and supported me throughout this journey.

Lastly, I would like to extend my deepest appreciation to my parents and family. Their unconditional love, continuous support, and constant encouragement have become the foundation of my academic journey and personal growth.

ABSTRACT

The integration of the Internet of Things (IoT) within smart factory environments is revolutionizing inventory management processes to make them more efficient and responsive. This project focuses on developing a simulation-based IoT inventory management system using Node-RED to demonstrate smart factory operations such as real-time monitoring, inventory tracking, predictive maintenance, and quality control. The system integrates technologies of Node-RED, MQTT, InfluxDB, and Telegram to simulate and visualize factory activities.

All data used in this simulation is generated virtually within Node-RED to mimic the behavior of a real-world manufacturing environment. This includes simulated temperature, humidity and pressure readings, electronic components movement across various stages (mounting, soldering, inspection, testing, packaging), inventory level changes, and machine runtimes. The system also features a notification mechanism via Telegram to alert factory managers in real time about key events such as high temperature, defective components, low inventory levels, and scheduled maintenance.

This simulation provides a cost-effective and scalable platform for testing and visualizing IoT-based inventory management solutions. The use of Node-RED allows flexible flow-based programming so that the system is highly adaptable for educational and prototyping purposes. Additionally, real-time data tracking, automated decision-making, and visualization through the Node-RED Dashboard enable a deeper understanding of how smart factories can operate more efficiently.

This project highlights the potential of IoT integration for achieving improved resource utilization, reduced downtime, and enhanced decision-making in inventory management processes. It serves as a foundation for further development toward more intelligent, connected, and automated factory systems.

Area of Study: **Internet of Things (IoT), Industrial Automation**

Keywords: **Smart Factory Simulation, Node-RED, Scalable Vector Graphics (SVG), Real-time Monitoring, Inventory Management**

TABLE OF CONTENTS

TITLE PAGE	i
COPYRIGHT STATEMENT	ii
ACKNOWLEDGEMENTS	iii
ABSTRACT	iv
TABLE OF CONTENTS	v
LIST OF FIGURES	ix
LIST OF TABLES	xiii
LIST OF ABBREVIATIONS	xiv
CHAPTER 1 INTRODUCTION	1
1.1 Problem Statement and Motivation	1
1.2 Project Objectives	4
1.3 Project Scope	5
1.4 Main Contributions from the Project	6
1.5 Organization of the Report	7
1.6 Concluding Remark	7
CHAPTER 2 LITERATURE REVIEW	8
2.1 Review of the Technologies	8
2.1.1 Hardware Platform	8
2.1.2 Database	10
2.1.3 Programming Language	12
2.1.4 Summary of the Technologies Review	14
2.2 Review of the Existing Systems/Application	16
2.2.1 Using Node-RED Platform inn an Industrial Environment	16
2.2.2 Industrial Real-Time Digital Twin System for Remote Teaching Using Node-RED	18
2.2.3 Advancing Small and Medium-Sized Enterprise Manufacturing: Framework for IoT-Based Data Collection in Industry 4.0 Concept	21

2.2.4	Warehousing 4.0: A proposed system of using node-red for applying internet of things in warehousing	24
2.2.5	Summary of the Existing Systems	26
2.3	Concluding Remark	29
CHAPTER 3 SYSTEM METHODOLOGY		30
3.1	System Development Models	30
3.1.1	Rapid Application Model (RAD)	30
3.1.2	Agile Development Model	32
3.1.3	Prototype Model	33
3.1.4	V-Model (Verification and Validation Model)	34
3.1.5	Selected Model – Prototype Model	35
3.2	System Requirements	37
3.2.1	Hardware	37
3.2.2	Software	39
3.3	Functional Requirement	42
3.4	Project Milestone	43
3.5	Concluding Remark	44
CHAPTER 4 SYSTEM DESIGN		45
4.1	System Architecture	45
4.2	Functional Modules in the System	47
4.3	System Flow	48
4.4	Database Design	50
4.5	GUI Design	51
4.6	Concluding Remark	54
CHAPTER 5 SYSTEM IMPLEMENTATION		55
5.1	Software Setup	55
5.1.1	Inkscape (SVG Editor) Installation	55
5.1.2	Node-RED Installation and Setup	56
5.1.3	InfluxDB Database Installation and Setup	59
5.1.4	MQTT Installation and Setup	62

5.1.5	Telegram Bot Setup and Configuration	63
5.2	Setting and Configuration	65
5.2.1	InfluxDB Setting and Configuration	65
5.2.2	MQTT Setting and Configuration	68
5.2.3	Telegram Bot Setting and Configuration	78
5.3	System Operation	82
5.3.1	Main Dashboard Interface	82
5.3.2	System Operation of Production Flow	85
5.3.3	System Operation of Environmental Monitoring	87
5.3.4	System Operation of Inventory Tracking	88
5.3.5	System Operation of Telegram Alerts and Commands	89
5.3.6	System Operation of IoT MQTT Panel	94
5.4	Concluding Remark	94
CHAPTER 6	SYSTEM EVALUATION AND DISCUSSION	95
6.1	System Testing and Performance Metrics	95
6.2	Testing Setup and Result	98
6.2.1	Main Dashboard	98
6.2.2	Production Flow	99
6.2.3	Inventory Table	100
6.2.4	Environmental Monitoring	100
6.2.5	Telegram Bot Integration	101
6.2.6	IoT MQTT Panel Integration	104
6.2.7	Database Logging and Gauges	105
6.3	Project Challenges	105
6.4	SWOT	107
6.5	Objectives Evaluation	108
6.6	Concluding Remark	110

CHAPTER 7 CONCLUSION AND RECOMMENDATION	111
7.1 Conclusion	111
7.2 Recommendation	112
 REFERENCES	 114
POSTER	118
APPENDIX	119

LIST OF FIGURES

Figure Number	Title	Page
Figure 2.2.1.1	System architecture of the proposed system	17
Figure 2.2.2.1	Project development diagram	19
Figure 2.2.2.2	Request system on dashboard in Node-Red	19
Figure 2.2.2.3	Real virtual platform and Digital Twin plan constructed with SVG images	20
Figure 2.2.3.1	Designing the process of upgrading SMEs to the Smart Factory level	22
Figure 2.2.3.2	Final deployment of framework for upgrading SMEs to Smart Factory level	23
Figure 2.2.4.1	Flowchart of the proposed system	25
Figure 3.1.1.1	Rapid Development Model	31
Figure 3.1.2.1	Graphical illustration of the Agile Model	32
Figure 3.1.3.1	Prototype Model phases	33
Figure 3.1.4.1	V-Model phases	34
Figure 3.2.1.1	IdeaPad 3 15ITL6	37
Figure 3.2.1.2	iPhone 15 Pro Max	38
Figure 3.2.2.1	The GUI of Node-RED	39
Figure 3.2.2.2	Implementation of the MQTT modules in Node-RED	40
Figure 3.2.2.3	The GUI of InfluxDB	40
Figure 3.2.2.4	The logo of Telegram Bot Father	41
Figure 3.2.2.5	Inkscape interface	41
Figure 3.4.1	Gantt Chart for FYP1	43
Figure 3.4.2	Gantt Chart for FYP2	43
Figure 4.1.1	Architecture diagram of the project	45
Figure 4.3.1	System flowchart of the project	48
Figure 4.4.1	InfluxDB database values	50
Figure 4.5.1	SVG graphics of smart factory layout (Part 1)	51
Figure 4.5.2	SVG graphics of smart factory layout (Part 2)	51
Figure 4.5.3	The Node-RED dashboard for assembly line	52

Figure 4.5.4	The Node-RED dashboard for real-time monitoring	52
Figure 4.5.5	MQTT panel	53
Figure 4.5.6	Telegram notifications	53
Figure 4.5.7	Telegram control menu	53
Figure 5.1.1.1	Inscape installation platform	55
Figure 5.1.2.1	Node.js download platform	56
Figure 5.1.2.2	Verification of Node-RED installation's command prompt	56
Figure 5.1.2.3	Command prompt installation	57
Figure 5.1.2.4	Node-RED launching command prompt	57
Figure 5.1.2.5	Node-RED panel	57
Figure 5.1.2.6	Node-RED palette	58
Figure 5.1.3.1	InfluxDB installation platform	59
Figure 5.1.3.2	InfluxDB directory	59
Figure 5.1.3.3	InfluxDB launching command prompt	60
Figure 5.1.3.4	InfluxDB login page	60
Figure 5.1.3.5	InfluxDB platform	61
Figure 5.1.4.1	IoT MQTT Panel application installation	62
Figure 5.1.4.2	Connection configuration for smart factory	63
Figure 5.1.5.1	Telegram Bot Father	63
Figure 5.1.5.2	Telegram Bot creation process	64
Figure 5.2.1.1	Configuration setting for influxdb server	66
Figure 5.2.1.2	Configuration setting for data storage parameters	66
Figure 5.2.1.3	InfluxDB interface	67
Figure 5.2.2.1	MQTT Broker configuration	68
Figure 5.2.2.2	Example payload	69
Figure 5.2.2.3	MQTT Node-RED flow	69
Figure 5.2.2.4	Temperature node function	70
Figure 5.2.2.5	Humidity node function	70
Figure 5.2.2.6	Pressure node function	70
Figure 5.2.2.7	Timestamp node function	71
Figure 5.2.2.8	Actuators toggle switches	71
Figure 5.2.2.9	Node-RED flow of actuators toggle switches	72

Figure 5.2.2.10	Node-RED flow of subscribed MQTT sensor data store in database	72
Figure 5.2.2.11	Real-time environmental monitoring dashboard	73
Figure 5.2.2.12	IoT MQTT Panel connection setup	74
Figure 5.2.2.13	Panel creation in IoT MQTT	75
Figure 5.2.2.14	Panel Configuration	76
Figure 5.2.2.15	Panel Configuration	77
Figure 5.2.2.16	Panel Configuration	77
Figure 5.2.3.1	Telegram Bot creation	78
Figure 5.2.3.2	Bot Configuration	79
Figure 5.2.3.3	Node-RED Flow of telegram receiver	80
Figure 5.2.3.4	Node-RED flow of telegram callback query	80
Figure 5.2.3.5	Answer callback query function node.	81
Figure 5.3.1.1	Main dashboard interface	82
Figure 5.3.1.2	Live machine success rates gauges	83
Figure 5.3.1.3	Control panel in main dashboard	83
Figure 5.3.1.4	Real-time inventory table	84
Figure 5.3.2.1	Production flow of assembly line	86
Figure 5.3.2.2	Production flow of logistics line	86
Figure 5.3.3.1	Environmental panel with sensor readings and corresponding device control indicators	87
Figure 5.3.4.1	Low stock warning Telegram alert	88
Figure 5.3.4.2	Out of stock warning Telegram alert	88
Figure 5.3.5.1	Temperature alert	89
Figure 5.3.5.2	Humidity alert	89
Figure 5.3.5.3	Error handling for unknown commands with helpful guidance	90
Figure 5.3.5.4	Error handling for invalid command format with usage examples	90
Figure 5.3.5.5	Comprehensive help command showing all available bot functions	91
Figure 5.3.5.6	Status command to show current inventory status	91
Figure 5.3.5.7	Restock Command	92

Figure 5.3.5.8	Inventory table updated after restocking	92
Figure 5.3.5.9	Detailed breakdown of all machines statistics report	93
Figure 5.3.5.10	Detailed breakdown of mounting machines statistics report	93
Figure 5.3.5.11	Detailed breakdown of soldering machines statistics report	93
Figure 5.3.5.12	Detailed breakdown of inspection machines statistics report	93
Figure 5.3.6.1	Actuator control through IoT MQTT Panel	94
Figure 6.3.1	Browser instability issue	106

LIST OF TABLES

Table Number	Title	Page
Table 2.2.5.1	Summary of Existing Systems	26-28
Table 3.2.1.1	Specifications of Laptop	37
Table 3.2.1.2	Specifications of Smartphone	38
Table 6.1.1.1	Description and expected outcomes of key testing metrics	95-97
Table 6.2.1.1	System testing and results for main dashboard	98
Table 6.2.2.1	System testing and results for production flow	99
Table 6.2.3.1	System testing and results for inventory table	100
Table 6.2.4.1	System testing and results for environmental monitoring	100
Table 6.2.5.1	System testing and results for telegram bot integration	101- 103
Table 6.2.6.1	System testing and results for MQTT IoT Panel integration	104
Table 6.2.7.1	System testing and results for database logging and gauges	105

LIST OF ABBREVIATIONS

<i>IoT</i>	Internet of Things
<i>MQTT</i>	Message Queuing Telemetry Transport
<i>SVG</i>	Scalable Vector Graphics
<i>KPI</i>	Key Performance Indicators
<i>PC</i>	Personal Computer
<i>RDBMS</i>	Relational Database Management System
<i>RAM</i>	Random Access Memory
<i>UI</i>	User Interface
<i>GPIO</i>	General-Purpose Input/Output
<i>JSON</i>	JavaScript Object Notation
<i>SQL</i>	Structured Query Language
<i>DT</i>	Digital Twin
<i>HMI</i>	Human- Machine Interface
<i>API</i>	Application Programming Interface
<i>SME</i>	Small and Medium-sized Enterprises
<i>PLC</i>	Programmable Logic Controller
<i>MES</i>	Manufacturing Execution System
<i>SCADA</i>	Supervisory Control and Data Acquisition
<i>OPC UA</i>	Open Platform Communication Unified Architecture
<i>RFID</i>	Radio Frequency Identification
<i>GUI</i>	Graphical User Interface
<i>RAD</i>	Rapid Application Development
<i>5G</i>	Fifth Generation
<i>API</i>	Application Programming Interface
<i>CPU</i>	Central Processing Unit
<i>GPIO</i>	General Purpose Input Output
<i>IOT</i>	Internet of Things
<i>IP</i>	Internet Protocol
<i>RAM</i>	Random Access Memory
<i>ML</i>	Machine Learning

Chapter 1

Introduction

1.1 Problem Statement and Motivation

Traditional inventory management systems face several challenges that can hinder their effectiveness in dynamic and fast-paced manufacturing environments.

Problem Statement 1: Inaccurate Data

One of the major drawbacks of conventional inventory management systems is reliance upon outdated manual methods of data entry and tracking. Inventory records are often maintained through spreadsheets, handwritten logs, or standalone software that does not synchronize with other parts of the manufacturing system. This fragmented approach is highly susceptible to human error especially when dealing with large volumes of items in fast-paced environments [1]. Therefore, it leads to common discrepancies in the counts of stock levels, mislabeled items, and overall inventory records that do not represent the actual status of materials or products in real time. Such inaccuracies in inventory data result in two major operational risks which are understocking and overstocking. In both cases, the company suffers from reduced efficiency, poor decision-making, and lower profitability. Without accurate, real-time data, decision-makers are unable to respond effectively to demand changes or disruptions.

Problem Statement 2: Limited Integration Between Systems

A significant limitation in conventional inventory management is the lack of integration between key operational systems. Production machines, inventory records, and environmental monitoring tools often function on separate platforms without automatic data exchange. This fragmentation makes it difficult to maintain a unified, real-time view of factory operations. Due to this disconnect, many tasks still rely on manual input. For instance, if a defect is detected on the production line, stock updates may not occur unless entered manually. Similarly, abnormal environmental readings may go unnoticed or unaddressed unless constantly monitored by staff. This results in delayed responses, increased human error, and reduced overall efficiency. Without integration, factories cannot fully leverage the benefits of digital transformation or operate at optimal efficiency.

Problem Statement 3: Lack of Real-Time Monitoring

The biggest drawback to most traditional inventory management systems is the lack of real-time monitoring [2]. In today's fast-moving manufacturing environments, periodic updates or manual monitoring are well below the mark of requirements for firms wanting to be responsive and agile as a way of staying competitive. Inventory managers and factory supervisors are forced to take a reactive approach to operations because they are unable to monitor inventory levels, equipment health, and production metrics in real time. Lack of real-time visibility into operations implies that they often could react only to events that have already happened rather than proactively preventing them. Inventory managers are unable to react quickly to demand shifts, supply chain disruptions, or equipment failures without real-time monitoring. This reactive approach results in increased operational costs, reduced productivity, and missed opportunities to optimize the manufacturing process [3]. Additionally, equipment failures can be particularly damaging in environments where production efficiency is paramount. Without real-time monitoring, equipment breakdowns may go unnoticed until they cause significant disruptions, halting production lines and resulting in costly downtime.

Motivation

In today's competitive and fast-paced manufacturing landscape, the ability to manage inventory efficiently is a key determinant of operational success. However, traditional inventory management systems often rely on outdated methods such as manual data entry, standalone software, or periodic reporting. These approaches are no longer sufficient to support the high-speed, data-driven nature of modern manufacturing environments.

The motivation behind this project arises from the growing need to modernize inventory management systems to keep up with the expectations and challenges posed by Industry 4.0. This new industrial era emphasizes real-time visibility, automation, predictive analytics, and system integration which are the features that are largely absent in conventional inventory systems. Manufacturing environments now require solutions that not only track inventory levels accurately but also monitor operational conditions, detect anomalies, and provide real-time feedback to stakeholders.

This project is motivated by the desire to bridge the gap between outdated inventory practices and technological advancements through IoT. The goal is to demonstrate how integrating smart technologies into inventory workflows can enhance visibility, accuracy, and responsiveness across the manufacturing process. Real-time tracking, automated alerts, system integration, and predictive capabilities are no longer optional, they are essential for factories to remain competitive and resilient in the face of changing demands.

1.2 Project Objectives

i. **To improve real-time inventory tracking**

The system will automate inventory tracking. Therefore, real-time visibility into the level of stock and location of every asset will be enabled. It eliminates manual checks, hence reducing human errors and increasing the efficiency of the overall inventory management.

ii. **To ensure robust quality control in the production process**

The system incorporates automated defect detection across multiple production stages, ensuring defective components are identified and rejected accurately. Loader color-coding, machine statistics, and real-time dashboards provide transparency and allow operators to track the overall success rate and quality of production.

iii. **To establish reliable IoT communication and notification systems**

The system integrates MQTT, IoT MQTT Panel, and Telegram Bot services to achieve stable communication and remote accessibility. Users can monitor, control, and query the system through multiple platforms while receiving timely alerts for inventory, environmental thresholds, and production status.

iv. **To enhance decision-making with data-driven insights**

The system leverages InfluxDB to store environmental data, component flows, and machine performance metrics with precise timestamps. Dashboards and queries enable both real-time monitoring and long-term analysis, supporting data-driven insights into throughput, defect rates, and efficiency.

1.3 Project Scope

This project focuses on the development of an IoT-based inventory management system for a smart factory using Node-RED, MQTT, InfluxDB, and Telegram. The system will automate and monitor various stages of the manufacturing process, including inventory tracking, defect detection, environmental monitoring, and predictive maintenance.

The project will first concentrate on simulating a factory environment with different production lines, where chips are processed through various stages including loading, mounting, soldering, inspection, and packaging. Different categories of electronic components (cheap, medium, and expensive) will be processed in separate stages.

To monitor environmental conditions, temperature, humidity and pressure will be detected and appropriate actions will be taken to ensure optimal production environments. Data collected will be sent via MQTT to a monitoring device, where it will be stored in InfluxDB for time-series analysis.

Additionally, the system will incorporate a real-time dashboard in Node-RED, which will visualize the status of inventory, machinery, and environmental conditions. The dashboard will display real-time data, including inventory counts, machine status, temperature, and humidity levels. It will also track defective items during the inspection stage, rerouting them to a rejected area.

The system will include a Telegram chatbot configured in Node-RED, allowing factory managers to monitor and control the system remotely. The chatbot will send alerts if any unusual conditions are detected, such as high temperatures, humidity, pressure, defective components and so on. Additionally, the chatbot will allow users to control actuators such as fans, humidifier and dehumidifier based on environmental conditions, as well as receive notifications for low or out of stock warnings, and status reports of the machines and components in the factory.

Once the system is built, performance and reliability testing will be conducted within a simulated factory environment. The system's ability to track inventory in real-time, predict maintenance needs, and ensure product quality will be evaluated based on different operational scenarios.

1.4 Main Contributions from the Project

This project demonstrates the feasibility of a smart factory inventory management system and highlights the effectiveness of IoT technologies in optimizing factory operations. The primary contribution is the development of a comprehensive system that integrates real-time data collection, predictive maintenance, and inventory management to improve operational efficiency and reduce manual intervention.

The system offers real-time monitoring of factory operations, where inventory movements and machine statuses are tracked and displayed on a user-friendly dashboard. This enables factory managers to gain actionable insights into the state of inventory and production without relying on manual observation. By providing data-driven decision-making tools, the system ensures that factory operations are more efficient and less prone to human error, ultimately enhancing overall productivity.

Another key contribution of the project is the automation of inventory management and maintenance tasks. Through the integration of predictive maintenance, the system continuously monitors machinery performance, predicting potential failures before they occur. This proactive approach helps to minimize downtime, extend the lifespan of equipment and ensure smooth operations. Furthermore, by automating inventory tracking and stock management, the system allows for better control over inventory levels and hence reducing the risks of stockouts or overstocking.

The integration of a notification system ensures that factory managers are alerted to any anomalies through Telegram, such as environmental conditions, machine failures or inventory shortages. This system enables managers to respond swiftly to critical issues to improve their ability to make timely decisions and take corrective actions, even when they are not on-site.

Additionally, the system's real-time data visualization capabilities offer factory managers the tools they need to monitor KPIs, such as inventory levels, machine performance, and environmental conditions. These insights support informed decision-making and help managers optimize their workflows for better efficiency.

Overall, this project contributes to the evolution of smart factories by offering an IoT-based solution that enhances operational efficiency, predictive maintenance, and real-time decision-making. By providing a flexible and scalable platform, it ensures that factories can

adapt to future needs and technological advancements. It will become a valuable contribution to the adoption of Industry 4.0 technologies.

1.5 Organization of the Report

The following chapters explore further details of this project. Chapter 1 introduces the project. It covers the problem statement, motivation, project objectives, scope, and expected contributions. It provides a clear understanding of the project's purpose and its significance in the context of smart factory inventory management. In Chapter 2, an extensive review of existing literature is conducted. The software and hardware technologies that might be employed throughout the project will be covered. Relevant systems and research studies will be reviewed to provide insights into the technologies and methods used in this project.

Chapter 3 describes the methodology used in the development of the IoT-based inventory management system. It outlines the development model, including the hardware and software tools employed, the functional requirements and the project milestone of the project. The System Design chapter elaborates on the detailed architecture and design of the inventory management system. It presents the flow of data, the integration of different system components, and the database and GUI design. The functional modules and their interactions are explained to give a clear understanding of how the system operates. The last chapter of Conclusion will conclude what has been done throughout the project.

1.6 Concluding Remark

This chapter introduced the fundamental concept of the project by identifying the current issues in inventory management, highlighting motivation, and setting the objectives the project aims to achieve. It also defined the scope of the project to ensure a clear direction during the development process. Furthermore, the expected contributions of the project were mentioned to emphasize how the IoT-based inventory management system can enhance the efficiency of factory operations, reduce human error and support better decision-making through real-time monitoring and automation. This chapter provided a solid foundation for understanding the purpose and direction of the system as detailed in the subsequent chapters.

Chapter 2

Literature Review

2.1 Review of the Technologies

2.1.1 Hardware Platform

In the context of IoT and smart factory applications, several hardware platforms are commonly used to support systems developed in this project. Although this project is software-based and runs entirely on a PC, understanding suitable hardware options is essential for real-world deployment.

Raspberry Pi

The Raspberry Pi is a compact, cost-effective computer widely used in education, prototyping, and IoT applications. It supports multiple programming languages including Scratch, Python, and Java [4].

Some of its key features include a 64-bit quad-core processor, RAM options ranging from 2 GB to 8 GB, dual-band 2.4/5.0 GHz wireless LAN, Bluetooth 5.0, Gigabit Ethernet, USB 3.0 ports, and Power over Ethernet (PoE) capability. The board's modular wireless LAN and Bluetooth certification allows it to be embedded in commercial end-products with minimal additional compliance testing, save both time and cost during development [5].

In this system, the Raspberry Pi may serve as the host for the Mosquitto MQTT broker and handles the deployment of Node-RED to enable seamless data communication and visual flow-based programming for the IoT setup.

Arduino

Arduino is an open-source electronics platform designed for building interactive and embedded system projects. It combines a microcontroller board with an easy-to-use development environment. Arduino uses a simplified version of the C or C++ programming language and provides an integrated development environment (IDE) for writing and uploading code to the board [6].

One of the most widely used models is the Arduino Uno Rev3, which is based on the ATmega328P microcontroller. The Arduino Uno Rev3 comes equipped with 14 digital General-Purpose Input/Output (GPIO) pins for connecting external components such as LEDs, buttons, or actuators. Unlike Raspberry Pi, it also features 6 analog input pins that are capable of reading data from analog sensors. The board operates on 5V and can be powered via a USB cable or an AC-to-DC adapter. Programming and uploading code are straightforward using the Arduino IDE and a standard USB connection to a computer [7].

Arduino Uno Rev3 could be integrated into future enhancements to enable interaction with physical sensors, control systems, or IoT devices using platforms such as Node-RED or MQTT.

NodeMCU

NodeMCU is an open-source IoT development board built around the ESP8266 Wi-Fi chip, which is produced by Espressif Systems. This board is widely used for IoT projects due to its low cost, ease of use, and built-in Wi-Fi capability. It supports various development environments such as Arduino IDE, Lua, and MicroPython [8].

The board runs on 3.3V and can be powered via a micro-USB port. It includes 32 KB instruction RAM, 80 KB user-data RAM, and around 200 KB of flash memory. NodeMCU provides 17 GPIO pins. Its pin multiplexing capability allows a single GPIO to serve multiple purposes which has enhanced flexibility [9].

With features of on-board voltage regulation, a 10-bit ADC and a range of digital and analog pins, NodeMCU enables seamless sensor and actuator integration. NodeMCU would be a suitable hardware platform for real-world deployment of Node-RED-based IoT systems due to its compatibility, versatility, and ability to host MQTT brokers or connect to external ones [10].

2.1.2 Database

MongoDB

MongoDB is an open-source, NoSQL database designed for flexibility and scalability and it is commonly used in modern applications. It stores data in JSON-like documents, which allows for dynamic schemas that means each document can have a unique structure. This schema-less approach provides agility in development and makes it easy to adapt and modify data models over time [11].

Unlike traditional SQL databases, MongoDB uses collections and documents instead of tables and rows, which simplifies the data structure and enhances performance, especially for large volumes of data. It also supports horizontal scaling which means it can distribute data across multiple servers and hence ensures high availability and performance even as the application grows.

MongoDB is widely used for real-time applications, such as web and mobile apps due to its ability to efficiently manage large datasets. Its built-in support for JSON data makes it a natural fit for integration with other modern technologies such as Node- RED. It offers seamless experience for developers who need to store and retrieve data quickly and flexibly.

MySQL

MySQL is a popular open-source relational database management system (RDBMS) developed by MySQL AB and later acquired by Oracle Corporation. MySQL uses SQL (Structured Query Language) for querying and managing databases and it is well- recognized for its performance, scalability, and ease of use. It is widely used in a variety of applications, ranging from small web projects to large-scale enterprise systems.

MySQL is ideal for projects that value quick data reading and ease of setup and usage. Smaller projects, rapid prototypes, or applications where prompt deployment is essential are frequently the ones that employ it. MySQL is adaptable to workload because of its variety of storage options, such as InnoDB for transactions and MyISAM for managing many reads simultaneously [12].

InfluxDB

InfluxDB is an open-source time-series database specifically chosen for its ability to store and manage data that changes over time [13]. The database organizes data by timestamp, which is crucial for analysing trends, generating reports, and enabling predictive maintenance features [14]. InfluxDB is lightweight and highly performant. These characteristics make it ideal for real-time applications. Its compatibility with Node-RED allows easy integration into the system flow and it enables developers to visualize and manipulate data without complex configurations.

2.1.3 Programming Language

JavaScript

JavaScript is a high-level and interpreted scripting language that plays a central role in web development. It was originally developed by Netscape in the mid-1990s to bring interactivity to websites, and it has since evolved into one of the most widely used programming languages in the world. Modern web-based applications are built on JavaScript, HTML, and CSS, which allows for dynamic content changes from validation, animations, and asynchronous server connection [15].

JavaScript's versatility and multi-paradigm nature are its advantages. Since it supports imperative, functional, and object-oriented programming styles, developers may select the best approach for their project. Although it may occasionally result in unexpected behavior if not carefully handled, JavaScript is also dynamically typed since its variables do not require a fixed data type, which makes the language simple to learn and rapid to develop.

Over time, JavaScript has expanded far beyond the browser. Technologies such as Node.js allow JavaScript to run on servers and enables full-stack development with a single language. The language has a vast ecosystem of libraries and frameworks that streamline development and improve efficiency.

In Node-RED, JavaScript is embedded within function nodes to write custom logic. This integration lets developers manipulate data, interact with APIs, and control automation flows within the visual programming interface. JavaScript is a powerful tool for frontend and backend development, as well as for automation, IoT and other real-time systems because of its ease of use, event-driven model, and large community support [16].

Python

Python is a high-level, general-purpose programming language. Python supports a wide range of applications. One of Python's strongest advantages is its extensive standard library and rich ecosystem of third-party libraries that support areas such as machine learning, data visualization, web development, robotics, and scientific computing.

Python's syntax is intuitive and beginner friendly. Python supports rapid development and prototyping, especially in fields including machine learning, natural language processing,

and data analytics. Popular libraries like NumPy, Pandas, TensorFlow, PyTorch, and Matplotlib provide researchers and developers with ready-to-use tools [17].

Integration of Python and Node-RED improves automation and IoT applications. Node-RED is good in managing data streams, constructing workflows, and integrating devices or APIs while Python provides robust frameworks for activities such as Artificial Intelligence, machine learning, and data analysis which are all essential components of Industry 4.0. While Python manages complicated processing, predictive modelling, or AI integration, Node-RED can handle device connectivity and data flow. This collaboration helps systems make intelligent, automated decisions by transforming raw data into useful insights [18].

Structured Query Language (SQL)

SQL is a standardized programming language used for managing and manipulating relational databases. It enables users to interact with databases to perform various tasks such as querying data, creating or modifying database structures, inserting, updating, and deleting records, and managing database access permissions. SQL is essential to a wide range of RDBMS including MySQL, PostgreSQL, Oracle, and SQL Server, which use it to store, retrieve, and manage data efficiently. SQL operates through a series of commands categorized into three main types which are Data Definition Language (DDL) that used to define and manage database structures, Data Manipulation Language (DML) that used for querying and modifying the data within a database and Data Control Language (DCL) which is responsible for managing access to the database [19].

A significant strength of SQL is its declarative nature. Users specify what data is needed or how it should be manipulated, rather than focusing on the steps required to perform those actions. This makes SQL a powerful and flexible tool for both developers and database administrators.

In Node-RED, SQL can be integrated using specialized nodes such as node-red-node-mysql or node-red-node-postgres. These nodes enable users to send SQL queries from Node-RED to a relational database and it allows for operations such as retrieving data, updating records, and performing other database tasks as part of the automation flows. Once the SQL queries are executed, the results can be processed, displayed, or used to trigger further actions in the flow [20].

2.1.4 Summary of the Technologies Review

Hardware platform

Although this project is software-based and runs on a PC, it is still important to explore hardware platforms suitable for real-world IoT and smart factory applications.

Raspberry Pi is a compact, cost-effective computer with features like a 64-bit quad-core processor, Wi-Fi, Bluetooth, Ethernet, and USB 3.0. It supports multiple programming languages and can host the Mosquitto MQTT broker and Node-RED for IoT workflows.

Arduino Uno Rev3 is an open-source microcontroller board using the ATmega328P. It offers 14 digital and 6 analog pins, operates on 5V. It is ideal for prototyping with sensors and actuators using the Arduino IDE.

NodeMCU is a low-cost, Wi-Fi-enabled development board based on the ESP8266 chip. It supports multiple protocols and development environments. With its built-in Wi-Fi and GPIO versatility, it is well-suited for deploying Node-RED and MQTT-based IoT systems.

Database

MongoDB is a flexible and scalable NoSQL database. It is ideal for applications with dynamic or unstructured data. It stores data in JSON-like documents and allows each document to have its own structure. This schema-less design allows for rapid development and easy adaptation of data models. MongoDB also supports horizontal scaling which makes it suitable for handling large datasets and real-time performance in applications. However, its lack of relational integrity can be a disadvantage when performing complex queries and managing consistency across large datasets can be challenging.

MySQL is a RDBMS that uses SQL for querying and managing structured data. It is well recognized for its performance, ease of use, and scalability. These characteristics make it suitable for both small and large-scale applications. MySQL is ideal for projects requiring strong data consistency and transaction support.

Nevertheless, NoSQL databases such as MongoDB are more effective at managing unstructured data and scaling to handle very huge datasets or complex queries.

InfluxDB is a time-series database designed to manage data that changes over time. It organizes data by a timestamp which makes it perfect for applications that require real-time analysis. InfluxDB is optimized for high write loads and fast data retrieval. Its seamless integration with platforms such as Node-RED is another advantage. However, it is specifically designed for time-series data and lacks the relational capabilities of MySQL and hence it is less suitable for non-time-series data or complex relational queries.

Programming Language

JavaScript is a dynamic, high-level scripting language central to web development. JavaScript, which was first utilized in browsers, is now used to power backend systems through Node.js and seamlessly integrates with Node-RED through function nodes. This makes it a key language for implementing real-time logic, handling APIs, and managing data within automation and IoT workflows.

Python stands out for its simplicity, readability, and powerful libraries that make it ideal for machine learning, data analysis, and scientific computing. Its integration with Node-RED enhances automation capabilities and allows Python to handle complex data processing and AI models while Node-RED manages device communication and workflow orchestration. This integration supports intelligent decision-making in Industry 4.0 systems.

SQL is essential for managing and manipulating relational databases. It operates through DDL, DML, and DCL command categories to handle database structure, data operations, and access control. In Node-RED, SQL is used via nodes of node-red-node-mysql to query and update databases. It is suitable for real-time data monitoring, analytics, and automation.

2.2 Review of the Existing Systems/Applications

2.2.1 Using Node-RED Platform in an Industrial Environment

The system developed demonstrates a prototype of a smart industrial environment using Node-RED as the main integration and visualization platform. The core of the system involves simulating real-time industrial machine data, such as speed, operating status, temperature, and humidity using Python scripts. These simulated parameters are published to an MQTT broker, which serves as the communication backbone for data exchange.

Once the data reaches the MQTT broker, Node-RED subscribes to the data topics, processes the incoming information and displays it on a custom-designed dashboard. This dashboard allows users to monitor all machine parameters in real-time. Additionally, the dashboard supports two-way communication and enables users to send control commands back to the system. For example, users can modify the speed or operating status of the simulated machines directly from the dashboard.

The system also incorporates a data storage solution by integrating Apache Cassandra, a NoSQL database. Node-RED is configured to store the received data into Cassandra. It ensures that all machine readings are logged for historical analysis and future reference. This adds a layer of persistence to the system, supporting long-term data analysis and decision-making.

In summary, the system achieves three main tasks: simulating machine data, enabling real-time monitoring and control through Node-RED, and storing data in a scalable NoSQL database. It serves as a fully functional prototype that reflects key Industry 4.0 principles which are automation, connectivity, and data-driven operations by using low-cost and open-source tools [21].

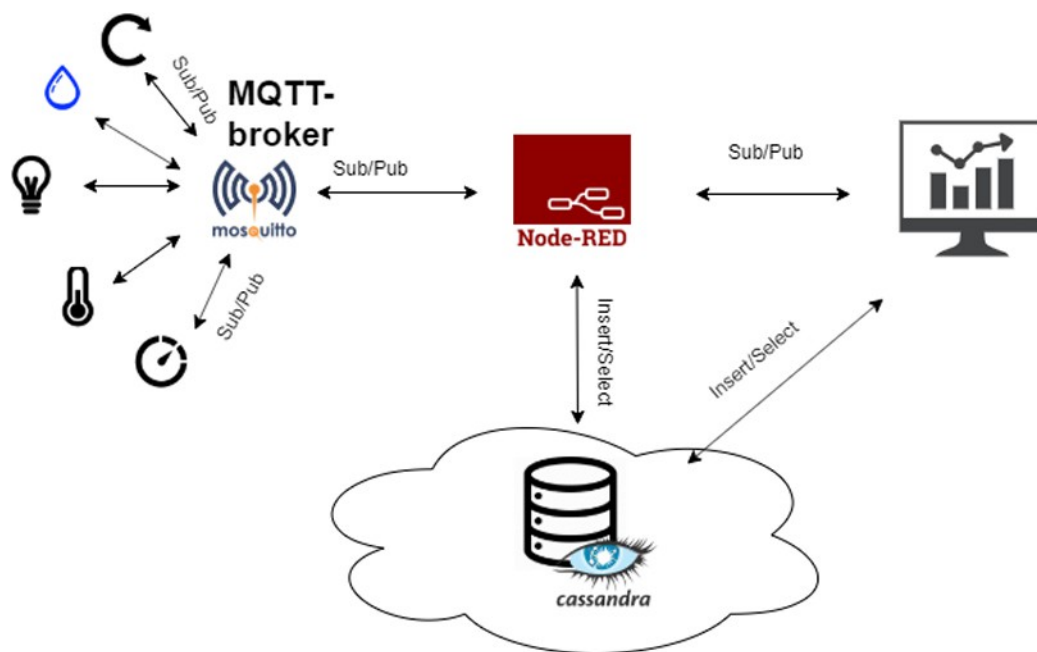


Figure 2.2.1.1: System architecture of the proposed system

2.2.2 Industrial Real-Time Digital Twin System for Remote Teaching Using Node-RED

The research paper demonstrates an innovative approach to teaching and implementing Industry 4.0 concepts, particularly focusing on Digital Twin (DT) technology. The system was built around a virtual plant developed using Factory I/O software and visualized through a Human-Machine Interface (HMI) created with Node-RED which enables effective real-time monitoring and remote learning. This integration reflects the operational principles expected in smart factory environments, where two-way communication between physical and virtual systems is essential for gathering real-time production and performance data.

The system flow begins with the Product Solicitation phase. Users interact with a Node-RED dashboard to input the desired type and quantity of stack boxes to produce. Using nodes such as node-red-dashboard, plcindustry, and node-red-contrib-ui-led, the system captures these inputs and sends the request data to the PLC through the S7 communication protocol. A request history is displayed on the dashboard, while all order details are also stored in a MongoDB database for traceability. The simple while user-friendly Node-RED interface aligns with the objectives of smart factories which emphasize the need for easy-to-use platforms to manage complex processes.

Then, the system transitions into Tracking Order Status. Within the TIA Portal, PLC variables monitor stock levels, and Node-RED reads these values in real time. The updated inventory information is then displayed graphically on the dashboard through gauge nodes. This continuous real-time monitoring of stock levels reflects smart inventory management strategies, where visibility and responsiveness are important to operational efficiency.

The most significant contribution is the development of a DT using Node-RED. Based on the research, the DT was constructed using scalable SVG vector graphics then integrated into the Node-RED environment via the Dashboard Template API. Animated elements such as moving crates and changing LED colors simulate the behavior of the actual plant. When a product moves through sensors on the plant floor, the corresponding animations update on the dashboard which offer users a synchronized virtual view of the factory floor. This flexible use of vector graphics also allows larger systems to be monitored on a single interface without compromising image quality.

The existing system's innovations provide a solid foundation for developing an IoT-Based Inventory Management System using Node-RED. The research demonstrates that effective virtual representations and real-time data handling are crucial components that can enhance inventory processes in smart factories. Adopting similar strategies from this study could lead to improved inventory tracking, forecasting, and management processes, ultimately contributing to a more integrated and responsive manufacturing environment [22].

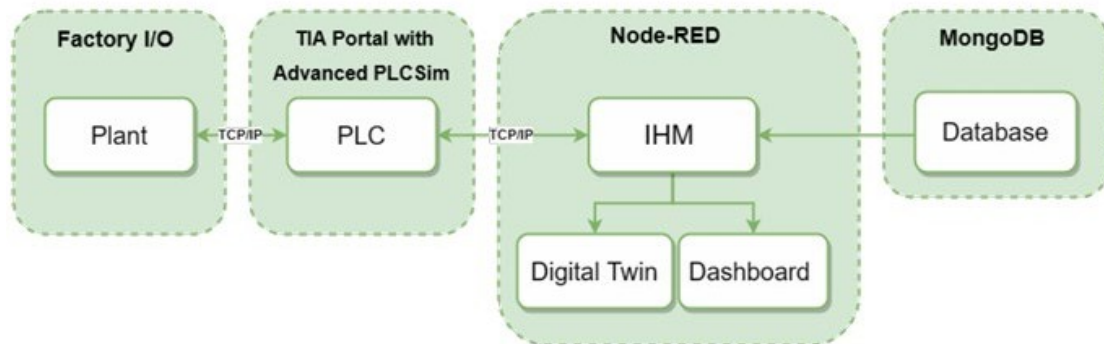


Figure 2.2.2.1: Project development diagram

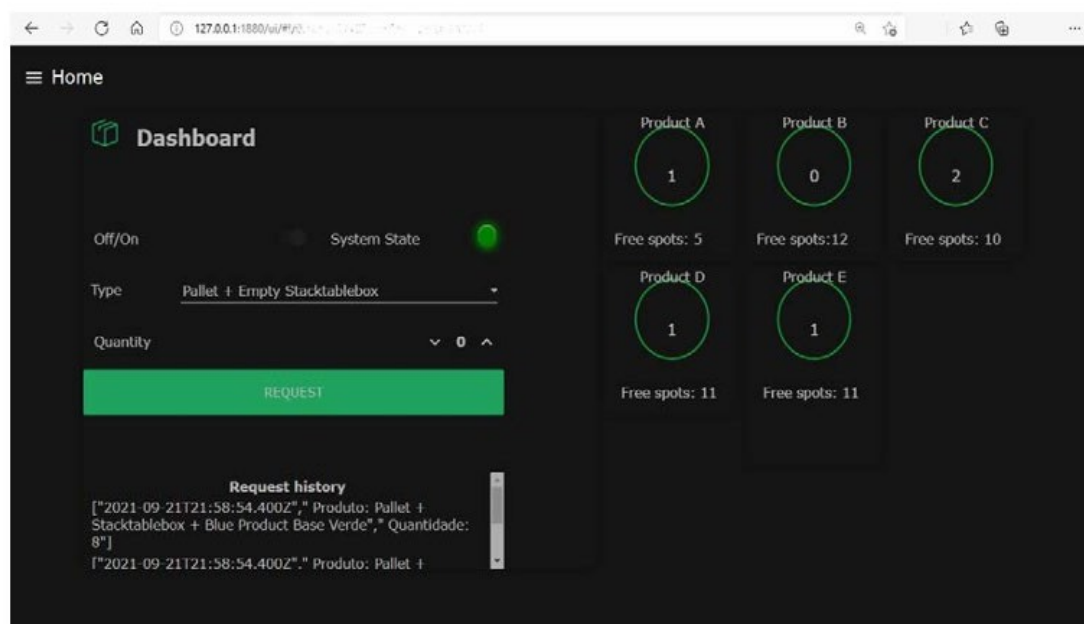


Figure 2.2.2.2: Request system on dashboard in Node-Red

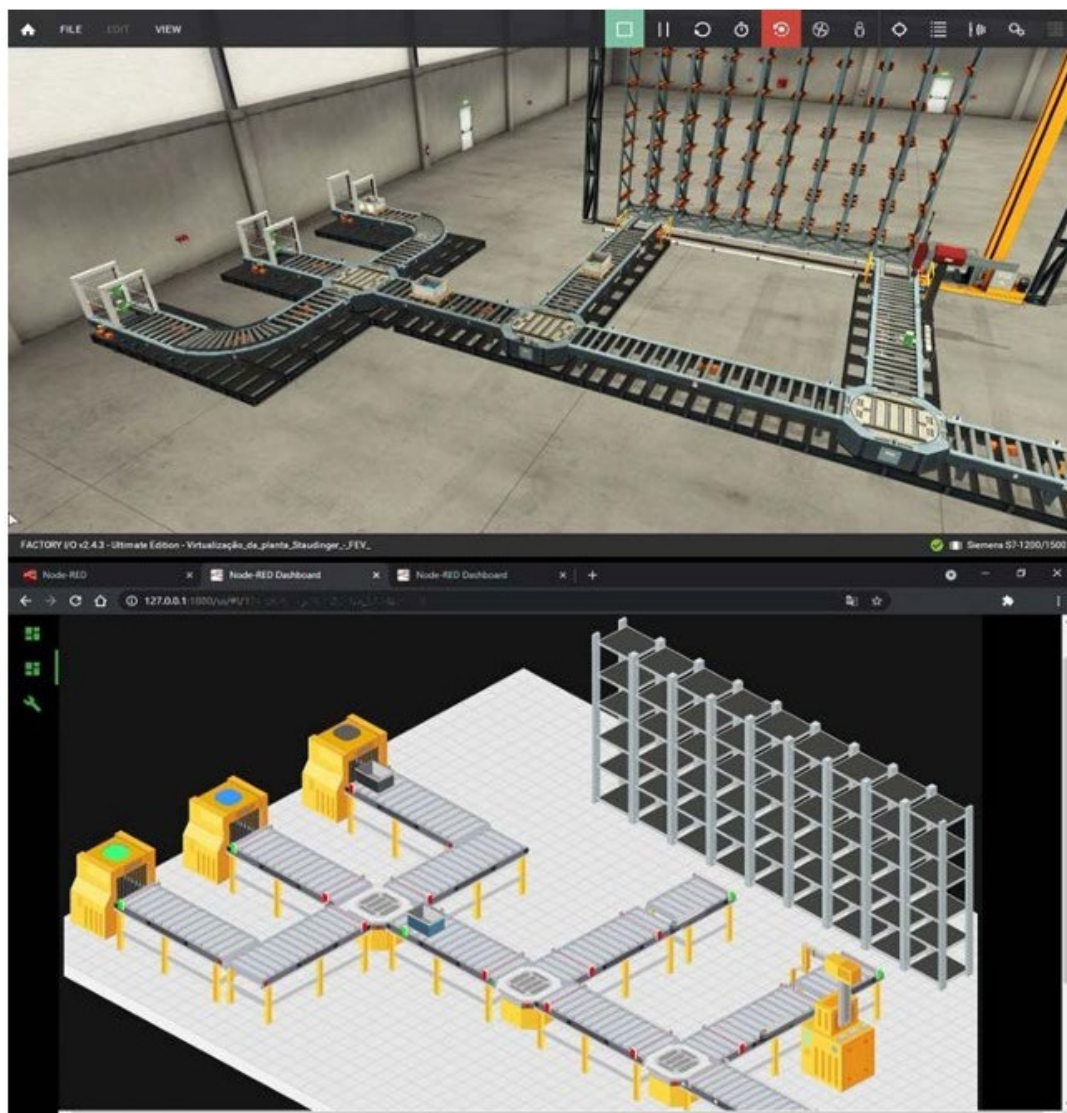


Figure 2.2.2.3: Real virtual platform and Digital Twin plan constructed with SVG images.

2.2.3 Advancing Small and Medium-Sized Enterprise Manufacturing: Framework for IoT-Based Data Collection in Industry 4.0 Concept

The research paper proposed an IoT-driven framework to support the digital transformation of SMEs toward Smart Factory environments. The framework strongly emphasizes the use of open-source, low-cost technologies to facilitate real-time data collection, processing, and system integration, particularly focus on Node-RED and the MQTT protocol. In architecture, Node-RED acts as a middleware platform, connecting IoT sensors and microcontroller devices (such as Raspberry Pi and Arduino) with backend storage systems (such as Hadoop HDFS). Its visual programming model enables rapid development and deployment without requiring extensive programming knowledge and this makes it ideal for SME environments.

MQTT is used for efficient, lightweight, and reliable data communication between microcontrollers and Node-RED which provides highly responsive operations suitable for real-time applications such as predictive maintenance and machine monitoring. The lightweight nature of MQTT, with minimal network bandwidth requirements, is particularly advantageous for SMEs that may have infrastructure limitations. Beyond simple data collection, Node-RED also enables seamless integration with MES or SCADA systems, database connectivity, and dashboard visualization. They support communication protocols such as Open Platform Communication Unified Architecture (OPC UA) for PLC interactions.

The effectiveness of the proposed system was validated through a practical deployment on an automated Argo, Food, and Beverage (AFB) production line. IoT sensors were connected to Raspberry Pi microcontrollers, with data transmitted via MQTT, processed using Node-RED, and stored in a Hadoop-based big data warehouse for further analysis. The deployment showed significant improvements in system responsiveness, real-time visibility, and the ability to perform predictive analytics, demonstrating the practical benefits of such lightweight and modular architecture.

In summary, the proposed system offers a cost-effective, scalable, and modular solution for SMEs aiming to enhance their production environments through IoT and Industry 4.0 technologies. Its strategic use of Node-RED and MQTT demonstrates how open-source tools can enable smart manufacturing capabilities and provides a strong reference model for the development of IoT-based management systems in smart factory simulations [23].

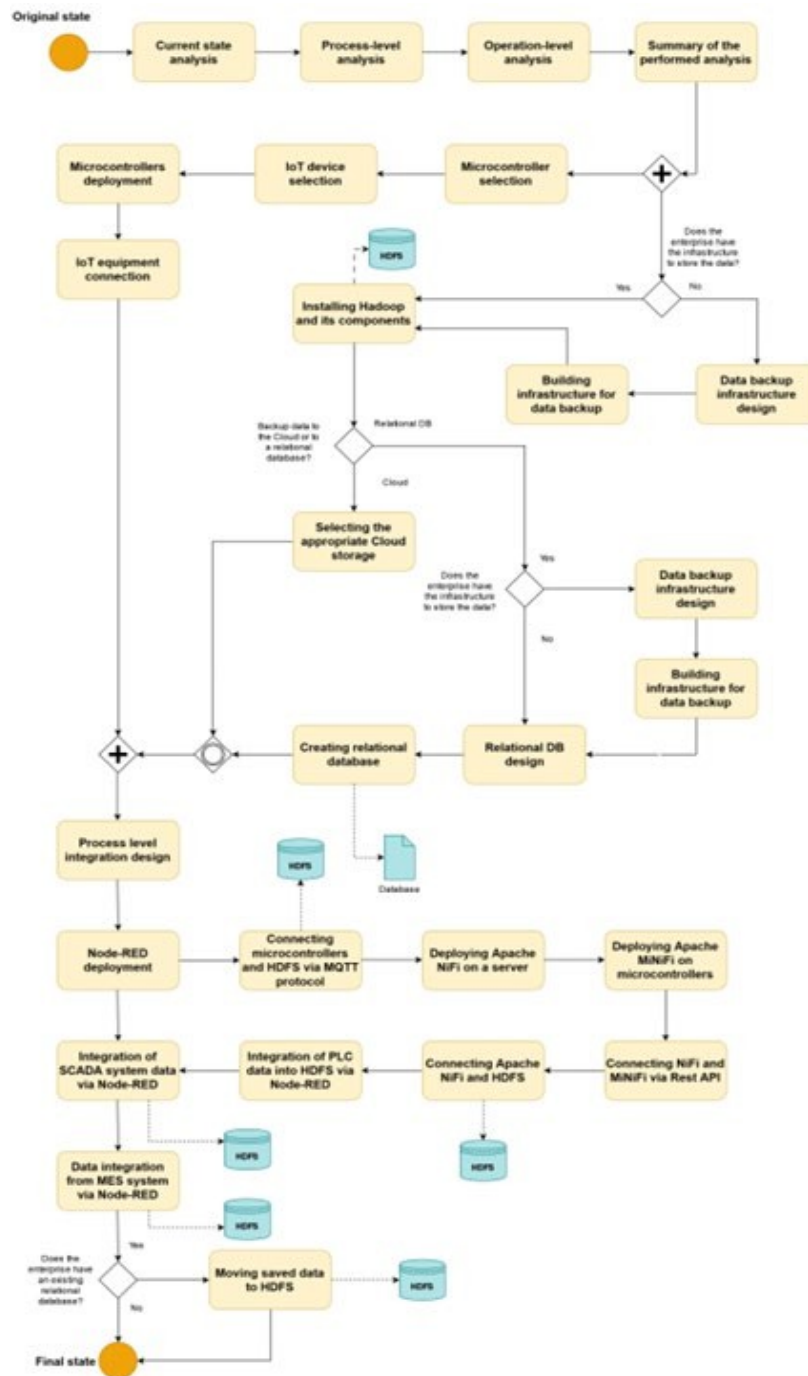


Figure 2.2.3.1: Designing the process of upgrading SMEs to the Smart Factory level

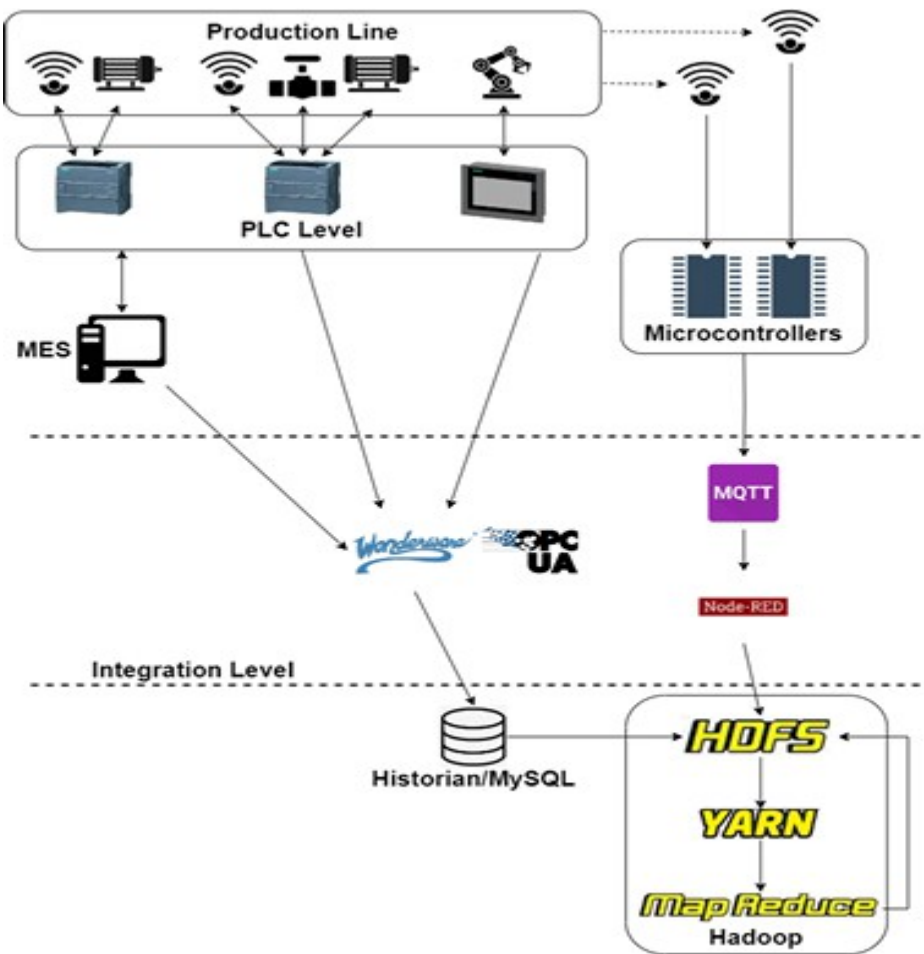


Figure 2.2.3.2: Final deployment of framework for upgrading SMEs to Smart Factory level

2.2.4 Warehousing 4.0: A proposed system of using node-red for applying internet of things in warehousing

The system proposed in the research paper titled presents an innovative approach to modernize traditional warehouse management systems through the integration of IoT technologies using Node-RED and MongoDB.

The system is designed to solve common issues found in conventional warehouse operations, such as data mismatches between actual inventory and system records, inefficient space utilization, manual labor dependency, and order fulfilment errors. The authors proposed a framework that uses RFID tags, sensors, and wireless communication technologies to automate inventory tracking and enhance accuracy.

Each product entering the warehouse is tagged with an RFID chip containing metadata such as product ID, expiry date, location, and weight. These tags are read by strategically placed readers at the gate, forklifts and shelves. After that, they are monitored by sensors to ensure that items are placed correctly, and environmental conditions are maintained. If an error or weight limit is detected, the system triggers an alarm. Node-RED is used to manage data flows, provide real-time feedback and interface with warehouse staff through visual dashboards.

The backend relies on MongoDB which is a NoSQL database for the purpose of efficient storage and retrieval of large volumes of data. The system updates inventory records in real time and allows order management by guiding forklift operators to pick items based on expiry dates to ensure optimized stock rotation. Forecasting functionalities are also implemented using real-time data, and a comparison between traditional forecasting and IoT-based forecasting shows a substantial improvement in accuracy.

In summary, this system demonstrates how Node-RED and IoT components can be applied in warehousing to achieve real-time visibility, automation, and enhanced forecasting. It provides a complete prototype that bridges the gap between digital technologies and operational logistics, and it offers a strong foundation for the development of smarter, more efficient inventory management systems. This aligns closely with the objectives of modern Industry 4.0 warehouses and has direct relevance to IoT-based smart factory projects [24].

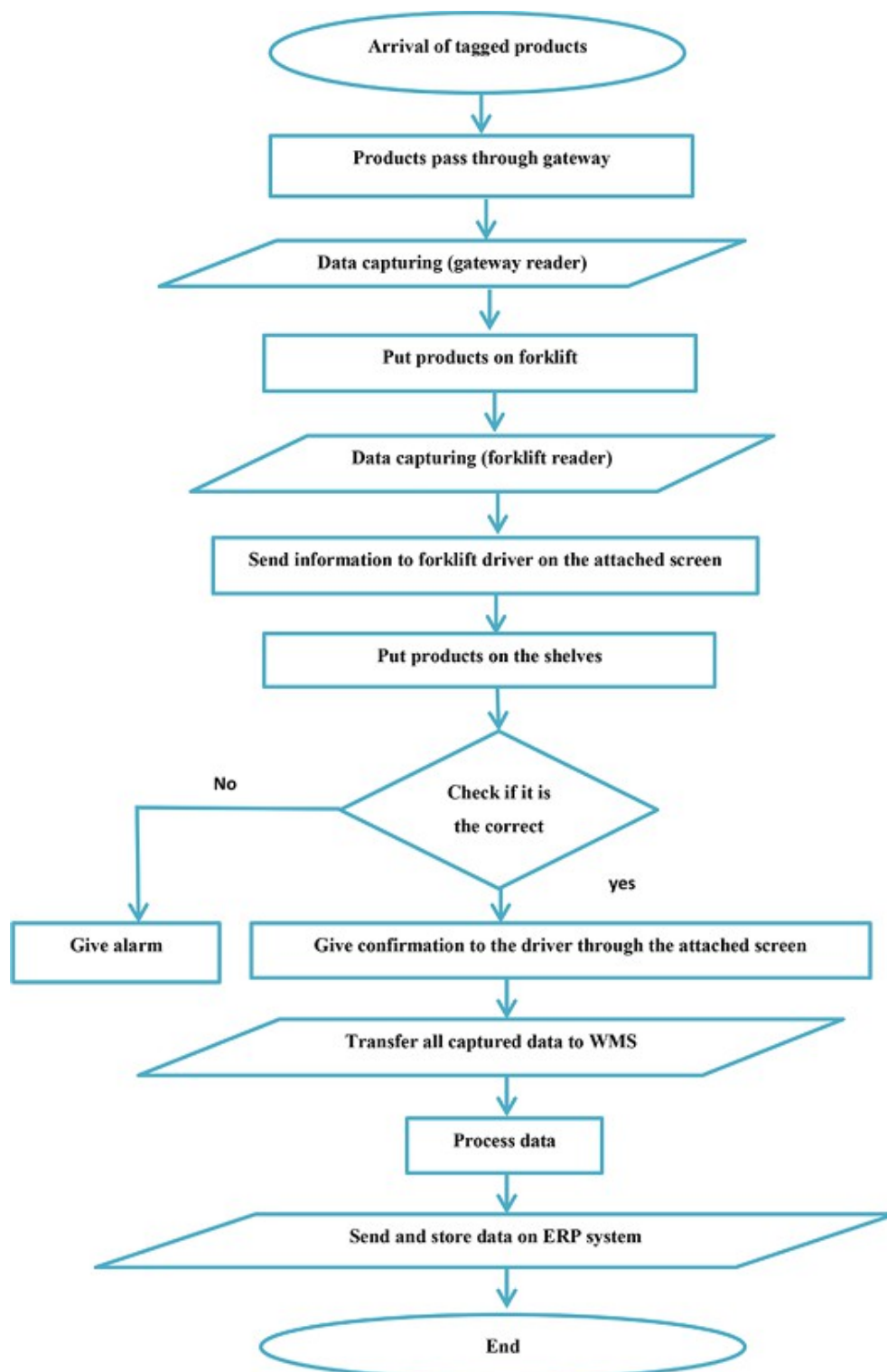


Figure 2.2.4.1: Flowchart of the proposed system

2.2.5 Summary of the Existing Systems

Existing System	Advantages	Disadvantages	Critical Comments
Using Node-RED Platform in an Industrial Environment	<ul style="list-style-type: none"> - Provide real- time monitoring for machine parameters. - Enable two-way communication. - Use open-source tools such as Node-RED, MQTT, and Apache Cassandra. - Efficient storage of large volumes of data 	<ul style="list-style-type: none"> - Limited Scalability. - No security layer. - Lacks predictive features. - Visualization is basic. 	<p>Although the proposed system is a good example of IoT integration in a smart factory context, it is quite basic and is only suitable for academic or prototype use. It presented the general concepts of Industry 4.0 such as real-time monitoring and remote control, but it is weak in scalability, security, and advanced analytics. Future improvements could include implementing predictive maintenance, stronger security measures, and enhanced visualizations to make the system more production ready.</p>

Industrial Real-Time Digital Twin System for Remote Teaching Using Node-RED	<ul style="list-style-type: none"> - Provides real-time monitoring. - User-friendly interface. - Scalability and flexibility which allows for easy integration of additional sensors and devices. - Remote accessibility. 	<ul style="list-style-type: none"> - Initial knowledge barrier (in digital twin technologies). - Limited security measures. - Primarily focuses on data collection and visualization but does not incorporate advanced predictive analytics. 	<p>The proposed system combines Node-RED's simplicity with real-time PLC data monitoring and a visually interactive Digital Twin. It successfully demonstrates how lightweight, open-source tools can achieve responsive and flexible production monitoring solutions. However, to be applied in a real industrial environment, future enhancements should address cybersecurity and error tolerance.</p>
Advancing Small and Medium-Sized Enterprise Manufacturing: Framework for IoT-Based Data Collection in Industry 4.0 Concept	<ul style="list-style-type: none"> - Provides real-time monitoring. - User-friendly interface. - Scalability and flexibility which allows for easy integration of additional sensors and devices. - Remote accessibility. 	<ul style="list-style-type: none"> - Limited built-in analytics. - Data security concerns. - Limited performance under high-load. 	<p>The proposed system offers a robust starting point for SMEs entering Industry 4.0. However, it lacks validation in larger-scale or real-world industrial environments, which may have different demands and constraints. The system can be</p>

	<ul style="list-style-type: none"> - Low cost and accessibility. - Ease of integration. - Scalable and modular architecture - Real-time data communication. 		improved by incorporating edge computing, security layers and data governance features to enhance its maturity and reliability for industrial use.
Warehousing 4.0: A proposed system of using Node-RED for applying Internet of Things in Warehousing	<ul style="list-style-type: none"> - Real-time inventory tracking. - Automation of warehouse operations. - Improves forecasting accuracy. - Cost effective and scalable. 	<ul style="list-style-type: none"> - Effectiveness heavily depends on the availability and accuracy of RFID tags and sensors. - Lacks advanced data analytics. 	The proposed system makes a valuable contribution to the concept of smart warehousing by integrating Node-RED and IoT technologies into traditional inventory operations. It successfully demonstrates how automation and real-time data collection can enhance warehouse efficiency and inventory accuracy. However, the system does not fully address enterprise-level concerns.

Table 2.2.5.1: Summary of existing system

2.3 Concluding Remark

In this chapter, the relevant technologies, platforms, and programming tools for developing the IoT-Based Inventory Management System were reviewed. The review covered essential aspects such as hardware platforms, operating systems, databases, and programming languages where they contribute to the system's functionality and performance. Existing systems and related research were also analyzed to identify the strengths and limitations of current solutions then offered critical insights into areas where improvements can be made.

Through this literature review, it helped to validate the feasibility of the proposed system design and provided important guidance for the subsequent system development phases. The knowledge and experience gained from the review ensures that the system will be developed based on proven technologies while addressing observed gaps in existing solutions.

Chapter 3

System Methodology

3.1 System Development Models

3.1.1 Rapid Application Development (RAD) Model

The Rapid Application Development (RAD) model is a software development methodology that prioritizes speed and flexibility through quick, iterative release cycles. RAD is designed to accommodate changing requirements and continuous user feedback throughout the development process [25].

RAD's key features include the use of powerful development tools such as Java, C++, Visual BASIC, and XML, as well as an emphasis on building prototypes rapidly to gather user feedback and refine the system. The model consists of 4 basic phases: Requirements Planning, User Description, Construction, and Cutover. These phases focus on gathering requirements, developing and validating prototypes, refining the system, and integrating and testing the final product.

RAD is ideal when project requirements are well-understood, timeframes are tight and continuous user input is necessary. It supports innovation and is particularly effective for small to medium-sized projects that can be modularized easily. However, it also has limitations. If the project is too complex or lacks clear modularity, RAD may not be the best fit.

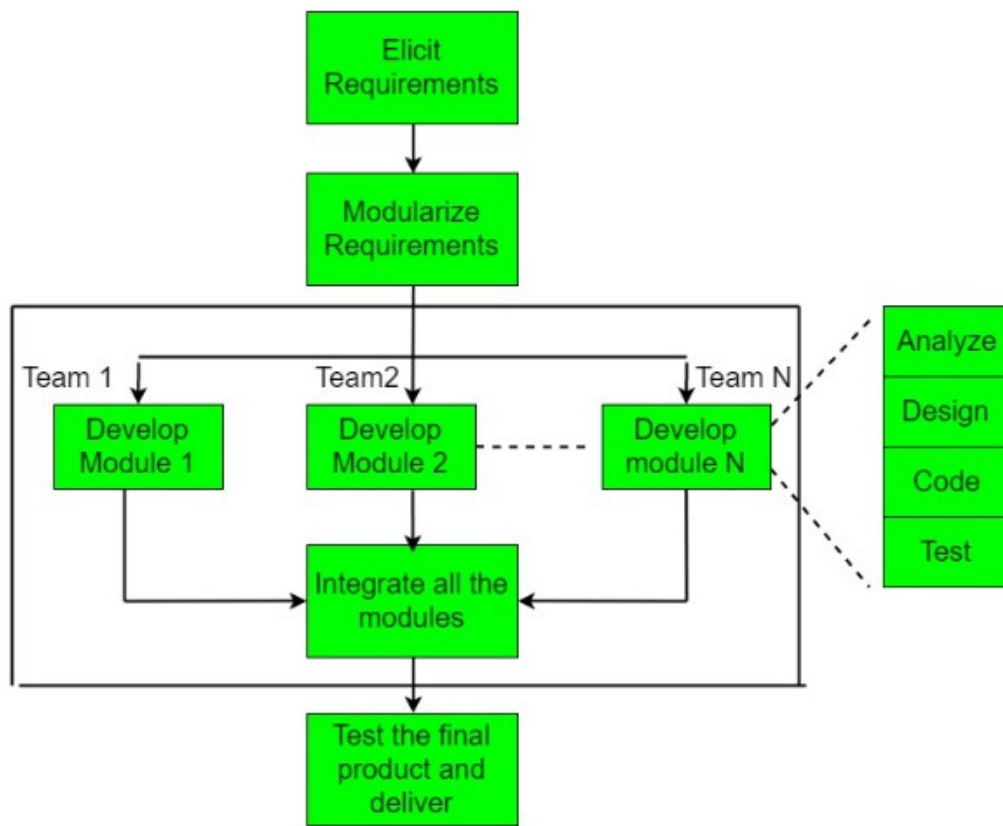


Figure 3.1.1.1: Rapid Application Development Model

3.1.2 Agile Development Model

The Agile Development Model is a modern approach to software development that emphasizes flexibility, collaboration, and customer satisfaction. Agile allows teams to adapt to changing requirements at any stage of the development process. Agile is built on iterative and incremental development, where projects are broken down into small, manageable units called iterations or sprints, each typically lasting 1 to 4 weeks. After each sprint, a working product is delivered, reviewed, and improved based on feedback.

In Agile, customer involvement is crucial. Continuous feedback from stakeholders ensures that the project stays aligned with user needs and market changes. Development teams work closely together and communicate frequently to ensure that everyone is on the same page. The Agile approach promotes faster delivery, better quality software, and more satisfaction for both developers and users. However, Agile also demands a skilled and highly collaborative team, and projects with complex dependencies may face difficulties under this model [26].

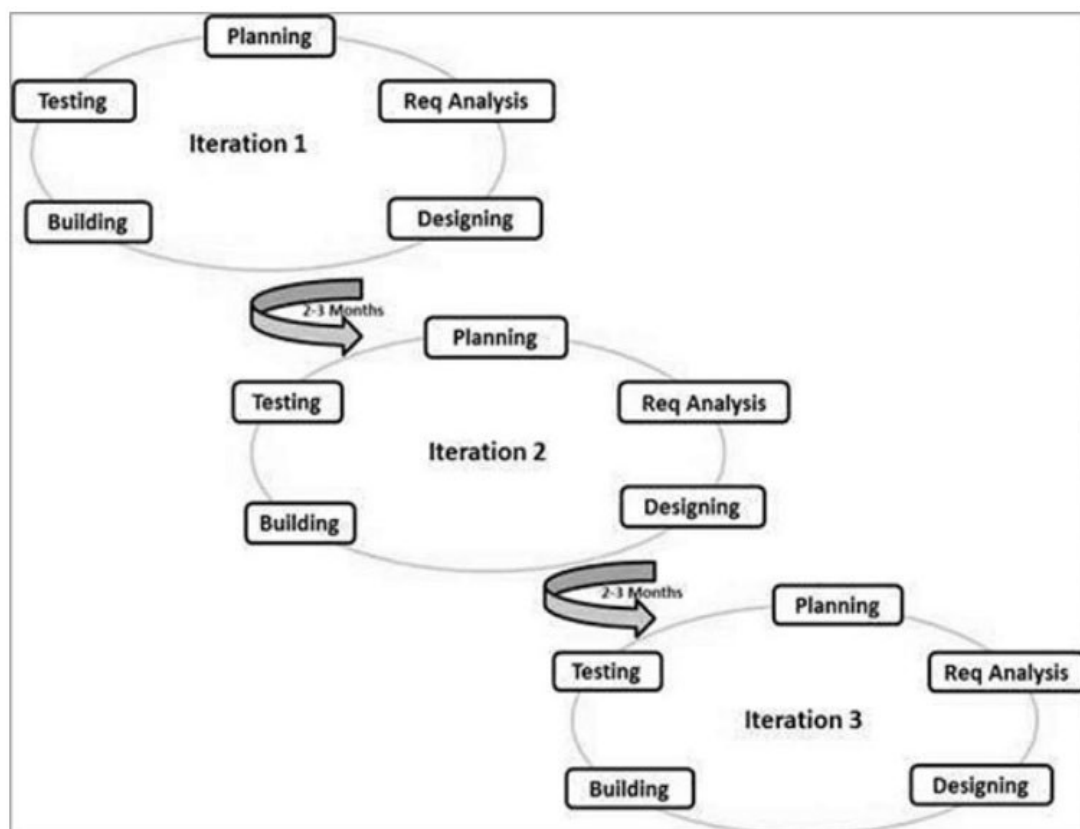


Figure 3.1.2.1: Graphical illustration of the Agile Model

3.1.3 Prototype Model

The Prototype Model is a software development approach where a prototype is built, tested, and refined repeatedly until it becomes acceptable to the users. Considering it is an iterative, trial-and-error process between developers and users, it is particularly helpful when the project requirements are not completely defined at the beginning. This method helps clarify requirements through regular feedback and adjustments [27].

The model follows six major phases. It begins with requirements gathering and analysis, where developers work closely with users to identify and understand their needs. Then the phase follows by the quick design phase, where a simple, initial design is created to offer a visual idea of the system. Based on this, the prototype is built, creating a working model of the software. After that, the prototype undergoes an initial user evaluation, where users provide feedback suggesting strengths and areas for improvement. This leads to the refining prototype phase, where changes are made based on user suggestions and iterations continue until the user is satisfied. Finally, once the prototype is approved, the product is implemented and maintained through thorough testing, deployment, and regular updates.

The advantages of the Prototype Model include early detection of errors, improved user involvement, better understanding of requirements, reduced risk of system failure, and higher user satisfaction. It encourages innovation, provides early training for users, and often leads to more successful software outcomes. However, it also comes with disadvantages, such as being time-consuming and expensive, encouraging frequent change requests, risking poor documentation, and sometimes causing users to have unrealistic expectations after seeing early prototypes [28].

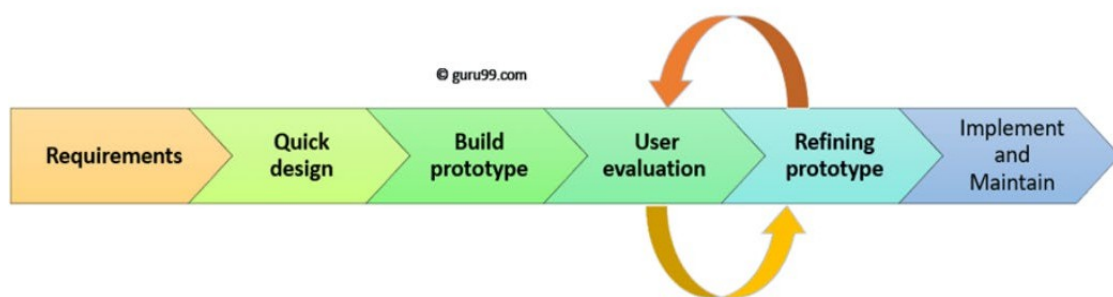


Figure 3.1.3.1: Prototype Model phases

3.1.4 V-Model (Verification and Validation Model)

The V-Model is an SDLC approach where processes are executed sequentially in a V- shape. It extends the Waterfall model by linking each development phase with a corresponding testing phase and ensures that testing is planned early and executed alongside development. Each phase must be completed before moving to the next which emphasizes discipline and structure.

In the V-Model, the left side focuses on Verification activities such as requirement analysis, system design, architectural design, and module design. These phases involve understanding customer needs, designing the system and its modules, and preparing test plans. Coding acts as the central point, bridging development and testing. The right side covers Validation activities, including unit testing, integration testing, system testing, and acceptance testing. The model ensures that each development stage is properly verified against requirements.

This model is best suited for projects where requirements are clear, stable, and unlikely to change, such as in medical or regulated industries. Although the V-Model is easy to understand and manage, it lacks flexibility. Any changes after development are costly and difficult to implement. It works well for small, well-defined projects but struggles with larger, dynamic, or long-term projects where frequent changes are expected [29].

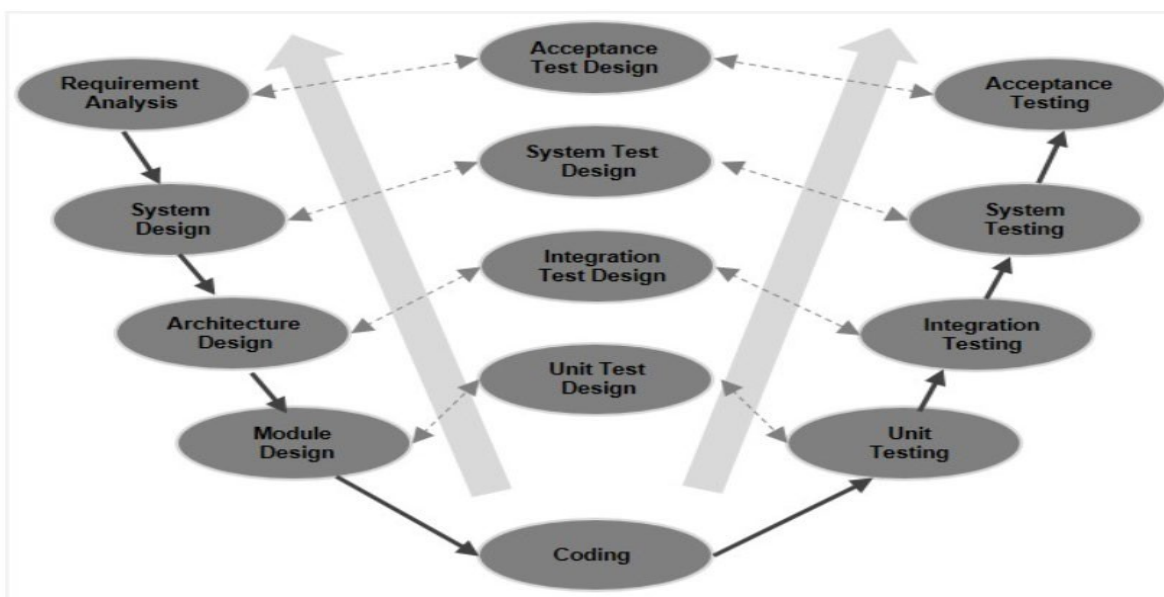


Figure 3.1.4.1: V-Model phases

3.1.5 Selected Model – Prototype Model

In developing the project, few system development models were reviewed, including Rapid Application Development, Agile, Prototype, and the V-Model. Prototype Model was selected as the most suitable model for this project. The main goal of this project is to develop and demonstrate a simulated IoT-based inventory management system for a smart factory environment using Node-RED. Since the project is simulation-based and does not involve physical sensors or hardware, the flexibility to build, test, and refine modules repeatedly makes the Prototype Model the most appropriate choice.

The Prototype Model allows the system to be developed iteratively, starting with basic flows and gradually incorporating more advanced features such as defective detection, environmental monitoring, Telegram alerts, and predictive maintenance. This model supports frequent refinement based on visual testing and simulated data outputs, which aligns well with the interactive nature of Node-RED dashboards and MQTT message flows. Each module can be built individually and revised based on how it performs in the simulation. The use of simulated data also allows the project to mimic realistic factory conditions and makes it ideal for prototyping without physical sensors.

On the other hand, the RAD Model focuses more on speed and iterative development so that it is more suitable for projects with well-defined modular structures and teams working in parallel. Since this project is developed individually with evolving requirements, RAD's emphasis on quick delivery in fixed phases may not provide the flexibility needed for continuous adjustments and visual testing.

The Agile Model offers strong support for iterative progress and collaborative development. However, Agile relies heavily on constant interaction with stakeholders and structured sprints, which is less applicable in a self-developed project where feedback is limited. Other than that, Agile is better suited for team-based environments with evolving customer requirements, rather than a single-developer prototype.

The V-Model was also considered, but it is more appropriate for projects with well-defined, fixed requirements and strict validation needs, usually applied in medical or safety-critical systems. Since this project involves simulated input, evolving designs, and flexibility to change flow logic or dashboard elements, the rigid phase-by-phase structure of the V-Model

would limit creative iteration and delay functional feedback which will greatly influence the system efficiency.

In conclusion, the Prototype Model was selected as the development model because it best supports the project's development goals: rapid iteration, easy integration of simulation elements, flexibility in testing, and continuous improvement of the user interface and functionalities. It provides a practical and adaptable approach for building a Node-RED-based smart factory simulation with full control over simulated data and conditions.

3.2 System Requirement (Technologies Involved)

3.2.1 Hardware

Laptop



Figure 3.2.1.1: IdeaPad 3 15ITL6

Description	Specifications
Model	IdeaPad 3 15ITL6
Processor	11th Gen Intel(R) Core (TM) i5-1135G7 @ 2.40GHz 2.42 GHz
Operating System	Windows 11
Graphics	Iris Xe Graphics
Memory	12GB DDR4 SDRAM
Installed RAM	8.00 GB
System Type	64-bit operating system, x64-based processor
Storage	475 GB

Table 3.2.1.1: Specifications of laptop

Smartphone



Figure 3.2.1.2: iPhone 15 Pro Max

Description	Specifications
Model	iPhone 15 Pro Max
Processor	Apple A17 Bionic APL1V02 (3nm)
CPU	2 X 3.78 GHz
GPU	Apple GPU (6-core graphics)
Operating System	iOS 18.3.2
RAM	8.00 GB, 4266 MHz
Storage	256 GB

Table 3.2.1.2: Specifications of smartphones

3.2.2 Software

Node-RED

Node-RED is an open-source development tool that enables users to design logic flows for IoT applications using a visual interface [13]. It operates through a web-based editor and allows users to easily connect different devices, APIs, and online services to build IoT solutions. By linking nodes where each represents a specific function, users can construct and manage complex workflows without extensive coding. Node-RED includes a large library of pre-configured nodes for many popular IoT platforms and services which help developers quickly create sophisticated IoT systems with minimal programming effort [30].

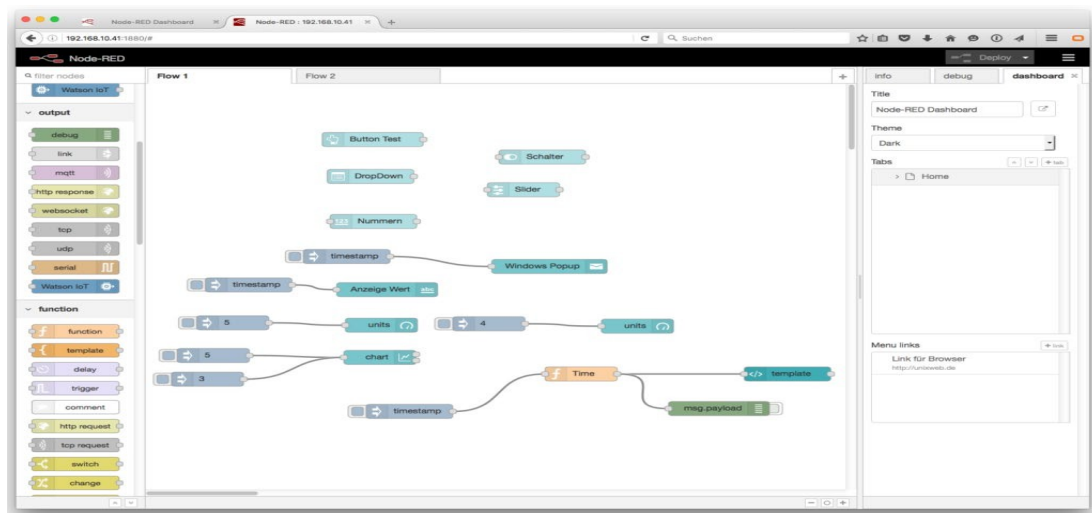


Figure 3.2.2.1: The GUI of Node-RED

Message Queuing Telemetry Transport (MQTT)

MQTT is a lightweight communication protocol designed for networks with low bandwidth, high latency, or unreliable connections. An MQTT broker serves as the central server that facilitates message exchange between MQTT clients, which can include devices, sensors, or software applications [31]. It receives messages from clients and distributes them to others according to their topic subscriptions. The broker ensures reliable and efficient delivery of messages to the appropriate clients. MQTT brokers are commonly used in IoT and Machine-to-Machine (M2M) communications, where numerous devices and sensors need to interact [4].

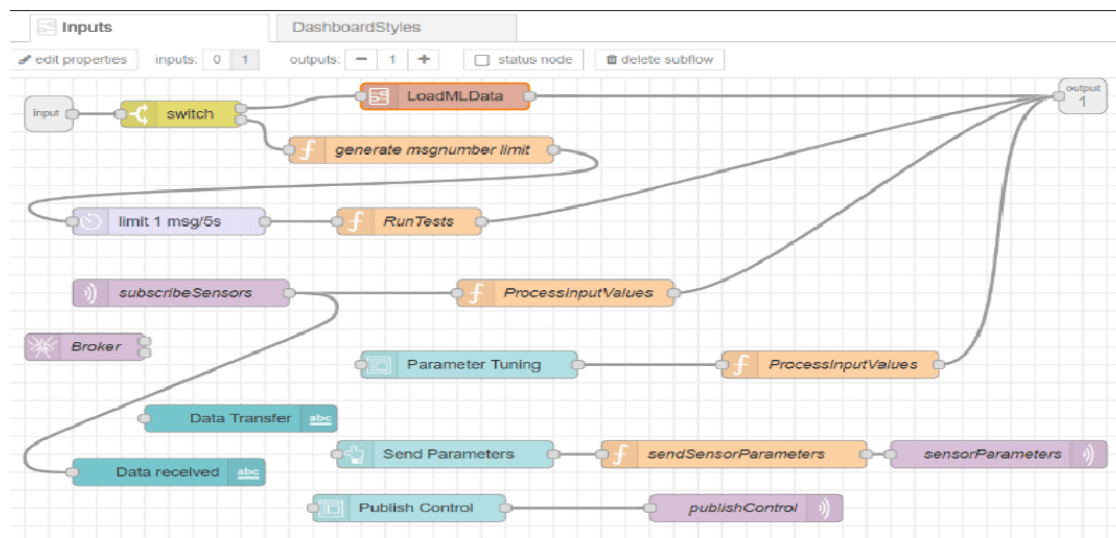


Figure 3.2.2.2: Implementation of the MQTT modules in Node-RED

InfluxDB

It will be used as the primary time-series database for storing real-time data collected. Its ability to handle large volumes of time-stamped data makes it ideal for storing continuous streams of environmental, inventory, and machinery data. For example, temperature, humidity and pressure fluctuations over time can be recorded and retrieved quickly and hence it provides insights into long-term trends and helps to identify potential inefficiencies or areas for improvement.

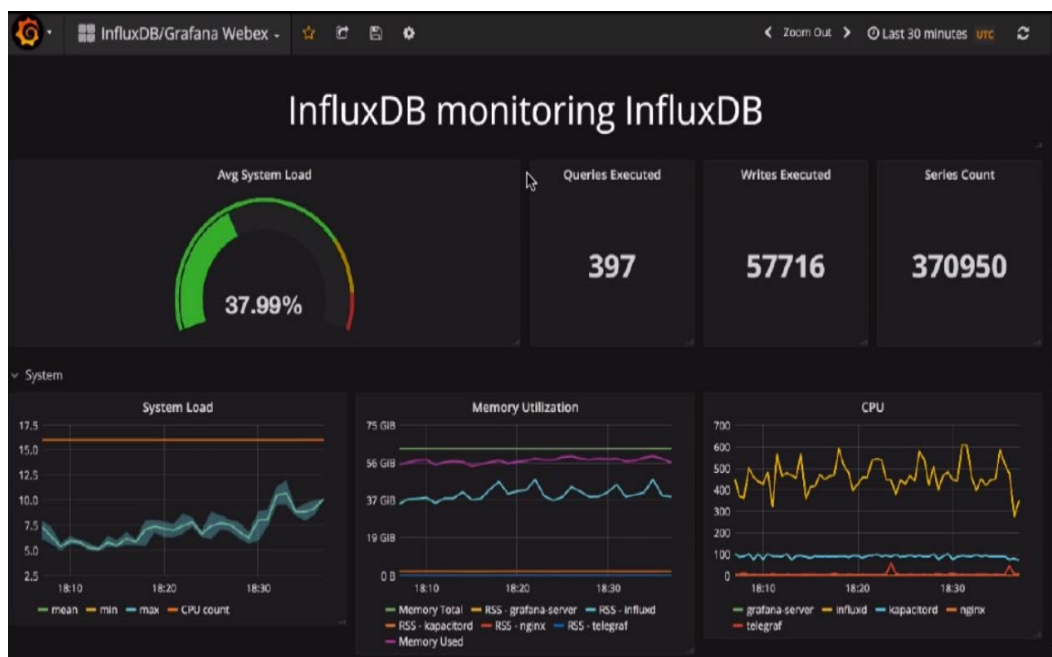


Figure 3.2.2.3: The GUI of InfluxDB

Telegram bot API

The Telegram Bot API is a set of HTTP-based tools provided by Telegram that allows developers to create and control bots on the Telegram platform. By using a unique bot token obtained from @BotFather, developers can program bots to send and receive messages, handle user commands, and interact with users in both private chats and groups [9].



Figure 3.2.2.4: The logo of Telegram Bot Father

Inkscape

Inkscape is a free and open-source vector graphics editor used to create and edit SVG files. It provides a wide range of drawing tools and supports precise control over shapes, paths, text, and colors. Inkscape is commonly used for designing diagrams, illustrations, icons, and technical drawings. Its SVG output is ideal for integration with Node-RED for interactive visualizations.

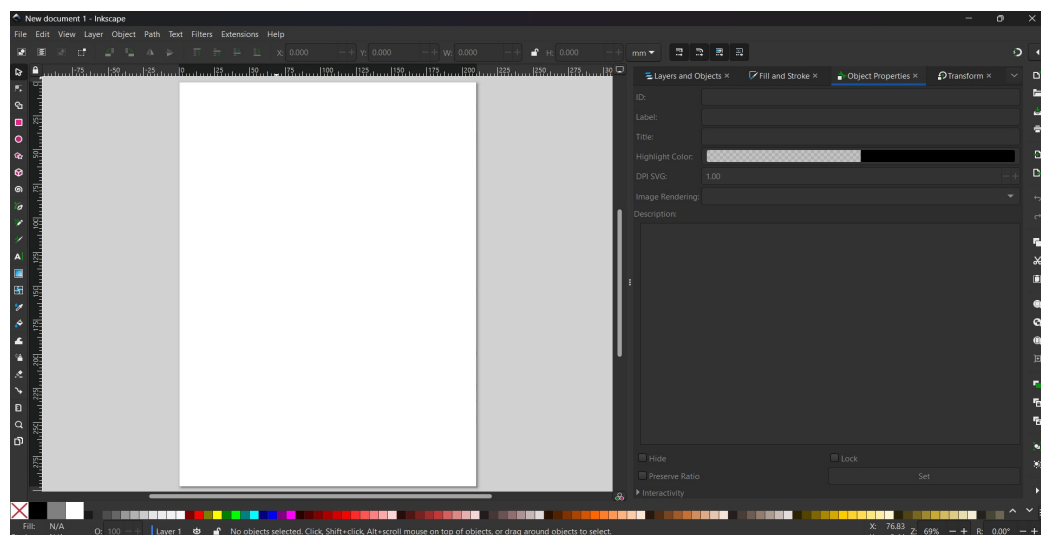


Figure 3.2.2.5: Inkscape interface

3.3 Functional Requirement

- i. **The system should monitor environmental conditions and trigger appropriate actions or alerts when values exceed defined thresholds.**

This includes simulating temperature readings to trigger real-time alerts via the Node-RED dashboard and Telegram if they exceed the safe range, automatically activating a virtual humidifier or dehumidifier to regulate humidity levels and sending high-priority alerts for abnormal environmental conditions to ensure a safe and controlled factory environment.

- ii. **The system should send real-time alerts via Telegram for all critical events.**

By using Telegram for instant notifications, the system ensures that key personnel are immediately informed of any urgent issues to enable quicker response times even when not on site.

- iii. **The system should log operational data into the InfluxDB database for historical tracking and analysis.**

This supports performance reviews, trend identification, and the development of predictive maintenance strategies based on accurate historical data.

- iv. **The system should display real-time system status and analytics on the Node-RED dashboard.**

The dashboard will visualize key metrics such as temperature, humidity, pressure, inventory levels, machine status, and maintenance indicators to provide operators with a comprehensive view of factory operations.

- v. **The system should track production success rate and provide performance insights.**

By monitoring the number of components processed, along with counts of passes and failures at each stage, the system can calculate overall production success rates. These statistics are displayed on the dashboard and accessible through Telegram commands, giving operators real-time visibility into production quality and supporting data-driven performance evaluation.

3.4 Project Milestones

FYP 1

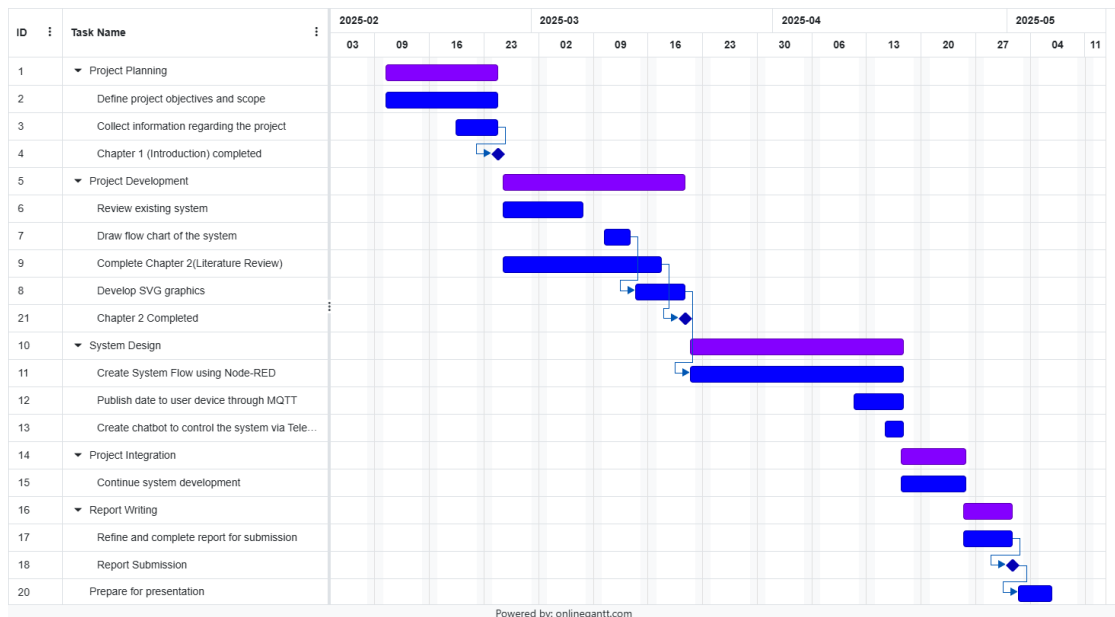


Figure 3.4.1: Gantt Chart for FYP1

FYP 2

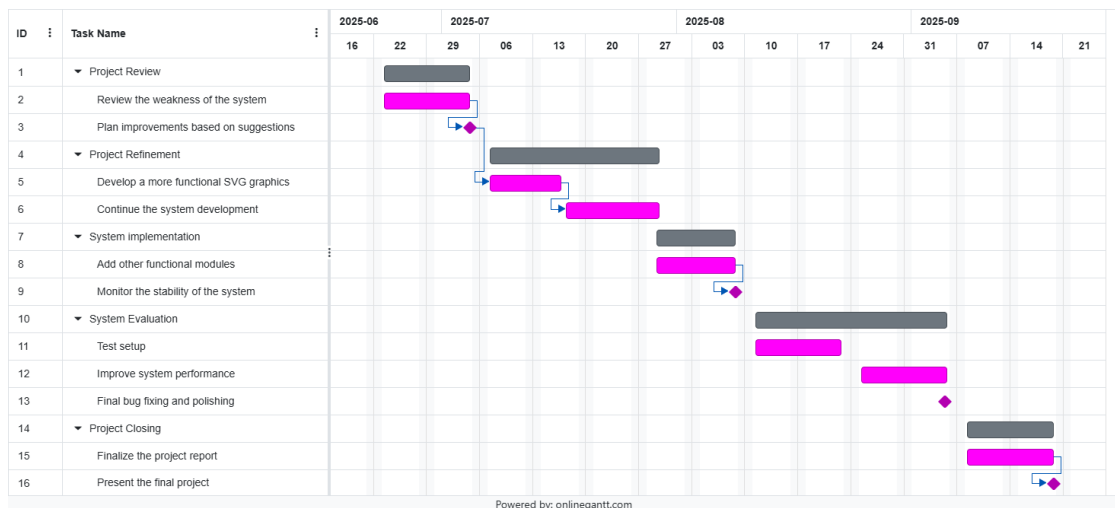


Figure 3.4.2: Gantt Chart for FYP2

3.5 Concluding Remark

The overall methodology of developing the IoT-Based Inventory Management System in Node-RED was outlined in this chapter. After reviewing a few other types of system development models, the Prototype Model was selected to be the most suitable approach. To make sure the project was headed in the right path, system requirements were identified. Additionally, a review of the technologies employed, including the software and hardware used to create the system, have been evaluated. project milestones were built. The chapter establishes the groundwork for the subsequent system design and implementation stages to ensure an organized and effective development process.

CHAPTER 4

System Design

4.1 System Architecture

The system design diagram presents the interaction of various components in an IoT- based inventory management system in the context of a smart factory.

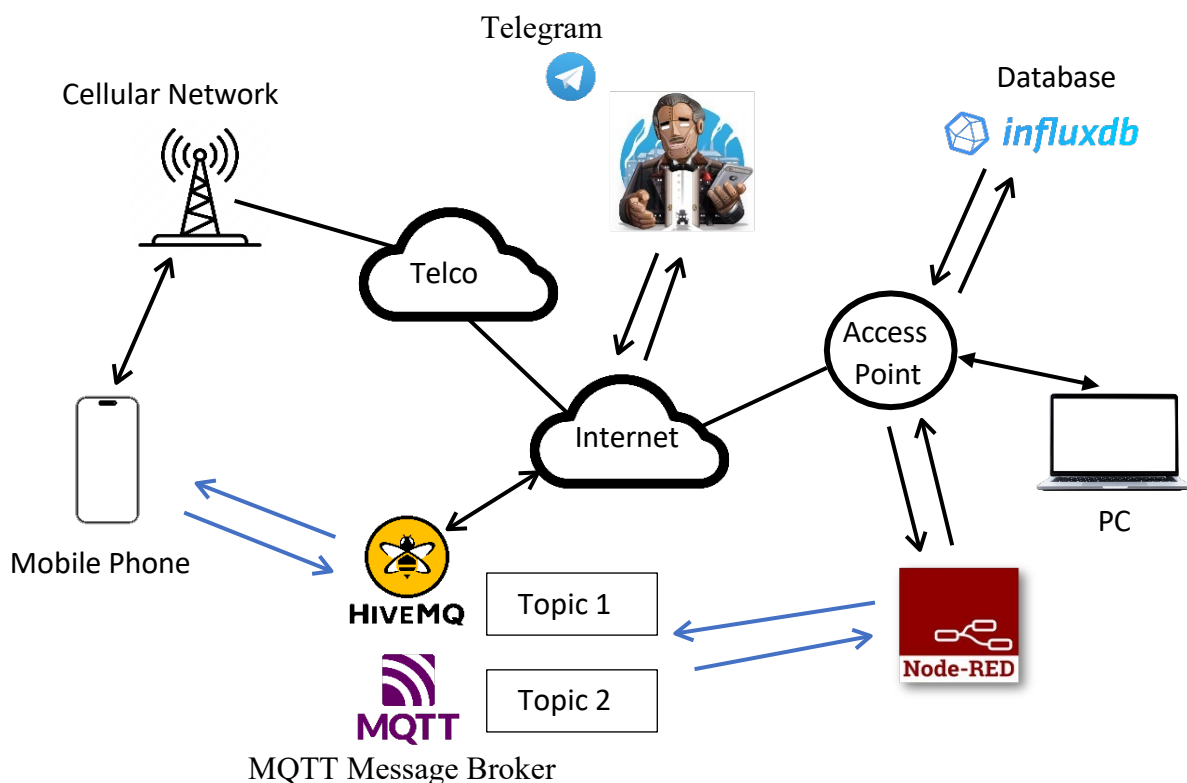


Figure 4.1.1: Architecture diagram of the project

The system architecture of the IoT-Based Inventory Management System is designed to simulate a smart factory environment by integrating Node-RED, MQTT, InfluxDB for data logging, and Telegram for real-time notifications.

The Node-RED platform is the core of the system running on a PC connected through an Access Point. Node-RED acts as the main processing and logic engine where the entire smart factory simulation process is integrated.

Communication between system components is managed through an MQTT Message Broker (HiveMQ), which operates as a lightweight publish-subscribe network protocol to ensure reliable and real-time data exchange. Node-RED both publishes and subscribes to various MQTT topics to simulate production processes and feedback within the system.

The system uses InfluxDB, which is a time-series database to store all critical data, including temperature, humidity and pressure readings. InfluxDB interacts directly with Node-RED and allows the system to log and retrieve real-time and historical data for performance monitoring and evaluation.

The system integrates with Telegram as its primary notification channel to deliver real-time alerts whenever critical conditions are detected, such as low inventory, abnormal machine performance, or maintenance needs. These alerts are triggered automatically by Node-RED and sent instantly over the Telco network to users' mobile devices, enabling them to monitor operations remotely and take quick, informed action. This integration ensures that users are always updated on the system's status and can respond effectively to maintain smooth factory operations.

4.2 Functional Modules in the System

i. Environmental Monitoring and Control Module

This module simulates environmental condition monitoring by generating random or patterned temperature, humidity and pressure values. Real-time temperature trends are displayed on the dashboard and all temperature data is recorded into InfluxDB. The same goes to humidity and pressure reading.

ii. Alert Notification Module

The alert module integrates Node-RED with a Telegram Bot to provide real-time notifications to users. It sends alerts for critical events. Alerts are delivered via Internet connection to mobile devices to make sure that users are always informed of important factory conditions even when remote.

iii. Data Logging and Analytics Module

This module manages the systematic recording of all factory activities into an InfluxDB time-series database. It ensures that the critical events in the factory are properly logged with timestamps. This data can later be used for analytics, performance evaluation, and system improvements.

iv. Inventory Management Module

This module manages all stock-related activities in the factory, including tracking, restocking, and monitoring inventory levels. It ensures that shortages and out-of-stock conditions are detected in real time, with updates reflected on the dashboard and alerts sent to users when immediate action is required.

v. Machine Performance Monitoring Module

This module oversees the performance of different machines in the factory, such as mounting, soldering, and inspection machines. It continuously tracks machine operations, detects abnormal behavior, and provides status updates. Any significant deviation in machine performance triggers real-time notifications for early intervention.

4.3 System Flow

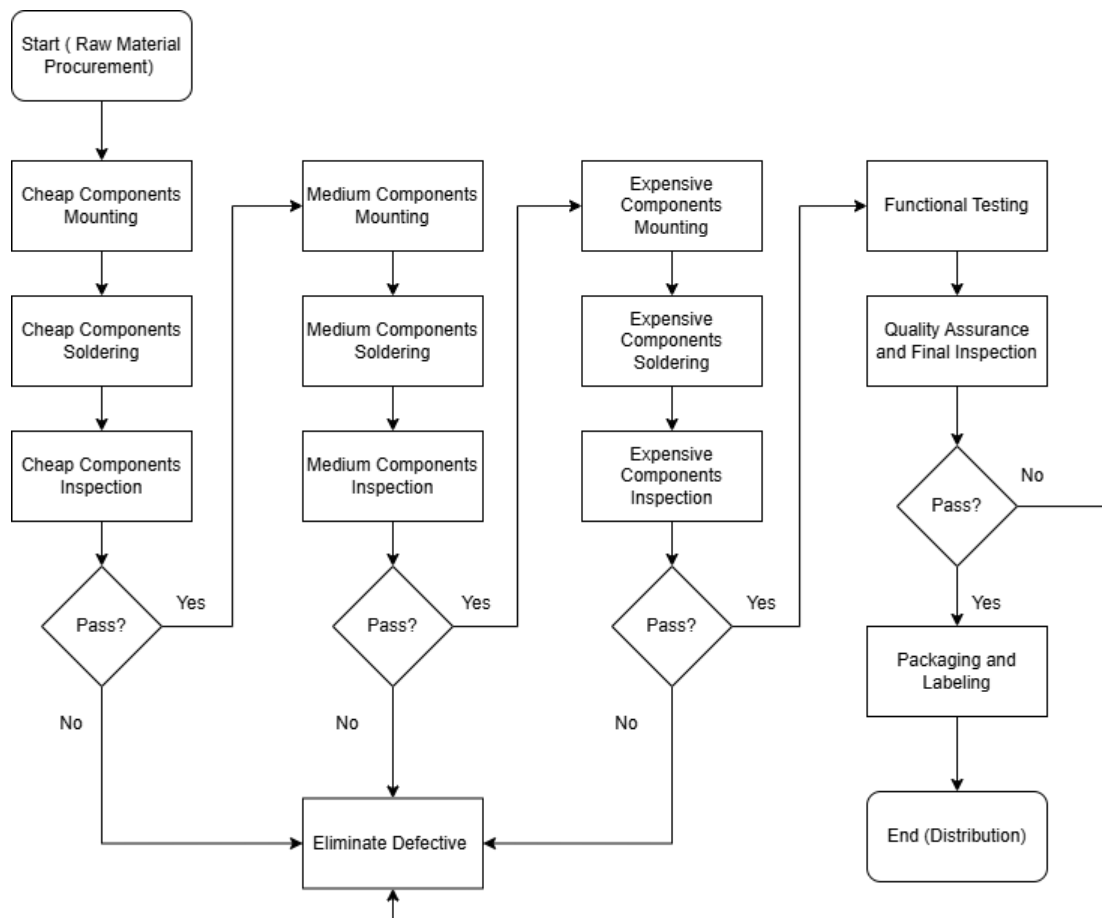


Figure 4.3.1: System flowchart of the project

The system starts with the procurement of raw materials. For cheap components, the system routes them through the first loader, where they undergo mounting, soldering, and inspection processes. During inspection, if the component passes the quality check, it proceeds to the next stage. If it fails, it will automatically route to the defective elimination path, visually updated in the SVG animation and logged into the database.

Similarly, medium components are processed by the second loader, and expensive components are handled by the third loader, each passing through their respective mounting, soldering, and inspection stages with the same pass or fail decision process.

After passing initial inspections, components converge into a shared functional testing stage, where their operational quality is further verified. Components that pass functional testing proceed to quality assurance stage and final inspection. Any components failing

functional testing or QA inspection are diverted for rejection or rework. After that, the successfully inspected and tested components are then packaged and labeled. Finally, the finished products are moved to the distribution and logistics section for dispatch.

Throughout the entire production process, the system continuously stimulates temperature, humidity, and pressure readings. If the temperature exceeds a predefined limit, the system pauses operations and triggers real-time alerts. If humidity falls below or rises above safe thresholds, the system activates either a humidifier or dehumidifier. If pressure readings exceed safe limits, a high-priority alert is sent to the factory manager via Telegram.

Machine runtime is tracked continuously. Once a machine exceeds its maintenance threshold, the system triggers a predictive maintenance alert to prompt the operator to schedule servicing. All critical events are logged into InfluxDB, and real-time dashboards are updated using Node-RED to display the system's operational status.

The system flow ensures that each stage of production, inspection, and monitoring is accurately simulated while aligning with the goals of real-time visibility, efficiency, predictive maintenance, and automated inventory management in a smart factory environment.

4.4 Database Design

The database of the inventory management system will be implemented using InfluxDB, which is a time-series database optimized for handling high-frequency data generated in the simulated smart factory environment. The database is responsible for storing all essential information. By visualizing this data, the key personnels can conveniently monitor the smart factory's environment and inventory status in real time.

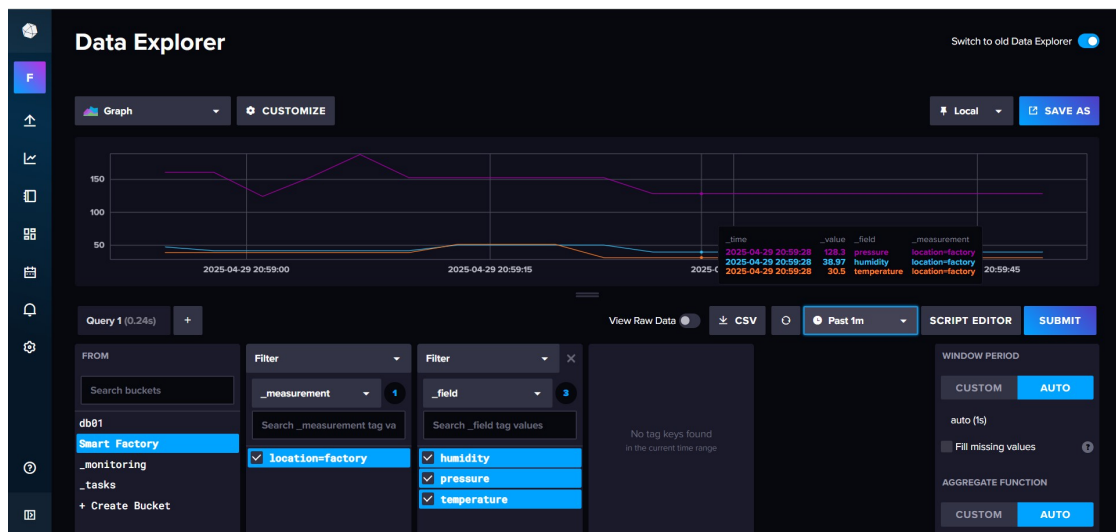


Figure 4.4.1: InfluxDB database values

4.5 GUI Design

First, SVG graphics of factory layout will be embedded into the dashboard to provide a graphical visualization of the factory's operation. The SVG will display critical areas such as loaders, mounting stations, soldering stations, inspection areas, testing stages, and packaging sections. As the simulation runs, electronic components will visually move across the SVG according to the process stages. Green indicators will show successful processing, while red indicators will represent defects detected at inspection points. The SVG will also display the environmental readings visually.

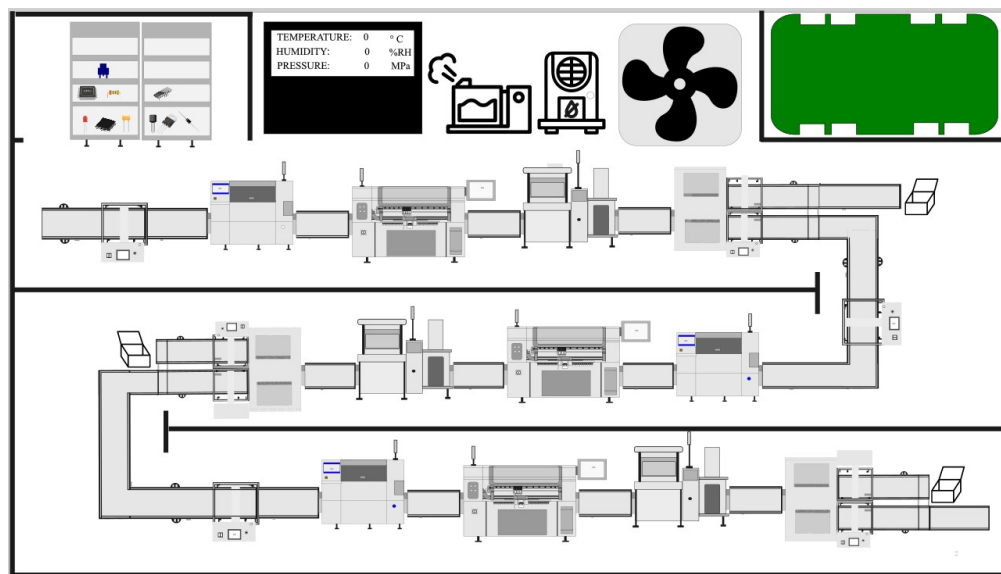


Figure 4.5.1: SVG graphics of smart factory layout (Part 1)

Logistics

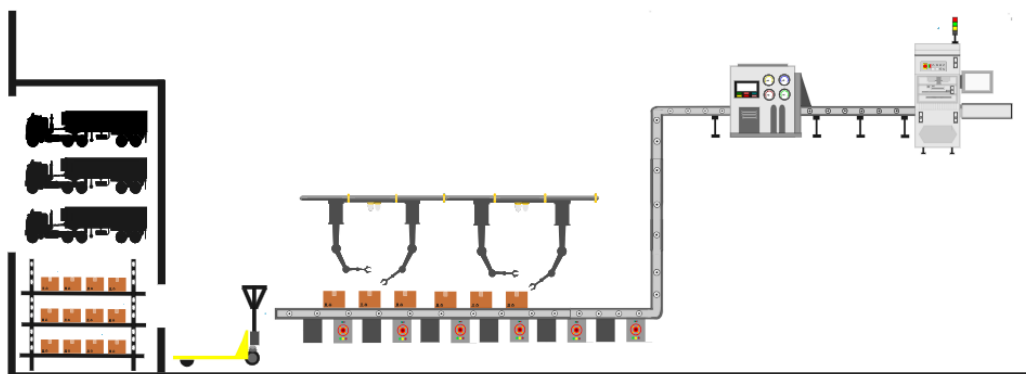


Figure 4.5.2: SVG graphics of smart factory layout (Part 2)

Besides, the dashboard will include environmental control devices such as the fan, humidifier and dehumidifier. Their operational status (on or off) will be synchronized with the dashboard so that users are always informed whether these devices are currently active. A main control section will also be provided to allow users to manually start, pause, or reset the simulation when needed.

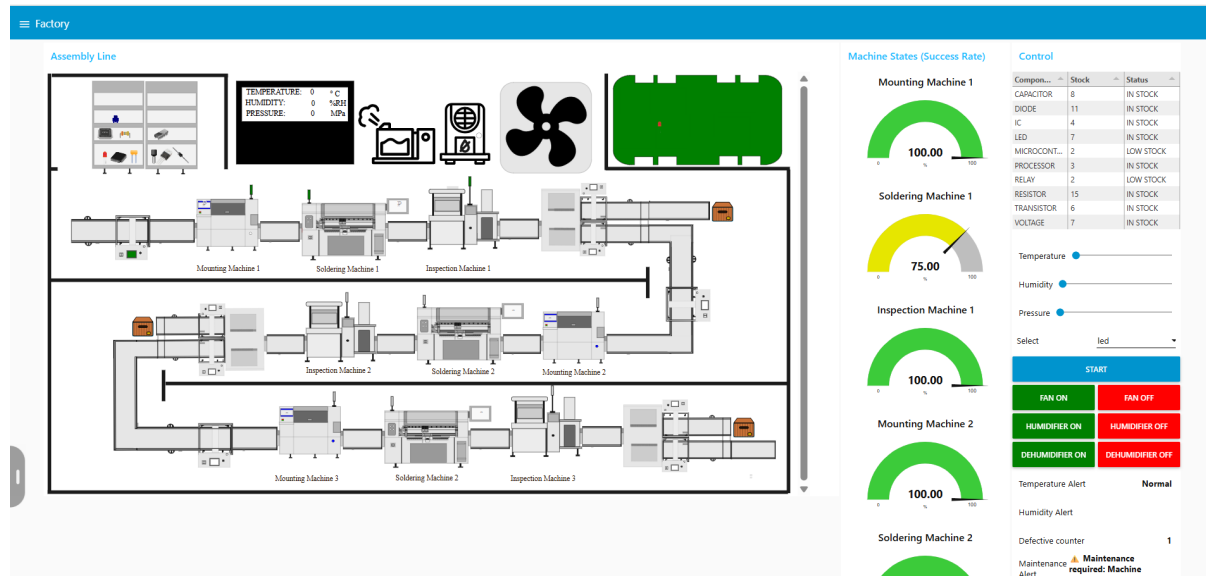


Figure 4.5.3: The Node-RED dashboard for assembly line (Part 1)

Additionally, there will be a Real-Time Monitoring Interface that displays real-time temperature, humidity, and pressure readings through dynamic graphs. Warning alerts will be sent if any environmental value exceeds the safe thresholds. This provides users with a clear and easy way to monitor environmental conditions continuously.

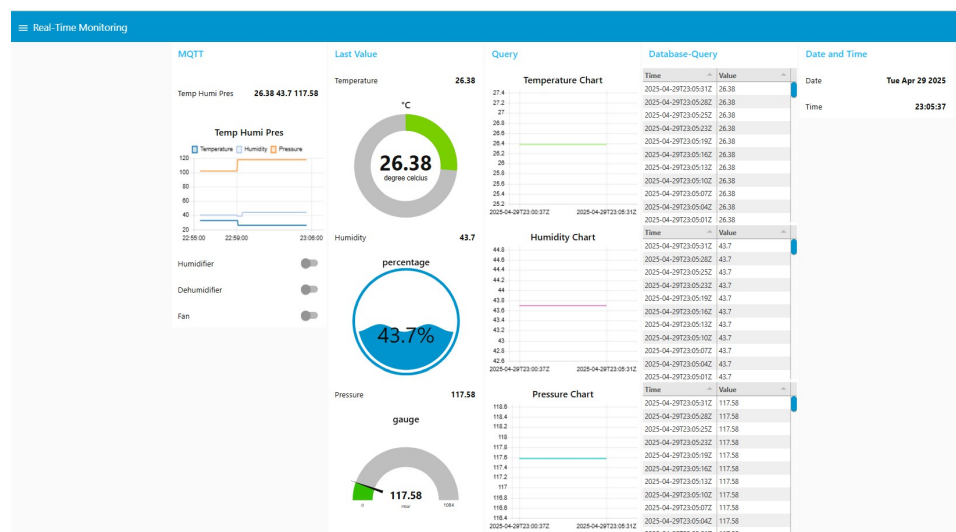


Figure 4.5.4: The Node-RED dashboard of real-time monitoring

The MQTT broker is also running, as it manages the communication between the devices and the dashboard. The system's sensors for temperature, humidity, and pressure should be publishing their data, along with timestamps to the MQTT topics configured in Node-RED. These messages containing readings will be sent to the MQTT Panel. Additionally, users can control the on/off status of the humidifier, dehumidifier and fan directly from the MQTT Panel.

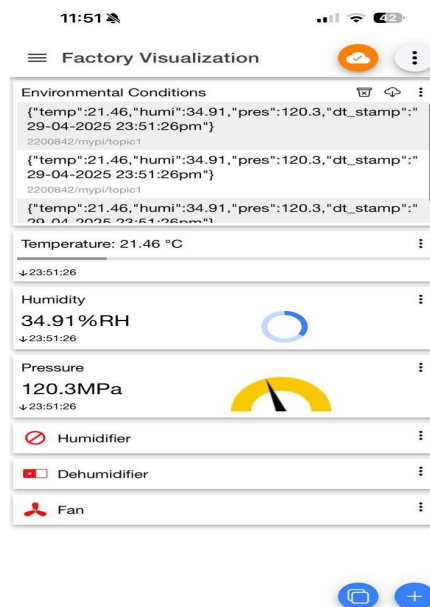


Figure 4.5.5: MQTT Panel

Moreover, the Telegram chatbot is also considered as a part of the user interface. A user-friendly menu is configured to allow users to check inventory status, monitor environmental conditions, control system automation, and receive critical alerts directly through the Telegram app. Users only need to click on the provided commands within the chatbot menu, it ensures easy remote monitoring and control without needing direct access to the dashboard.

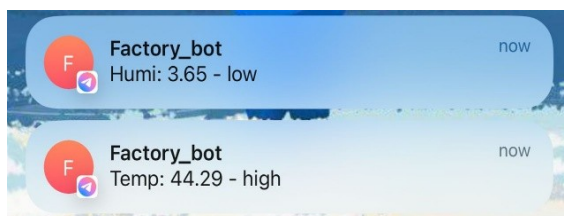


Figure 4.5.6: Telegram notifications Figure 4.5.7: Telegram control menu

4.6 Concluding Remark

This chapter described the technical and architectural design of the IoT-Based Inventory Management System in smart factory, including its system architecture, functional modules, system flow, database design, and graphical user interface layout. Through careful design choices such as using Node-RED for flow control, InfluxDB for time-series data logging, and a user-friendly Node-RED Dashboard for monitoring, the system ensures real-time visibility, efficiency, and scalability.

CHAPTER 5

System Implementation

5.1 Software Setup

The implementation of the IoT-Based Inventory Management System requires the installation and configuration of several software components to create a fully functional smart factory simulation environment.

5.1.1 Inkscape (SVG Editor) Installation

Inkscape is a free and open-source vector graphics editor used to design and edit Scalable Vector Graphics (SVG) files. SVG is an XML-based image format that supports interactivity and animation, making it suitable for creating the digital twin visualization in this project.

- **Download link:** <https://inkscape.org/release/>
- **Usage:** Used to design, edit, and assign IDs to smart factory components. These SVG files were later integrated into Node-RED for real-time animation.

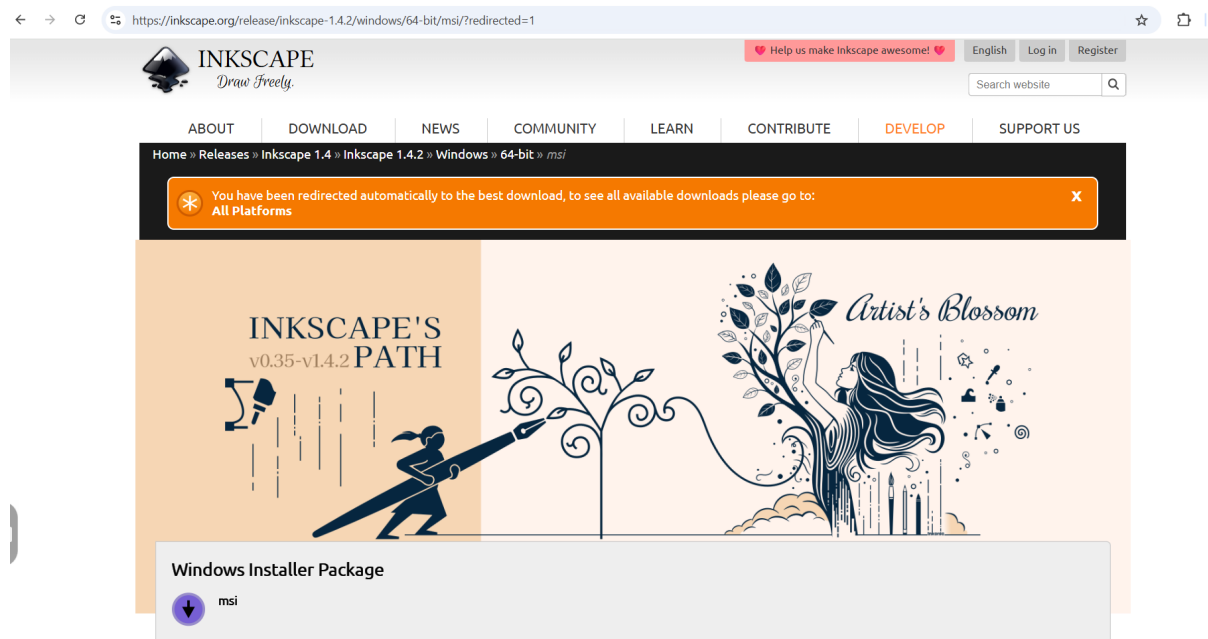


Figure 5.1.1.1: Inkscape installation platform

5.1.2 Node-RED Installation and Setup

Step 1: Install Node.js

Node.js is required as the runtime environment for Node-RED.

1. Download the latest LTS version of Node.js from the official Node.js homepage.
2. Run the downloaded installer file (MSI for Windows). Administrator rights are required.
3. Accept the default installation settings.

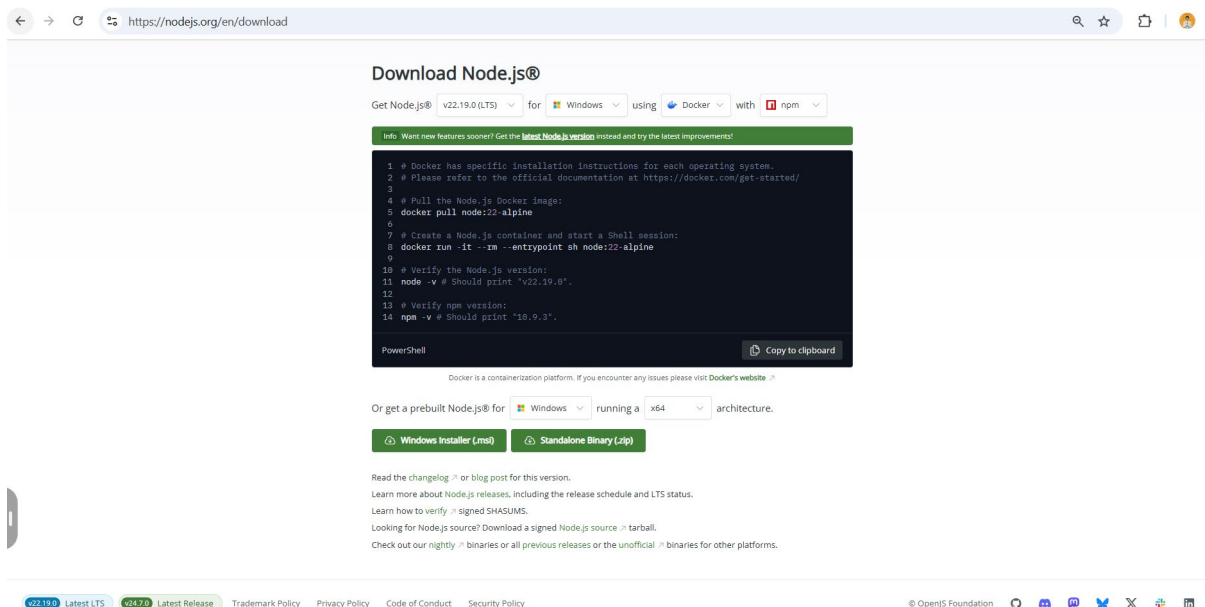


Figure 5.1.2.1: Node.js download platform

4. Once installation is complete, restart any open command prompts to refresh environment variables.
5. Verify installation in command prompt: ***node --version && npm --version***

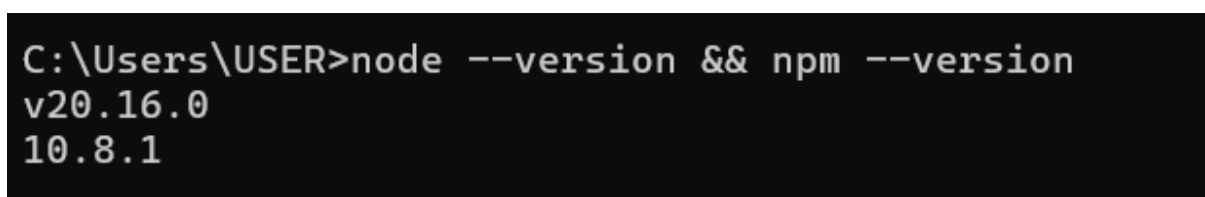


Figure 5.1.2.2: Verification of Node-RED installation's command prompt

Step 2: Install Node-RED

Node-RED is installed globally using npm. Open a command prompt and run: ***npm install -g --unsafe-perm node-red***

```
C:\Users\USER>npm install -g --unsafe-perm node-red
```

Figure 5.1.2.3: Command prompt installation

This will install Node-RED and add the node-red command to the system path.

Step 3: Run Node-RED

Once installation is complete, Node-RED can be launched with: ***node-red***

```
C:\Users\USER>node-red
10 Sep 22:50:24 - [info]

Welcome to Node-RED
=====

10 Sep 22:50:24 - [info] Node-RED version: v4.0.2
10 Sep 22:50:24 - [info] Node.js version: v20.16.0
10 Sep 22:50:24 - [info] Windows_NT 10.0.26100 x64 LE
```

Figure 5.1.2.4: Node-RED launching command prompt

By default, the editor runs at <http://localhost:1880/> and can be accessed through a web browser <http://localhost:1880/ui>

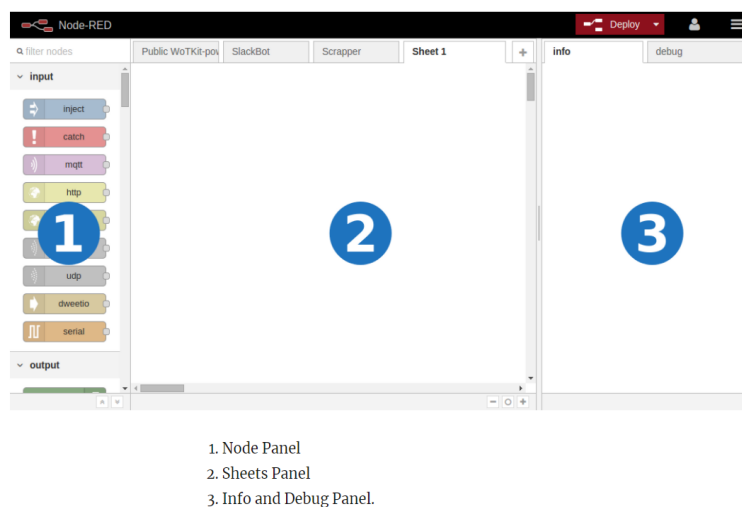


Figure 5.1.2.5: Node-RED panel

Step 4: Install Additional Nodes (Project-Specific)

To support the smart factory simulation, several additional Node-RED nodes were installed:

- node-red-dashboard - for dashboard visualization.
- node-red-contrib-influxdb - to integrate with InfluxDB.
- node-red-contrib-telegrambot - to enable Telegram notifications.
- node-red-node-random - for simulation of random defect/environmental conditions.

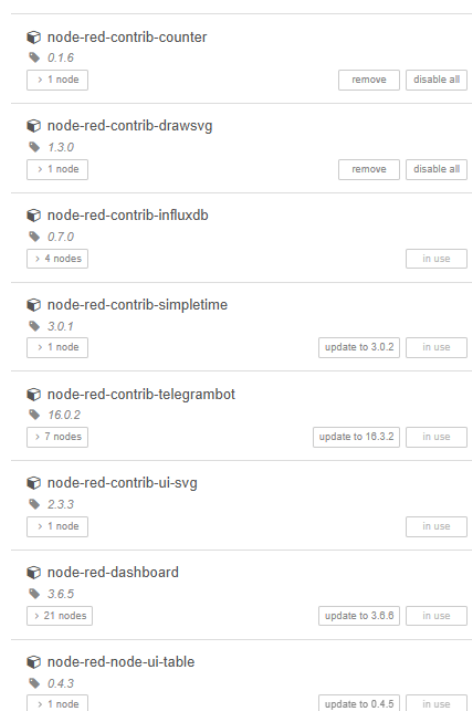


Figure 5.1.2.6: Node-RED palette

These nodes were installed using npm or directly via Node-RED's Manage Palette option.

5.1.3 InfluxDB Database Installation and Setup

The InfluxDB installation process began by navigating to the official InfluxData documentation at <https://docs.influxdata.com/influxdb/v2/install/?t=Windows> and clicking the download link.

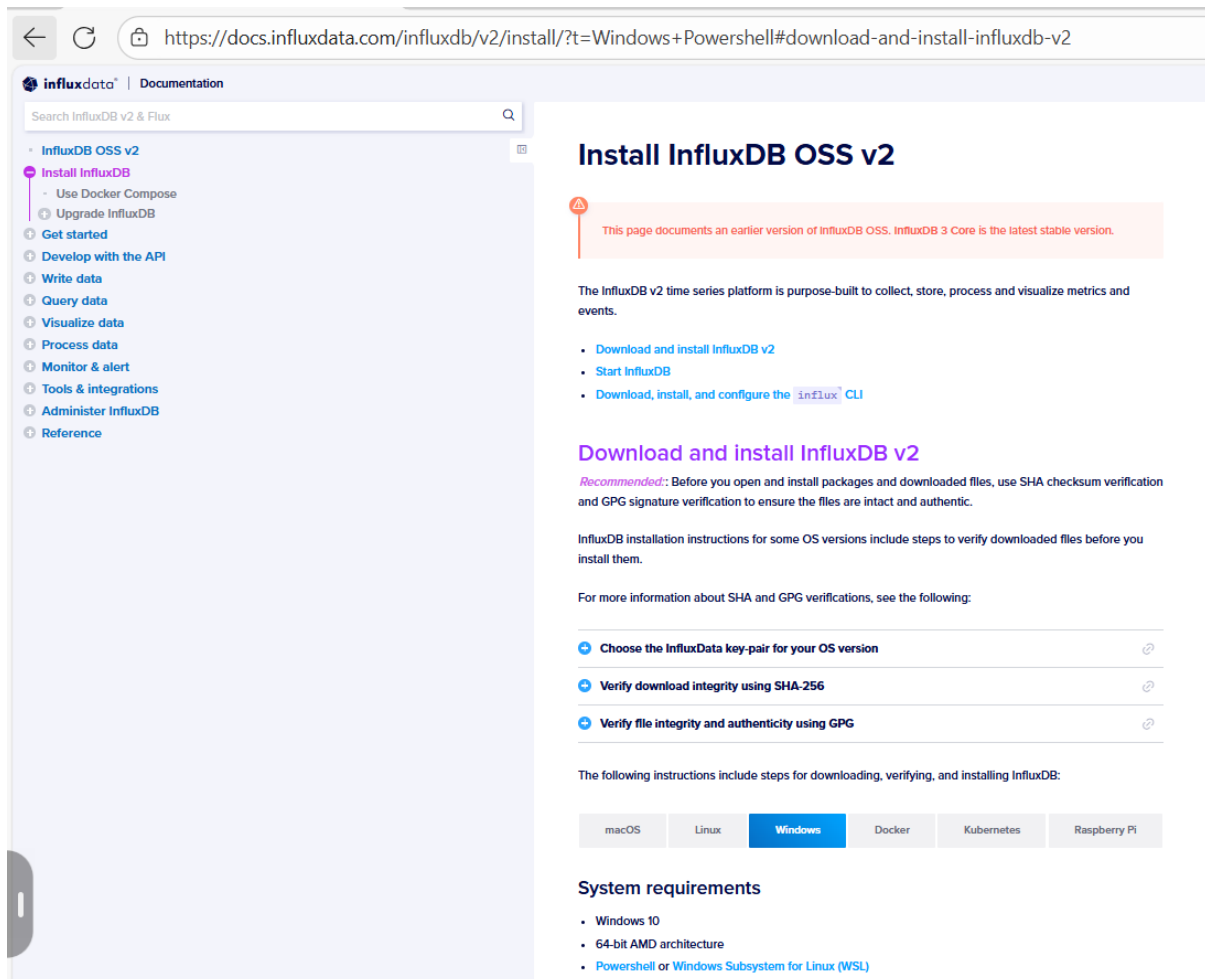


Figure 5.1.3.1: InfluxDB installation platform

The downloaded package was unzipped to the directory: C:\Program Files\InfluxData\

```
C:\>cd Program Files
C:\Program Files>cd InfluxData
C:\Program Files\InfluxData>
```

Figure 5.1.3.2: InfluxDB directory

The database server was initiated by opening a command prompt, navigating to the InfluxData directory, and executing the following command: *influxd*

```
C:\Program Files>cd InfluxData
C:\Program Files\InfluxData>influxd
2025-09-10T23:55:21.367161Z info Welcome to InfluxDB {"log_id": "0yuWjB_l000", "version": "v2.7.8", "commit": "18c989726c", "build_date": "2024-07-25T19:55:40Z", "log_level": "info"}
2025-09-10T23:55:21.388960Z info Resources opened {"log_id": "0yuWjB_l000", "service": "bolt", "path": "C:\\Users\\USER\\.influxdbv2\\influxd.bolt"}
2025-09-10T23:55:21.390796Z info Resources opened {"log_id": "0yuWjB_l000", "service": "sqlite", "path": "C:\\Users\\USER\\.influxdbv2\\influxd.sqlite"}
2025-09-10T23:55:21.423208Z info Checking InfluxDB metadata for prior version. {"log_id": "0yuWjB_l000", "bolt_path": "C:\\Users\\USER\\.influxdbv2\\influxd.bolt"}
2025-09-10T23:55:21.424374Z info Using data dir {"log_id": "0yuWjB_l000", "service": "storage-engine", "service": "store", "path": "C:\\Users\\USER\\.influxdbv2\\engine\\data"}
2025-09-10T23:55:21.425882Z info Compaction settings {"log_id": "0yuWjB_l000", "service": "storage-engine", "service": "store", "max_concurrent_compactions": 4, "throughput_bytes_per_second": 50331648, "throughput_bytes_per_second_burst": 50331648}
2025-09-10T23:55:21.425882Z info Open store (start) {"log_id": "0yuWjB_l000", "service": "storage-engine", "service": "store", "op_name": "tsdb_open", "op_event": "start"}
2025-09-10T23:55:21.592123Z info index opened with 8 partitions {"log_id": "0yuWjB_l000", "service": "storage-engine", "index": "tsi"}
```

Figure 5.1.3.3: InfluxDB launching command prompt

Once the server was running, the InfluxDB web interface was accessed through a browser at: <http://127.0.0.1:8086/>

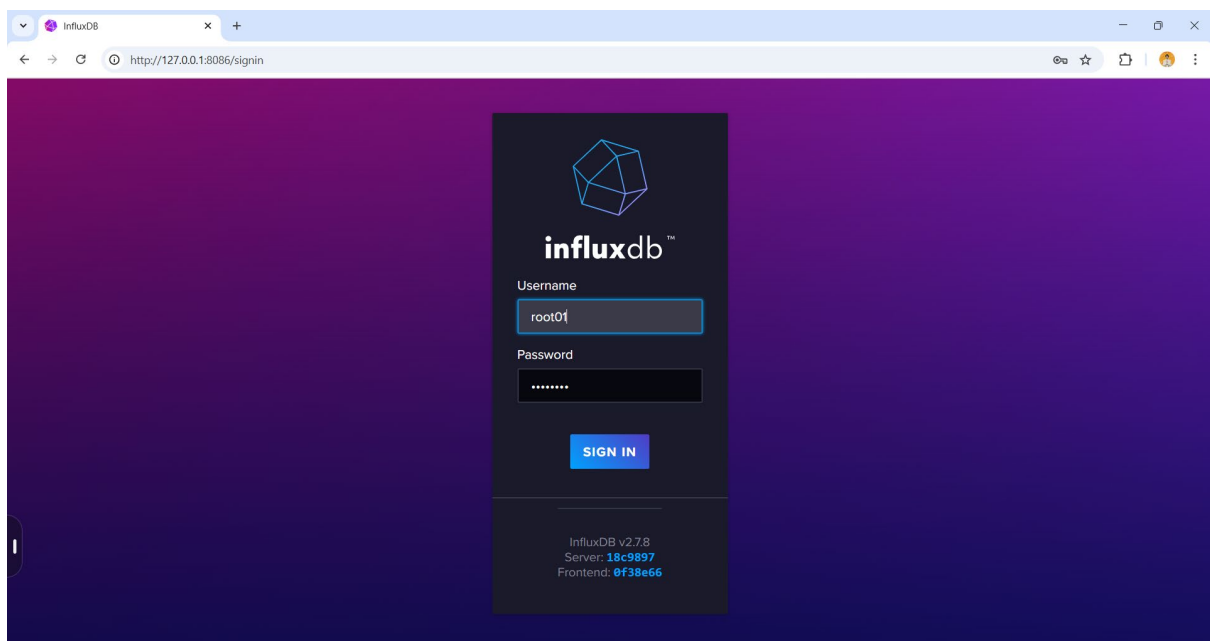


Figure 5.1.3.4: InfluxDB login page

The initial setup was completed by configuring the following parameters:

- **Username:** root01
- **Password:** 12345678
- **Organization:** FICT
- **Bucket:** Smart Factory

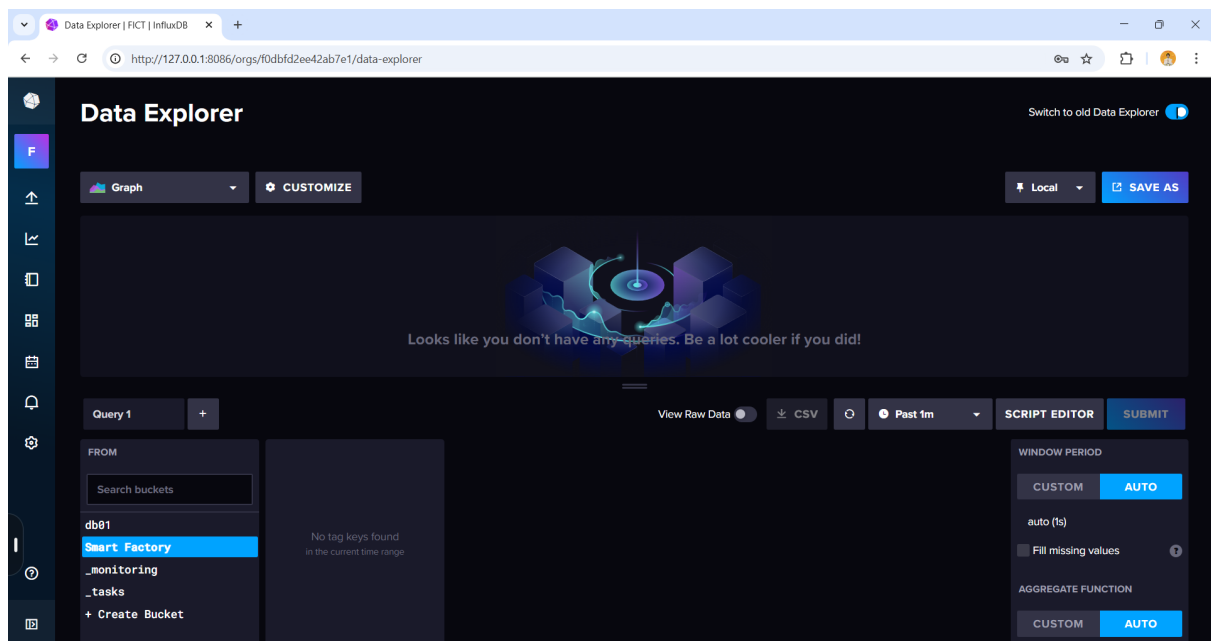


Figure 5.1.3.5: InfluxDB platform

During the setup process, an API token was generated and securely stored for future database connections. This token is critical for programmatic access to the database and cannot be retrieved again once the setup is completed.

5.1.4 MQTT Installation and Setup

For MQTT communication, the HiveMQ public broker was utilized. The IoT MQTT Panel application was installed on a mobile device from the respective app store to facilitate testing and monitoring of MQTT messages.

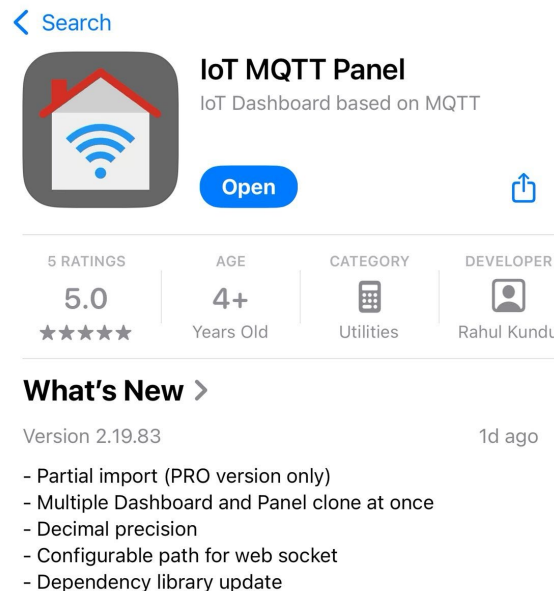


Figure 5.1.4.1: IoT MQTT Panel application installation

After installation, create a new connection.

A new MQTT broker was configured with the following settings:

- **Connection Name:** Factory
- **Server:** broker.hivemq.com
- **Port:** 1883
- **Network Protocol:** TCP

Edit Connection

Connection name *
Factory

Client ID
wteCy0tVMi

Broker address *
broker.hivemq.com

Port *
1883

Network protocol
TCP

Add Dashboard

Factory Visualization

Additional options

CANCEL SAVE

Figure 5.1.4.2: Connection configuration for smart factory

5.1.5 Telegram Bot Setup and Configuration

The Telegram bot integration was implemented to enable remote monitoring and alert notifications. The setup process began by accessing Telegram on a mobile device and searching for @BotFather, which is Telegram's official bot creation service.

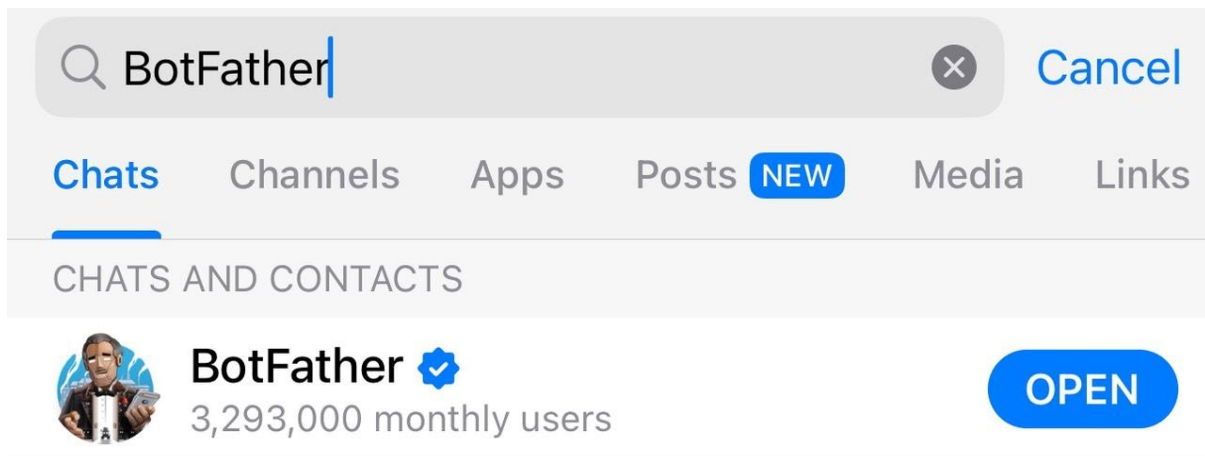


Figure 5.1.5.1: Telegram Bot Father

The bot creation process involved the following steps:

1. Sending /start command to BotFather to access the menu
2. Typing /newbot to initiate new bot creation
3. Providing a bot name ("Factory_bot")

4. Setting a unique username (“factory666_bot”)
5. Receiving the bot token for HTTP API access

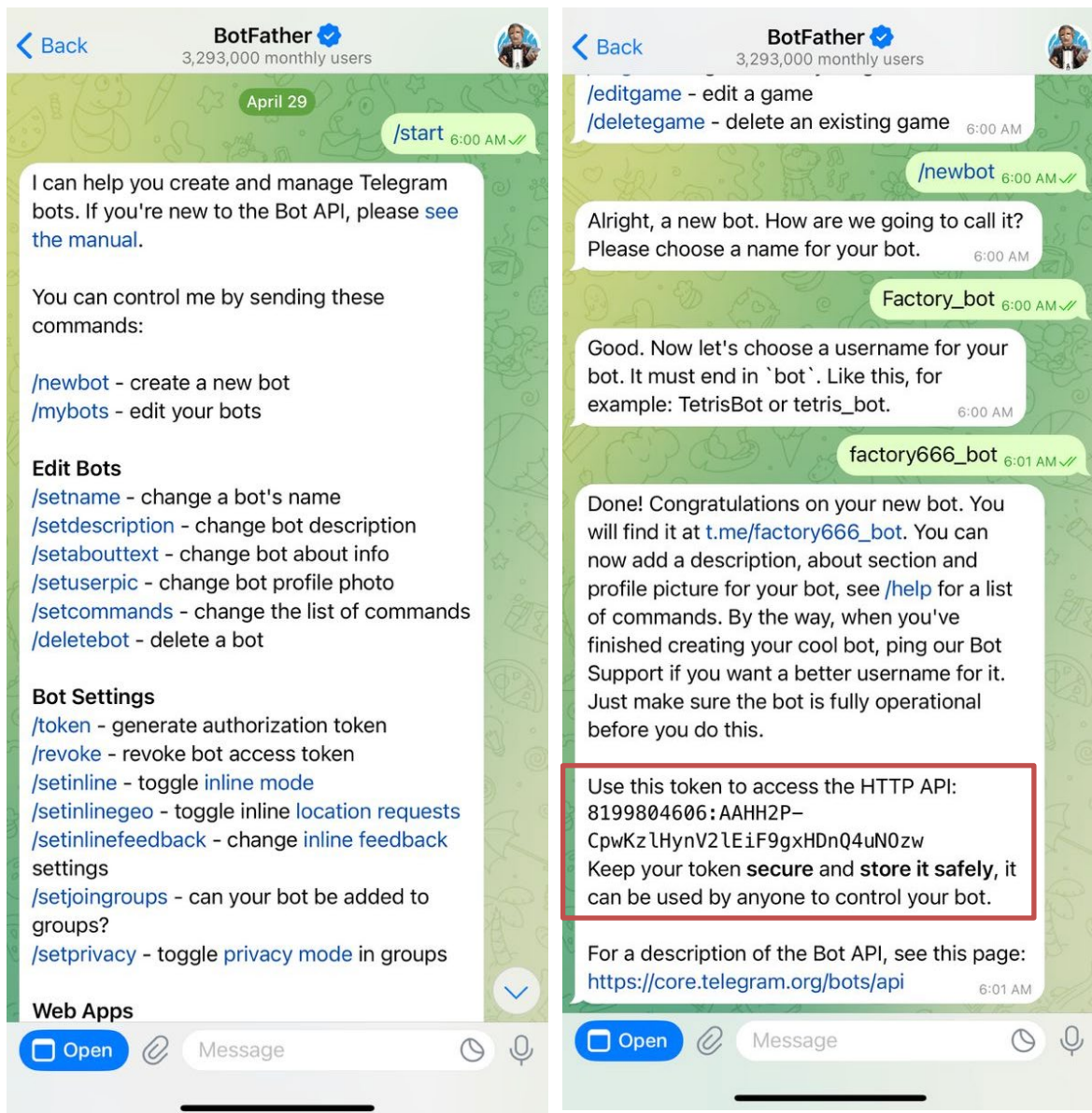


Figure 5.1.5.2: Telegram Bot creation process

The bot token was securely stored as it serves as the authentication key for all bot operations. The bot's chat interface was accessed by clicking the provided t.me link.

5.2 Setting and Configuration

To ensure proper functionality of the IoT-Based Inventory Management System, each component must be integrated correctly into the Node-RED environment. Node-RED serves as the central platform managing logic flows and communication between different services including InfluxDB, MQTT, Telegram Bot, and dashboard components. Specialized nodes require precise configuration of connection parameters, authentication settings, and flow logic to ensure seamless service interaction.

5.2.1 InfluxDB Setting and Configuration

InfluxDB was used to store, manage, and query environmental sensor data (temperature, humidity, and pressure) in the Smart Factory system. The InfluxDB setup in Node-RED enabled real-time monitoring of environmental data. With Flux queries, the system can extract latest values, analyze short-term trends, and display results in multiple dashboard widgets (text, gauge, chart, table). The integration also ensures automated alerting when thresholds are crossed.

Database Connection Setup

To enable InfluxDB to receive environmental data from the system, the InfluxDB node must be added and connected to the environmental data processing function nodes within Node-RED. Drag the influxdb out node from the node palette and position it after the environmental data formatting function. Connect the data formatting function output to the influxdb out node input. Configure the database connection parameters:

Server Configuration:

- Name: Smart Factory Database Connection
- Version: Select 2.0 (InfluxDB version 2.x)
- URL: `http://localhost:8086` (default InfluxDB server address)
- Token: Enter the API token generated during InfluxDB initial setup
- Enable “Verify server certificate” for secure connections

Edit influxdb in node > Edit influxdb node

Delete Cancel Update

Properties

Name Smart Factory Database Connection

Version 2.0

URL http://localhost:8086

Token

Connection timeout (seconds) 10

☒ Verify server certificate

Figure 5.2.1.1: Configuration setting for influxdb server

Measurement Configuration: After establishing the server connection, configure the data storage parameters:

- Organization: FICT (as configured in InfluxDB setup)
- Bucket: Smart Factory (destination bucket for time-series data)
- Measurement: location=factory (measurement identifier for environmental data)
- Time Precision: Seconds (s) (timestamp precision level)

Edit influxdb out node

Delete Cancel Done

Properties

Name Name

Server [v2.0] Smart Factory Database Connection

Organization FICT

Bucket Smart Factory

Measurement location=factory

Time Precision Seconds (s)

Figure 5.2.1.2: Configuration setting for data storage parameters

Verification of InfluxDB Integration:

After deployment, verify successful data storage by accessing the InfluxDB web interface at <http://localhost:8086>. Navigate to the Data Explorer, select the “Smart Factory” bucket, and query the “location=factory” measurement to confirm environmental data is being stored with proper timestamps and field values.

The data should appear with three fields (temperature, humidity, pressure) and associated metadata tags, updating in real-time as the Node-RED flow processes environmental sensor data.

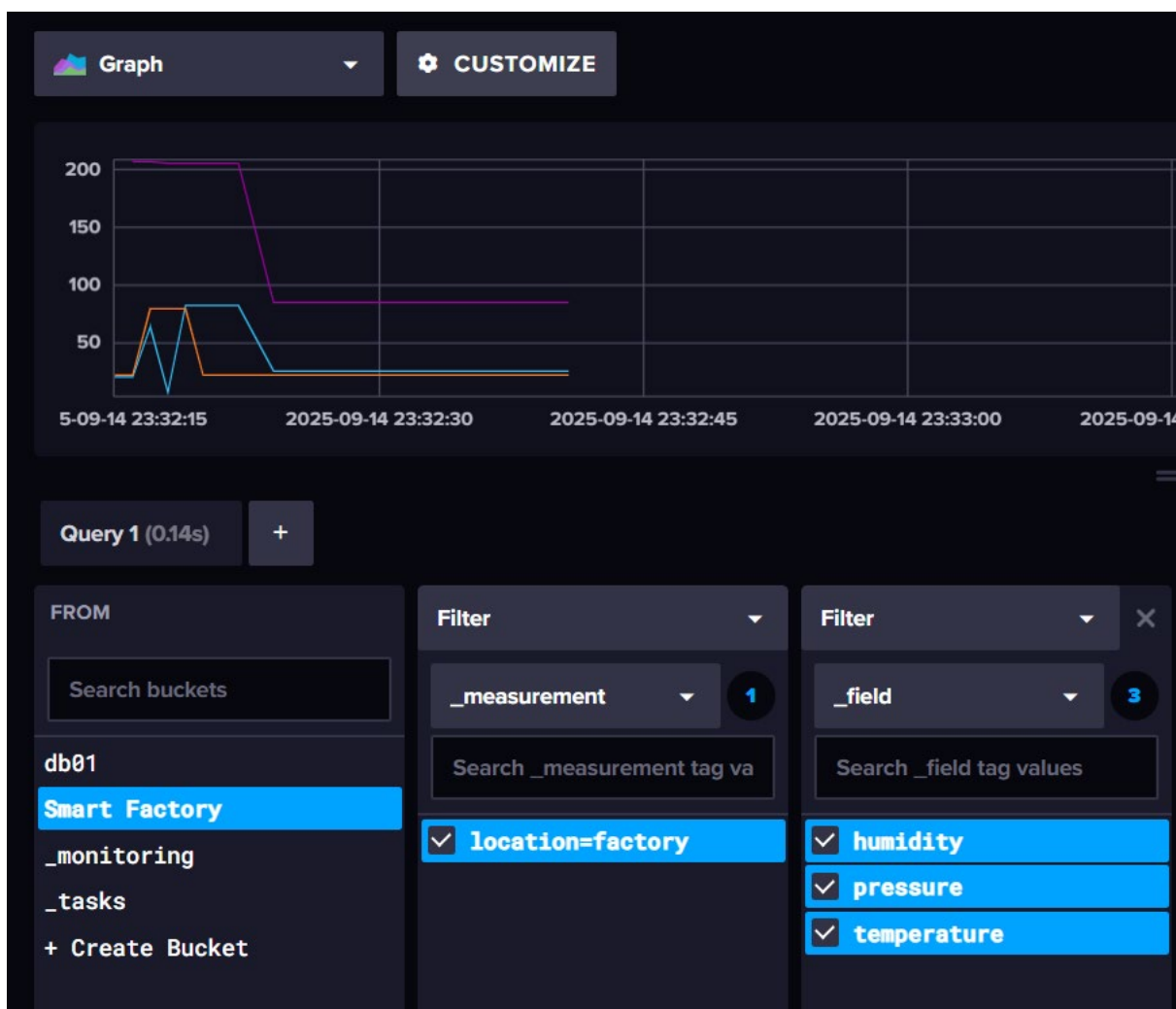


Figure 5.2.1.3: InfluxDB interface

5.2.2 MQTT Setting and Configuration

MQTT Broker Configuration

MQTT was integrated into the Smart Factory system to enable lightweight, real-time communication between sensors, actuators, and the Node-RED dashboard. By using a public broker (HiveMQ), the system achieved seamless data exchange for environmental monitoring (temperature, humidity, pressure) and remote actuator control (fan, humidifier, dehumidifier). The MQTT configuration provided bi-directional communication between sensors, actuators, and the Smart Factory dashboard. Sensor readings were published to topic1, while actuator commands were sent to topic2. Subscribed data was processed in real-time, stored in InfluxDB, and visualized on the dashboard.

Step 1: MQTT Broker Configuration

- A public MQTT broker was used for testing. The broker ensures all messages published by devices (sensors/actuators) are available to subscribers in real time.

The screenshot shows the Node-RED MQTT Broker configuration interface. The window title is "Edit mqtt out node > Edit mqtt-broker node". At the top, there are three buttons: "Delete", "Cancel", and "Update". Below these is a "Properties" tab. The "Name" field is set to "Hivemq Factory". The "Connection" tab is selected, showing the "Server" field set to "broker.hivemq.com" and the "Port" field set to "1883". There are checkboxes for "Connect automatically" (checked) and "Use TLS" (unchecked). The "Protocol" dropdown is set to "MQTT V3.1.1". The "Client ID" field is set to "Leave blank for auto generated". The "Keep Alive" field is set to "60". The "Session" checkbox "Use clean session" is checked.

Figure 5.2.2.1: MQTT Broker configuration

Step 2: Publishing Data to MQTT

Sensor Data Transmission

Multiple environmental parameters (temperature, humidity, pressure, and timestamp) were combined into a JSON object and published to the broker on topic: 2200842/mypi/topic1

```
9/15/2025, 8:12:15 PM node: debug 6
2200842/mypi/topic1 : msg.payload : Object
▶ { temp: 56.51, humi: 61.84, pres:
165.73, dt_stamp: "15-09-2025
20:12:15pm" }
```

Figure 5.2.2.2: Example payload

- Node-RED Join node aggregates multiple sensor values into one message.
- JSON Node converts the payload into a valid string format before publishing.
- The MQTT Out node publishes the data.

This ensures consistent and structured data transmission from sensors to the broker.

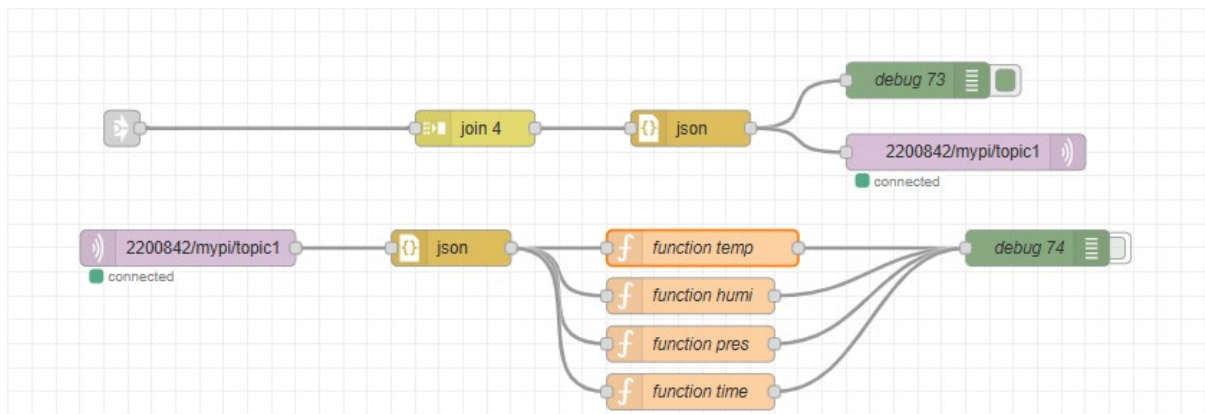


Figure 5.2.2.3: MQTT Node-RED flow

Step 3: Subscribing to Sensor Data

- MQTT In Node subscribes to topic: 2200842/mypi/topic1
- Incoming JSON payloads are decoded and processed by Function Nodes.

Example Functions:

- Temperature Node:

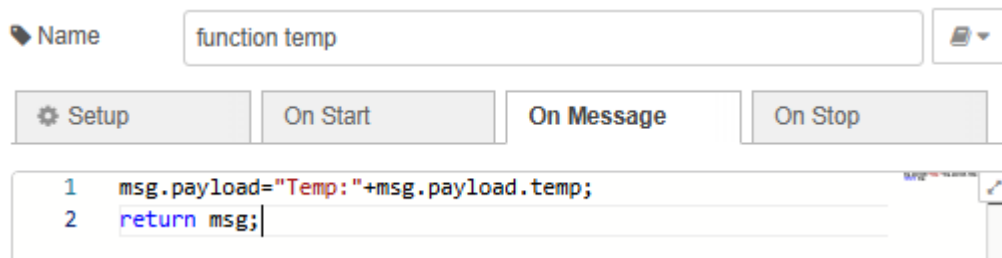


Figure 5.2.2.4: Temperature node function

- Humidity Node:

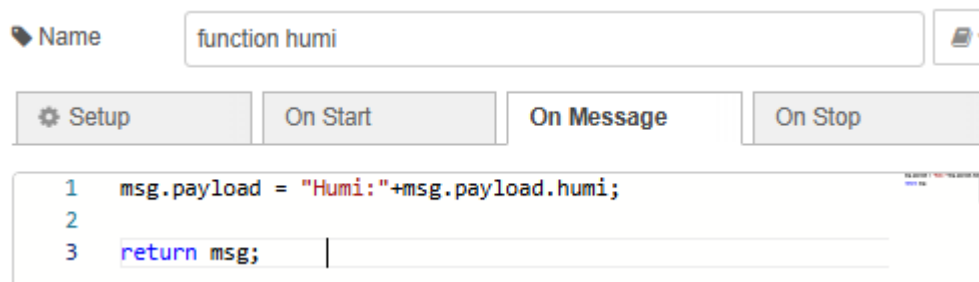


Figure 5.2.2.5: Humidity node function

- Pressure Node:

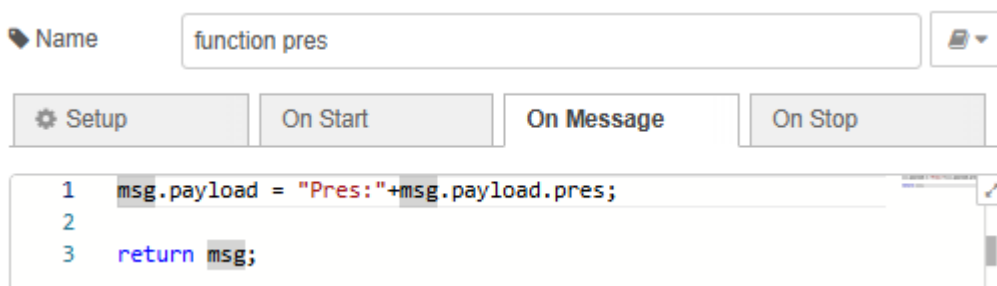


Figure 5.2.2.6: Pressure node function

- Timestamp Node:

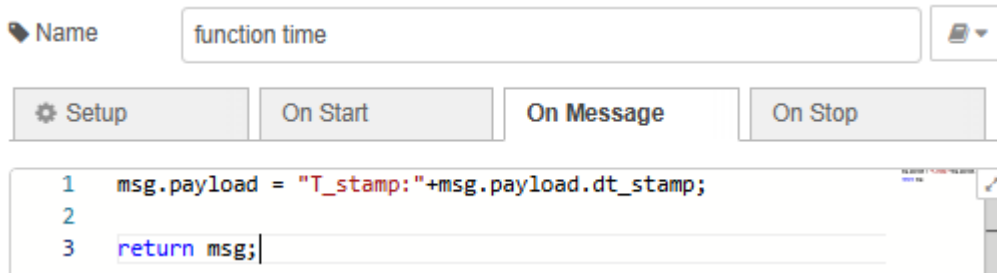


Figure 5.2.2.7: Timestamp node function

The data is then displayed in:

- UI Text Widget → Real-time values
- UI Chart Widget → Graphical line chart of Temperature/Humidity/Pressure trends
- InfluxDB → Long-term data storage

Step 4: Actuator Control via MQTT

A second topic was dedicated to actuator control: 2200842/mypi/topic2

The Node-RED dashboard provided UI Switches for Fan, Humidifier, and Dehumidifier.

Example Workflow for Humidifier Control

1. User toggles Humidifier Switch on the dashboard.

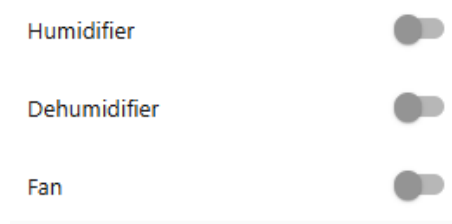


Figure 5.2.2.8: Actuators toggle switches

2. The switch outputs payload → “humidifier_on” or “humidifier_off”.
3. Function Node converts it to JSON.

4. Message is published to 2200842/mypi/topic2.
5. Subscriber (another Node-RED flow or IoT device) interprets the command.
6. SVG updates:
 - ON → humidifier turns green
 - OFF → humidifier turns black

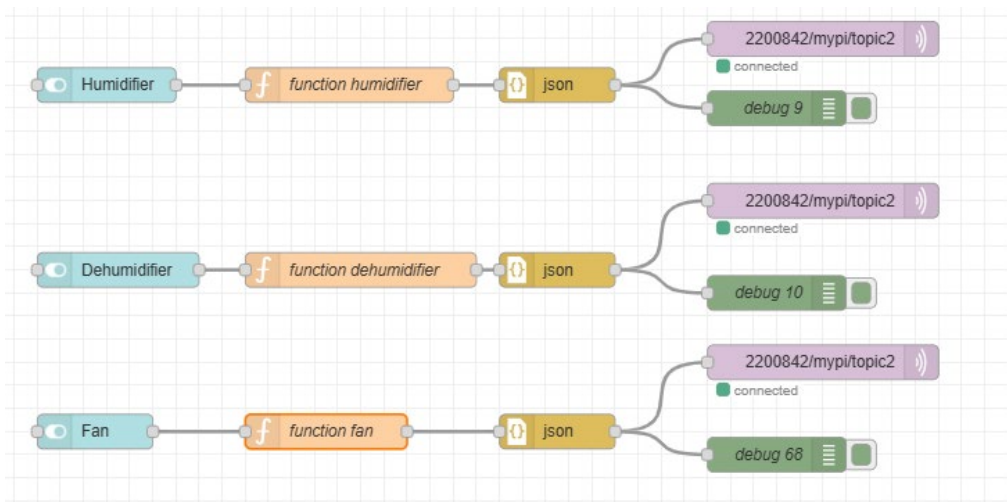


Figure 5.2.2.9: Node-RED flow of actuators toggle switches

Step 5: Data Logging with InfluxDB

- Subscribed MQTT sensor data was also written to InfluxDB for persistent storage.
- Data schema:
 - Measurement: location==factory
 - Fields: temperature, humidity, pressure
 - Timestamp: auto-generated

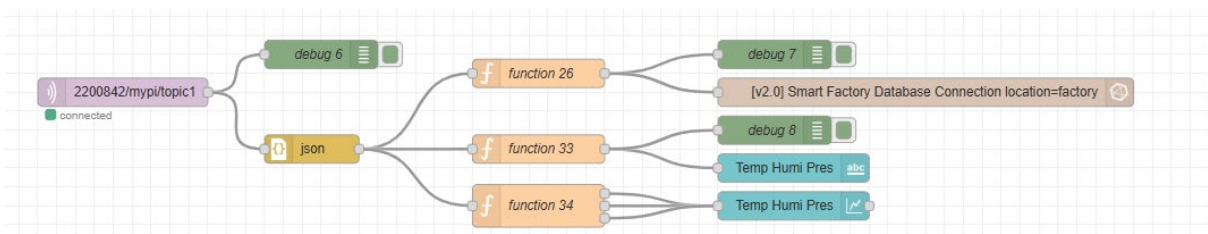


Figure 5.2.2.10: Node-RED flow of subscribed MQTT sensor data store in database

Step 6: Real-Time Dashboard Integration

- Node-RED Dashboard included:
 - Text Widgets → Latest Temp/Humi/Pres values
 - Line Chart → Trends over time
 - Switch Widgets → Remote control of actuators
- All dashboard updates were linked to MQTT to ensure synchronization between broker, IoT devices, and real-time visualization.

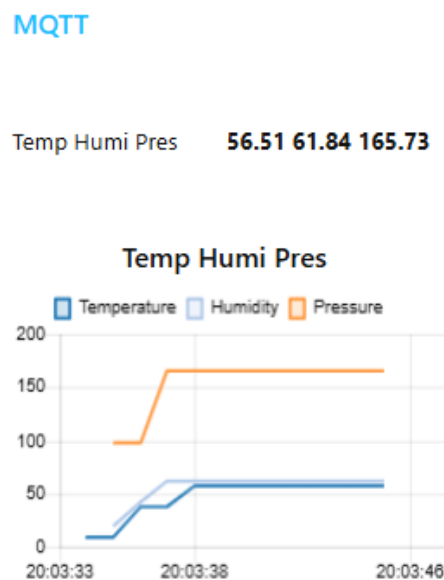


Figure 5.2.2.11: Real-time environmental monitoring dashboard

MQTT Configuration with IoT MQTT Panel

In addition to configuring MQTT nodes in Node-RED, the system was also integrated with the IoT MQTT Panel mobile application. This allows real-time monitoring and control of devices directly from a smartphone, complementing the Node-RED dashboard interface.

First, a new connection was created in the IoT MQTT Panel application. The same broker settings used in Node-RED were applied:

- **Broker:** broker.hivemq.com
- **Port:** 1883 (default for MQTT without TLS)
- **Client ID:** Automatically generated or user-defined
- **Clean Session:** Enabled

Once saved, the application established a direct connection with the public HiveMQ broker which allows it to subscribe to and publish messages on the same topics as Node-RED.

The screenshot shows the 'Edit Connection' interface of the IoT MQTT Panel application. It features a list of configuration fields, each with a help icon (question mark in a circle) to its right:

- Connection name ***: Factory
- Client ID**: wteCy0tVMi
- Broker address ***: broker.hivemq.com
- Port ***: 1883
- Network protocol**: TCP (with a dropdown arrow)

Below these fields are two interactive elements:

- Add Dashboard**: A button with a plus sign icon.
- Factory Visualization**: A button with a house and grid icon.

At the bottom, there is a section for **Additional options** with a downward arrow, and two large buttons: **CANCEL** and **SAVE**.

Figure 5.2.2.12: IoT MQTT Panel connection setup

After the broker connection was configured, panels were added for specific MQTT topics. This provided a user-friendly graphical interface for monitoring sensor data and controlling actuators.

- To create a panel, the user selects the connection and taps the “+” icon.
- A panel type is chosen based on functionality (e.g., Switch Panel for actuators, Text Panel for monitoring sensor values).
- Each panel is assigned a name and linked to an MQTT topic that matches the topic defined in the Node-RED mqtt in or mqtt out node.

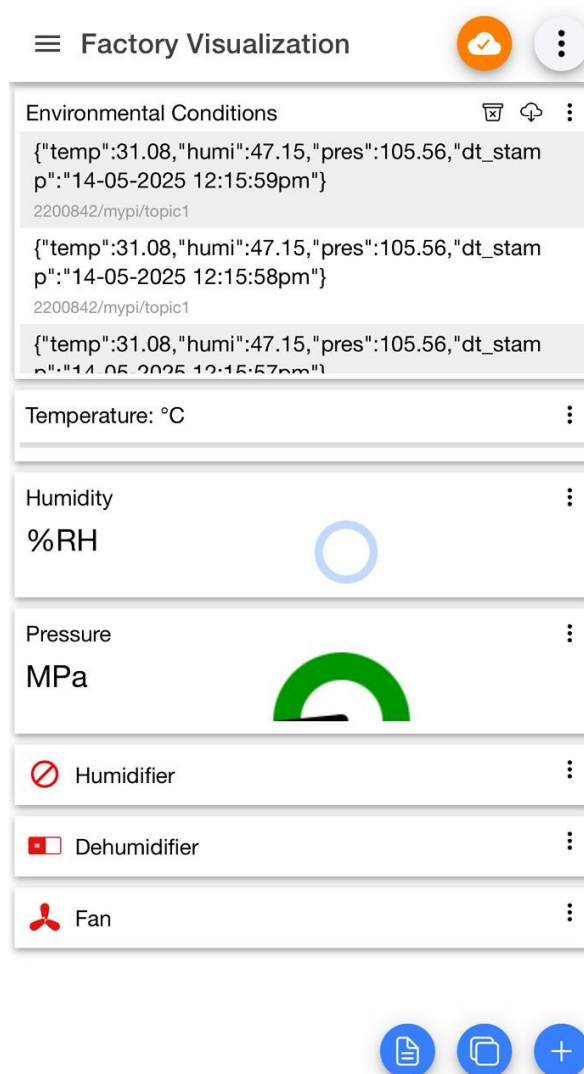


Figure 5.2.2.13: Panel creation in IoT MQTT

For environmental device control (humidifier, fan, dehumidifier), Switch Panels were configured in the IoT MQTT Panel. Each switch was linked to the topic 2200842/myipi/topic2, which is used in Node-RED to receive actuator control commands. These payloads directly match the format expected by the switch nodes in Node-RED. As a result, when the user toggles a switch in the mobile app, the Node-RED flow immediately updates the SVG interface and controls the corresponding device.

The payloads were defined as follows:

← Edit Panel

Panel name *
Humidifier

☐ Disable dashboard prefix topic

Topic *
2200842/myipi/topic2

Subscribe Topic

Payload on *
{\"act\":\"humidifier_on\"}

Payload off *
{\"act\":\"humidifier_off\"}

☒ Use icon switch

On icon ● Icon color #47e026

Off icon ● Icon color #ff0000

Icon size Small ▾

☐ Payload is JSON Data

☐ Show received timestamp

☐ Show sent timestamp

☐ Confirm before publish

Figure 5.2.2.14: Panel Configuration

For real-time monitoring of sensor data (temperature, humidity, pressure), Text Panels and Graph Panels were created. These were linked to the MQTT topic 2200842/myipi/topic1, which publishes JSON-encoded sensor values.

The IoT MQTT Panel automatically parses the incoming data and displays value dynamically. Graph panels allow users to visualize environmental parameter changes over time, complementing the Node-RED dashboard charts.

← Edit Panel

Panel name *

Environmental Conditions

☐ Disable dashboard prefix topic

?

Topic *

2200842/mypi/topic1

☐ Show last message only

Max visible lines

4

Max persistence

100

☐ Hide topic

☐ Payload is JSON Data

☐ Show received timestamp

QoS

0

▼

CANCEL

SAVE

Figure 5.2.2.15: Panel Configuration

← Edit Panel

Panel name *

Temperature

☐ Disable dashboard prefix topic

?

Topic *

2200842/mypi/topic1

Payload min *

0

Payload max *

100

Factor

1

Decimal precision

Unit

°C

Progress type

Horizontal

▼

Color

▼

☐ Dynamic color

☒ Payload is JSON Data

JsonPath for subscribe *

\$.temp

?

☒ Show received timestamp

QoS

0

▼

CANCEL

SAVE

Figure 5.2.2.16: Panel Configuration

5.2.3 Telegram Bot Setting and Configuration

Telegram was integrated into the Smart Factory system to provide real-time alerts and remote actuator control. The configuration involved creating a bot, setting up receiver and sender nodes in Node-RED, processing user commands, and giving interactive responses. This allowed operators to control devices such as the fan, humidifier, and dehumidifier directly from Telegram.

Step 1: Creating a Telegram Bot

1. Search for @BotFather in Telegram
2. Send /newbot command
3. Provide bot name: Factory Bot
4. Provide username: factory666_bot
5. Save the API token provided by BotFather
6. Send a test message to the bot to obtain Chat ID

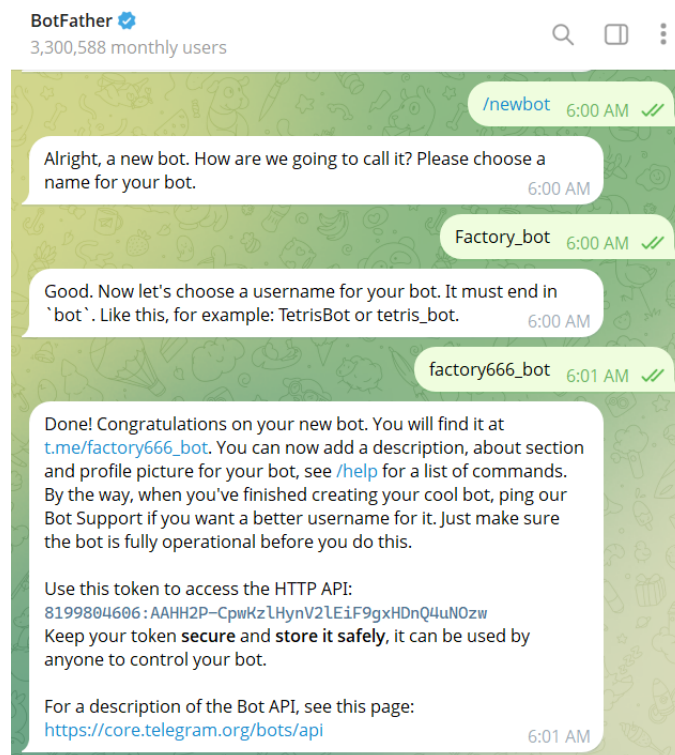


Figure 5.2.3.1: Telegram Bot creation

Step 2: Configuring the Telegram Bot Node

Drag telegram sender node from the node palette. Configure the following:

Bot Configuration:

- Bot Name: Factory_bot
- Token: Enter the token from BotFather

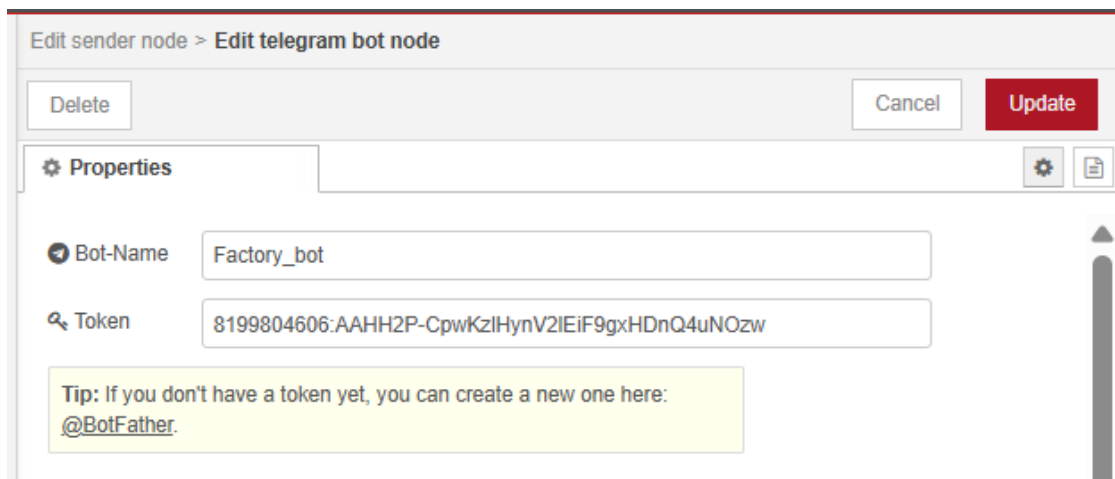


Figure 5.2.3.2: Bot configuration

Connect alert generation functions to this sender node for automated notifications.

Step 3: Receiving Messages from Telegram

- Telegram Receiver Node: Captures all incoming messages from the bot.
- Switch Nodes: Used to filter specific commands such as:
 - /fan_on, /fan_off
 - /humidifier_on, /humidifier_off
 - /dehumidifier_on, /dehumidifier_off

For example, when a user sends /fan_on, the switch node routes the message to the Fan on Function Node.

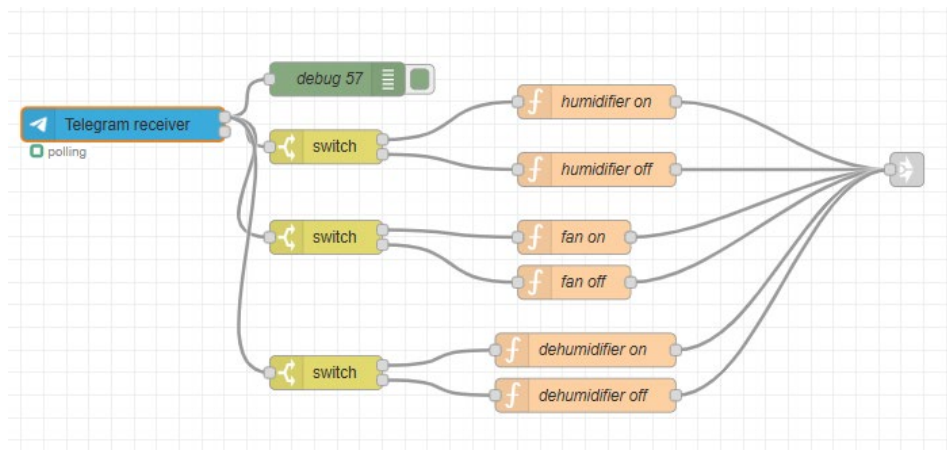


Figure 5.2.3.3: Node-RED Flow of telegram receiver

Step 4: Device Control via Function Nodes

Each device has two Function Nodes (ON and OFF) that send SVG update commands to the dashboard.

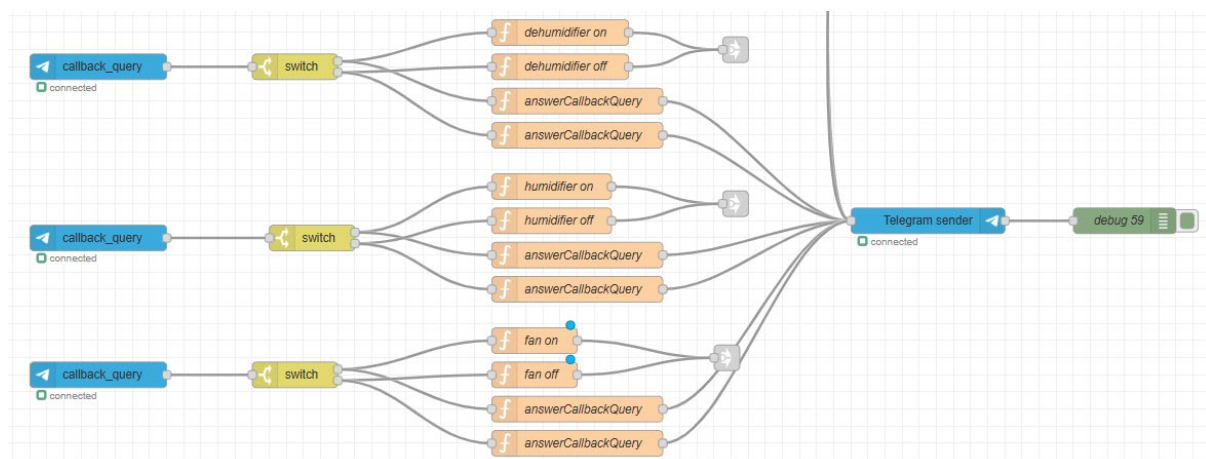


Figure 5.2.3.4: Node-RED flow of telegram callback query

Step 5: Sending Feedback to Users

After executing commands, the bot sends a confirmation message back to the user using an Answer Callback Query Function Node.

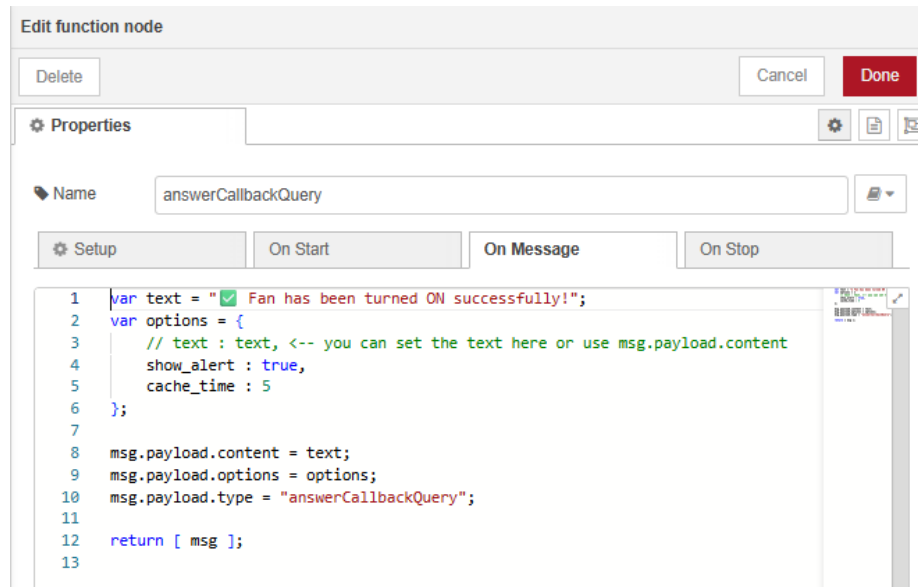


Figure 5.2.3.5: Answer callback query function node.

Step 6: Sending Alerts from Node-RED to Telegram

The Telegram Sender Node is used to deliver alerts and updates. Environmental monitoring logic triggers Telegram notifications when thresholds are exceeded.

5.3 System Operation (with screenshot)

5.3.1 Main Dashboard Interface

The system uses the Node-RED dashboard integrated with SVG graphics to simulate the real-time operations of the smart factory. Main Dashboard Tab contains the overall production flow visualization, real-time counters, and control buttons.

Components are visually represented as SVG objects moving across different stages: loading, mounting, soldering, inspection, and final assembly.

- Green-colored indicators represent components that successfully pass each stage.
- Red-colored indicators highlight defective components automatically routed to the rejection area.

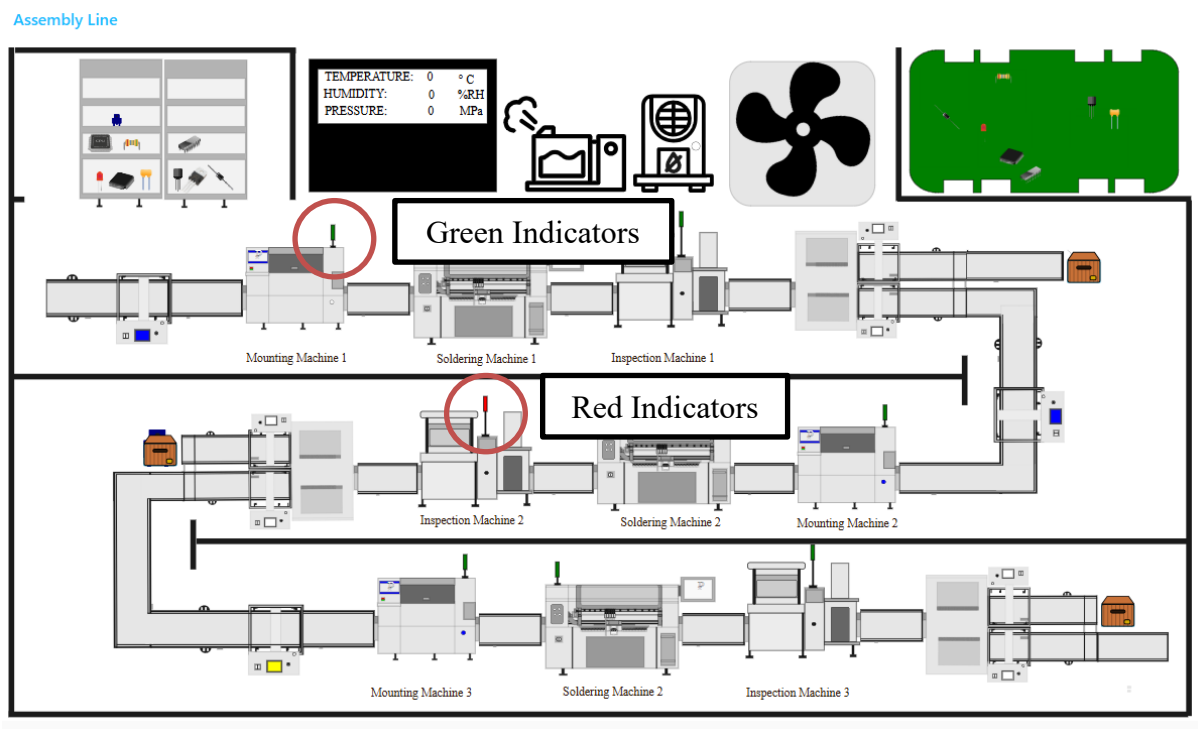


Figure 5.3.1.1: Main dashboard interface

The control panel also displays live machine success rates (Mounting Machine 1,2,3; Soldering Machine 1,2,3; Inspection Machine 1,2,3) using gauges. These gauges update in real time, reflecting the number of processed, passed, and failed components.

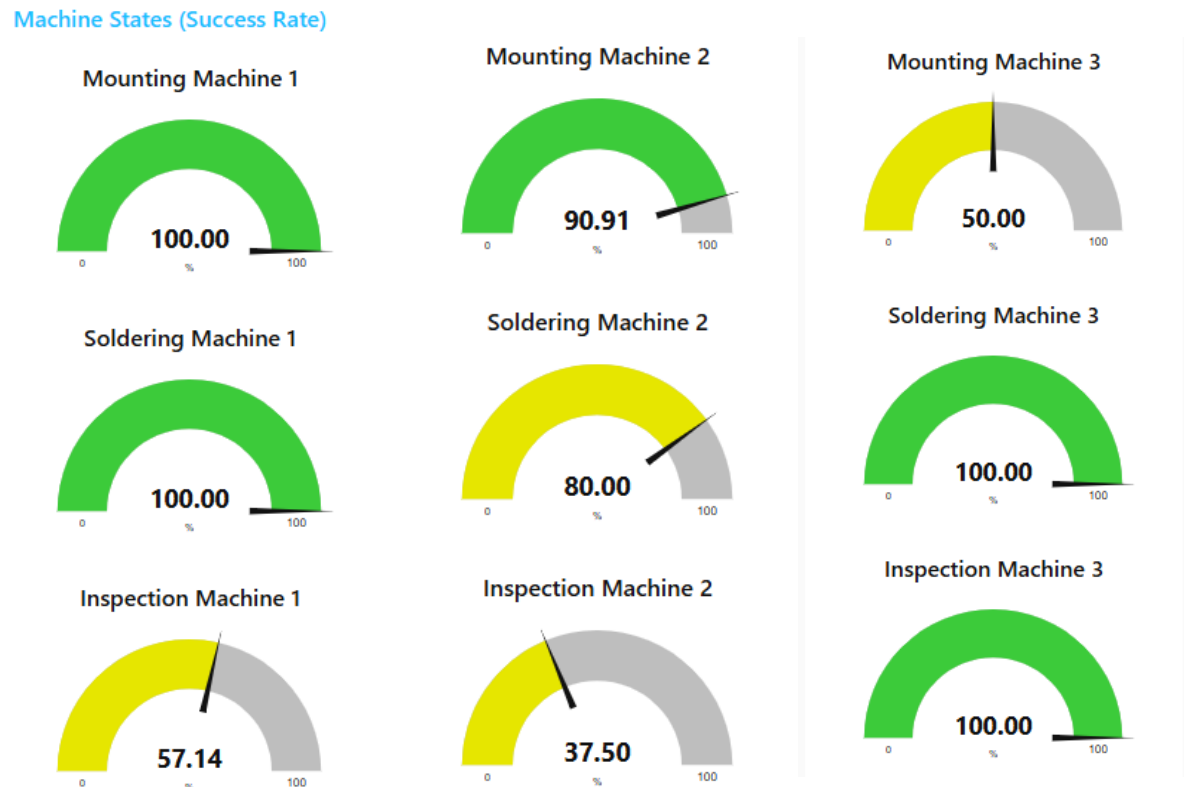


Figure 5.3.1.2: Live machine success rates gauges

The control panel also includes manual controls for environmental devices such as fans, humidifiers, and dehumidifiers.

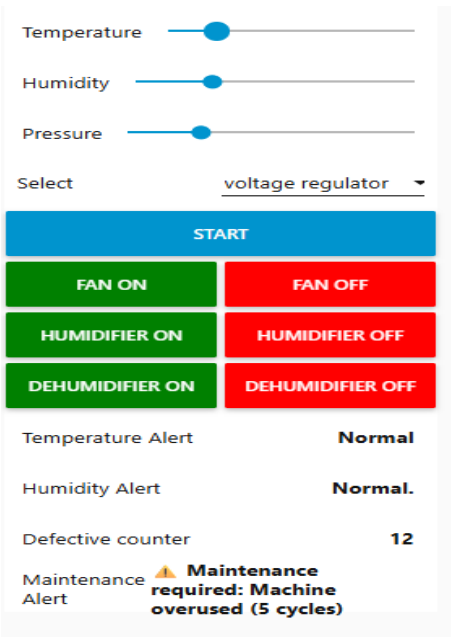
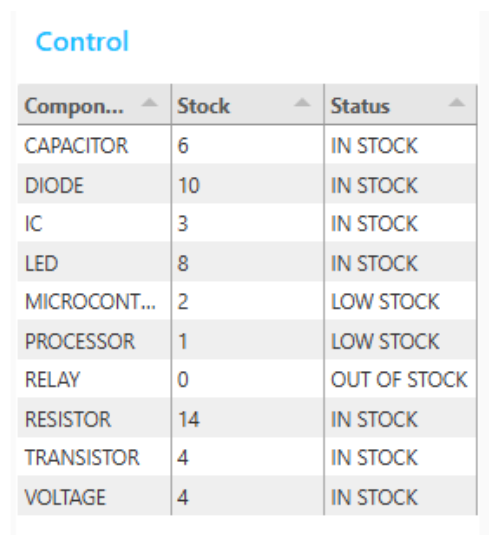


Figure 5.3.1.3: Control panel in main dashboard

The Main Dashboard also includes a real-time inventory table displaying the stock status for all component types. The table automatically updates whenever a component is used in the production line:

- If the stock value is greater than 3, the status shows “IN STOCK”.
- If the stock value drops to 3 or below, the status changes to “LOW STOCK” and is highlighted accordingly.
- If the stock reaches 0, the status changes to “OUT OF STOCK”, and further production of that component is halted until restocked.

This mechanism ensures that operators always have an accurate and up-to-date view of resource availability while running simulations.



The screenshot shows a web interface titled "Control" in blue. Below the title is a table with three columns: "Compon...", "Stock", and "Status". Each column has a small upward-pointing triangle icon. The table lists various electronic components and their current stock levels and statuses.

Compon...	Stock	Status
CAPACITOR	6	IN STOCK
DIODE	10	IN STOCK
IC	3	IN STOCK
LED	8	IN STOCK
MICROCONT...	2	LOW STOCK
PROCESSOR	1	LOW STOCK
RELAY	0	OUT OF STOCK
RESISTOR	14	IN STOCK
TRANSISTOR	4	IN STOCK
VOLTAGE	4	IN STOCK

Figure 5.3.1.4: Real-time inventory table

5.3.2 System Operation of Production Flow

The production flow visualization provides a detailed SVG-based animation of how components are processed throughout the smart factory pipeline.

- Raw material loading – Components are classified into three categories: cheap, medium, and expensive. Each category follows its own dedicated conveyor line. To enhance operator visibility, the loader screen displays different color indicators when a component is detected:
 - Cheap components → Loader screen turns **Green**
 - Medium components → Loader screen turns **Blue**
 - Expensive components → Loader screen turns **Yellow**

This immediate color feedback allows operators to quickly identify the type of component entering the production line.

- Mounting process - Each component moves to the mounting station, where the system simulates placement onto the PCB. Defect detection logic is applied, and any defective components are flagged for removal.
- Soldering process - Successfully mounted components proceed to the soldering station, where solder joints are simulated. Components with defective soldering are automatically marked as failed and redirected away from the main production path.
- Inspection process - Components undergo automated quality inspection. Those that pass continue to functional testing, while failed components are automatically rerouted to the rejection path.
- Final assembly and distribution - Only components that successfully pass all previous stages reach the final assembly stage, after which they are distributed to the final product board for packaging and visualization.

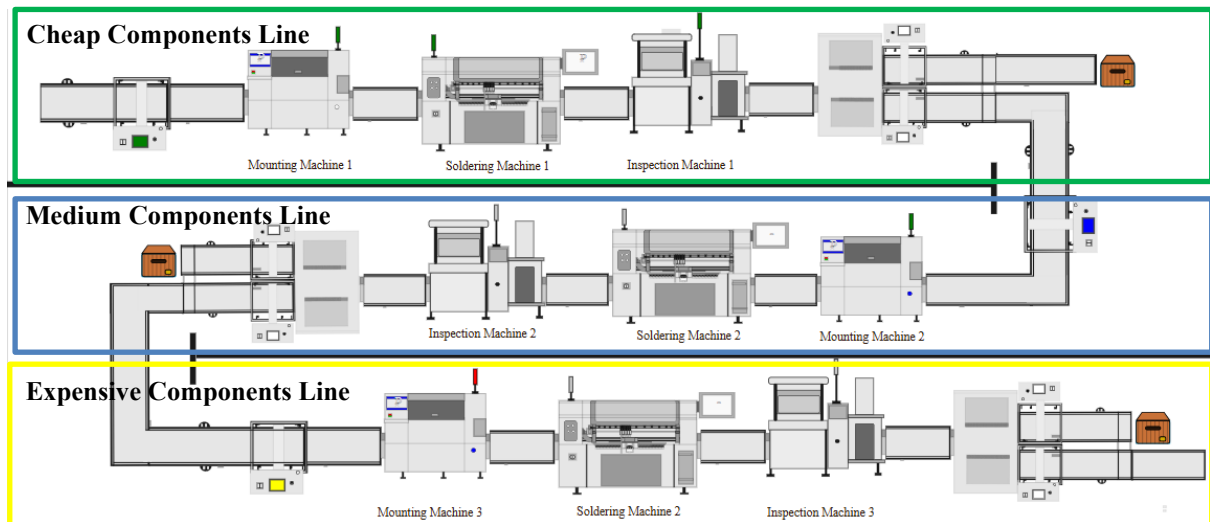


Figure 5.3.2.1: Production flow of assembly line

Logistics

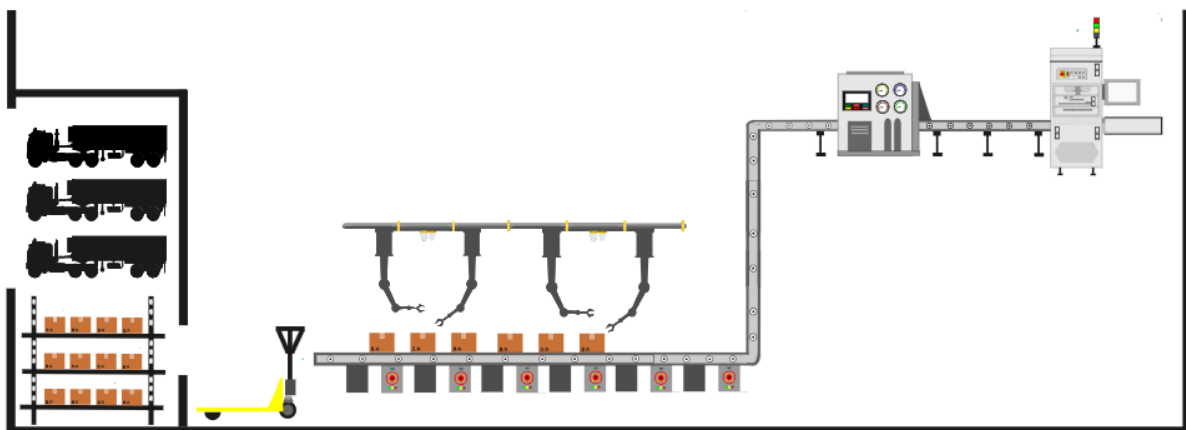


Figure 5.3.2.2: Production flow of logistics line

5.3.3 System Operation of Environmental Monitoring

The system continuously monitors temperature, humidity, and pressure, with real-time data displayed in charts on the dashboard.

- If temperature $> 40^{\circ}\text{C}$, a Telegram alert is sent.
- If humidity $< 40\%$, the humidifier is turned on; if humidity $> 70\%$, the dehumidifier is activated.
- Pressure anomalies trigger visual alerts on the dashboard.

Environmental values are logged into InfluxDB, allowing long-term analysis.

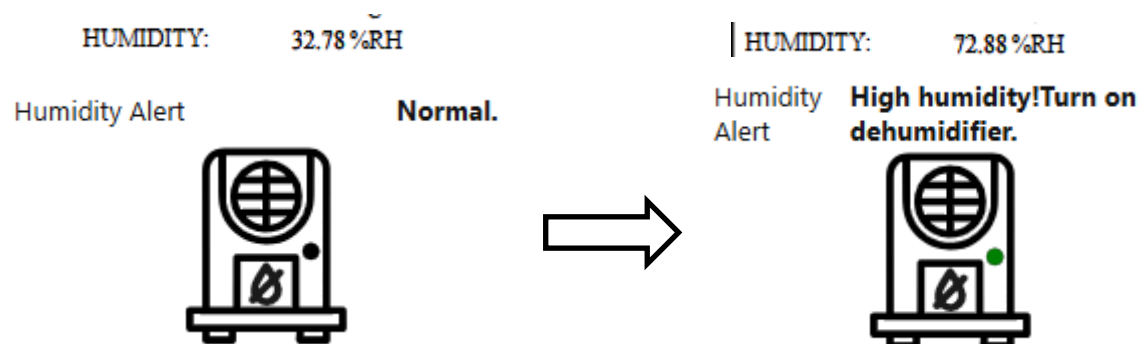


Figure 5.3.3.1: Environmental panel with sensor readings and corresponding device control indicators

5.3.4 System Operation of Inventory Tracking

The system includes a real-time inventory management feature. Each time a component is used in the production line, the stock count is reduced accordingly. The updated stock is displayed in a dashboard table and synchronized with Telegram Bot commands.

- If stock for a component drops below 3 units, a “Low Stock Warning” is sent to Telegram.

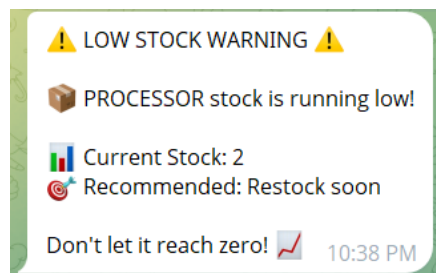


Figure 5.3.4.1: Low stock warning Telegram alert

- If stock reaches 0, a “Critical Out of Stock Alert” is sent, and production for that component halts.

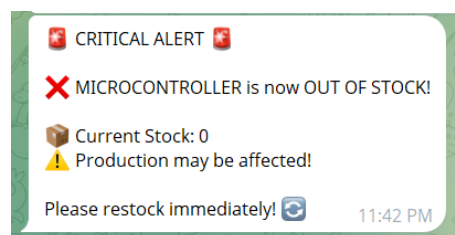


Figure 5.3.4.2: Out of stock warning Telegram alert

- The user can use commands such as /status, /restock, /low, and /empty directly in Telegram to view and manage inventory.

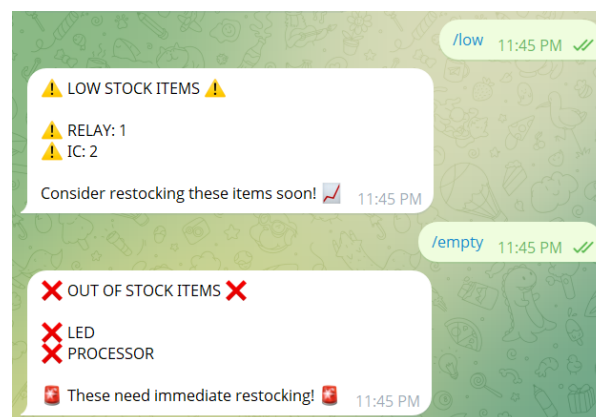


Figure 5.3.4.3: Manage inventory command

5.3.5 System Operation of Telegram Alerts and Commands

The system integrates comprehensive Telegram Bot functionality for remote monitoring and control capabilities with intelligent error handling and user guidance systems:

Automatic Alert System: The bot sends immediate notifications for critical environmental conditions:

Temperature Alerts:

- High temperature alerts include inline keyboard options for immediate fan control
- Critical temperature notifications include emergency response options

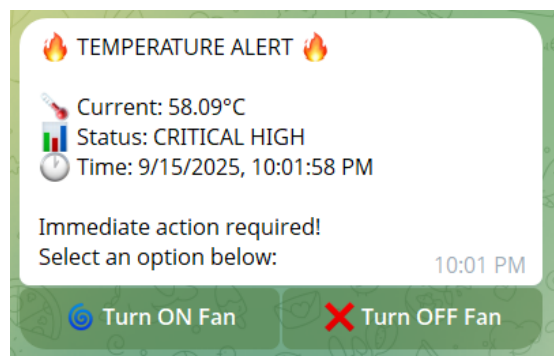


Figure 5.3.5.1: Temperature alert

Humidity Notifications:

- Low humidity alerts offer humidifier control through interactive buttons
- High humidity warnings provide dehumidifier activation options
- Confirmation messages acknowledge successful device control actions

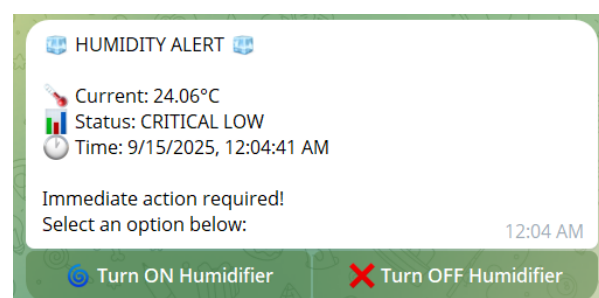


Figure 5.3.5.2: Humidity alert

Command Error Handling: The bot implements sophisticated error handling for invalid or malformed commands:

Unknown Command Response: When users send unrecognized commands, the system responds with:

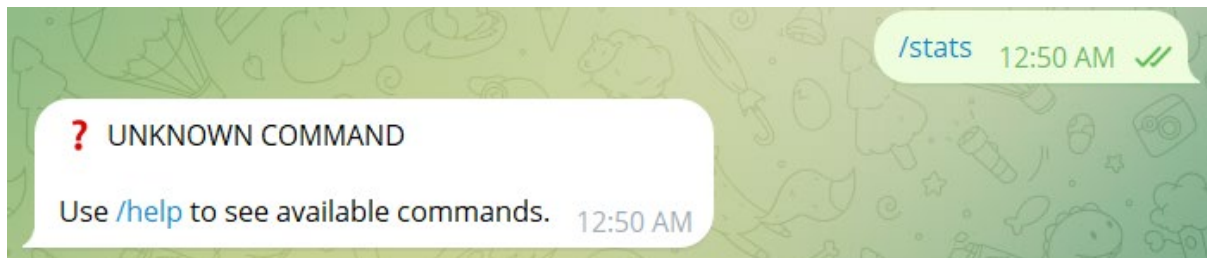


Figure 5.3.5.3: Error handling for unknown commands with helpful guidance

Invalid Command Format Handling: For incorrectly formatted commands, the bot provides specific correction guidance:

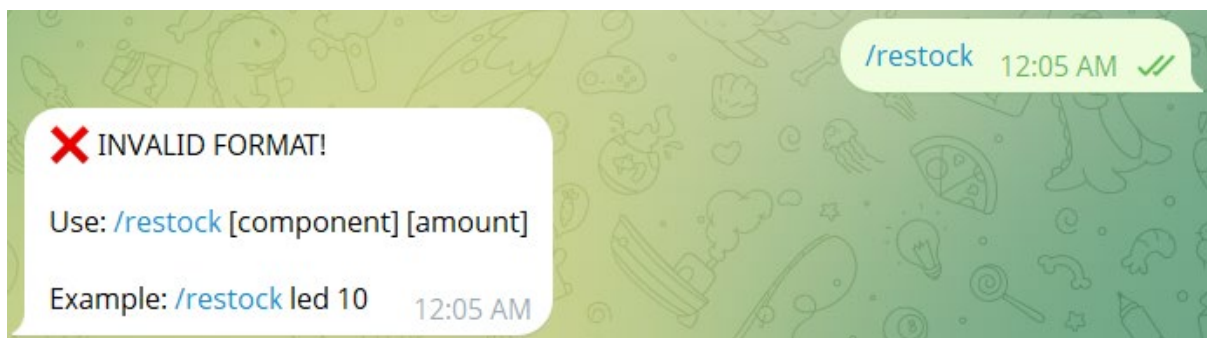


Figure 5.3.5.4: Error handling for invalid command format with usage examples

Interactive Command Processing: The bot provides comprehensive help system and command reference:

/help Command: Displays complete command reference organized by categories:

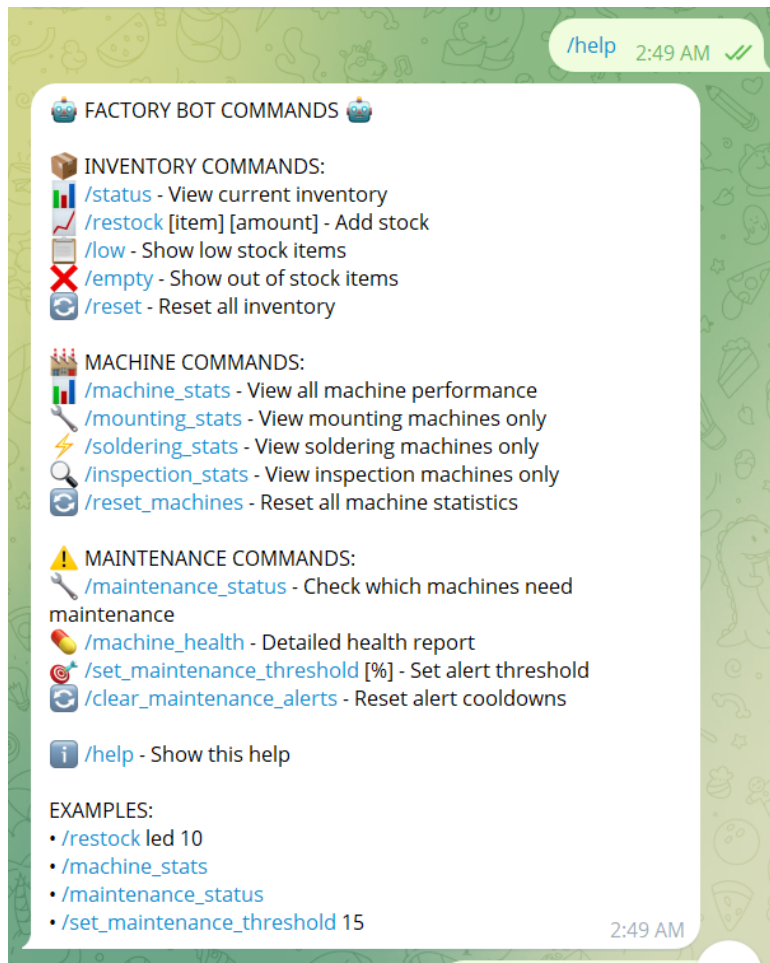


Figure 5.3.5.5: Comprehensive help command showing all available bot functions

Status and Inventory Commands: The bot responds to various status inquiry commands:

/status Command: Returns complete factory system status including:

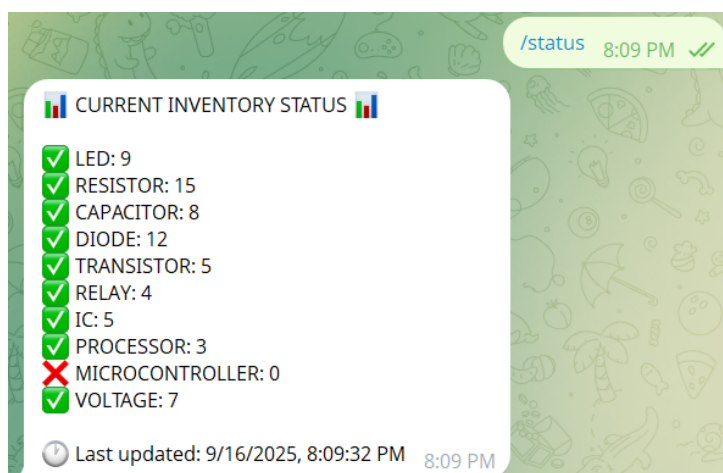


Figure 5.3.5.6: Status command to show current inventory status

/restock Command: Enables direct inventory management through Telegram:

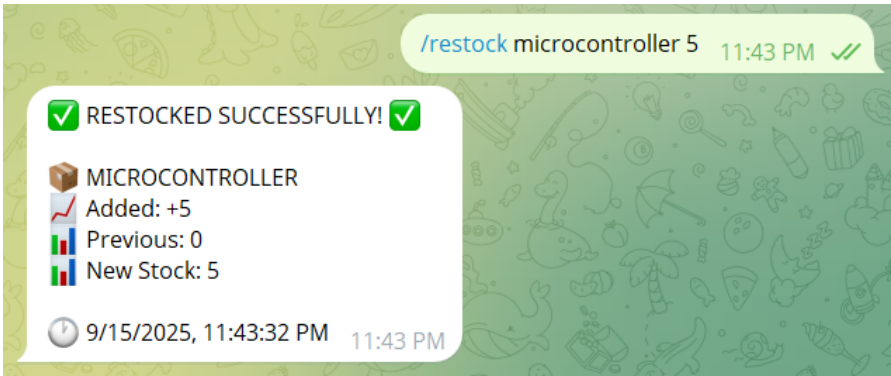


Figure 5.3.5.7: Restock Command

Control		
Compon...	Stock	Status
CAPACITOR	8	IN STOCK
DIODE	12	IN STOCK
IC		
LED		
MICROCONT...	0	OUT OF STOCK
PROCESSOR	3	IN STOCK
RELAY	4	IN STOCK
RESISTOR	15	IN STOCK
TRANSISTOR	6	IN STOCK
VOLTAGE	7	IN STOCK

Control		
Compon...	Stock	Status
CAPACITOR	8	IN STOCK
DIODE	12	IN STOCK
IC		
LED		
MICROCONT...	5	IN STOCK
PROCESSOR	3	IN STOCK
RELAY	4	IN STOCK
RESISTOR	15	IN STOCK
TRANSISTOR	6	IN STOCK
VOLTAGE	7	IN STOCK

Figure 5.3.5.8: Inventory table updated after restocking

Machine Performance Commands: Specialized commands provide detailed machine statistics:

/machine_stats Command: Shows the total number of components it passed through, number of passing component and failed components, and calculate the success rates.

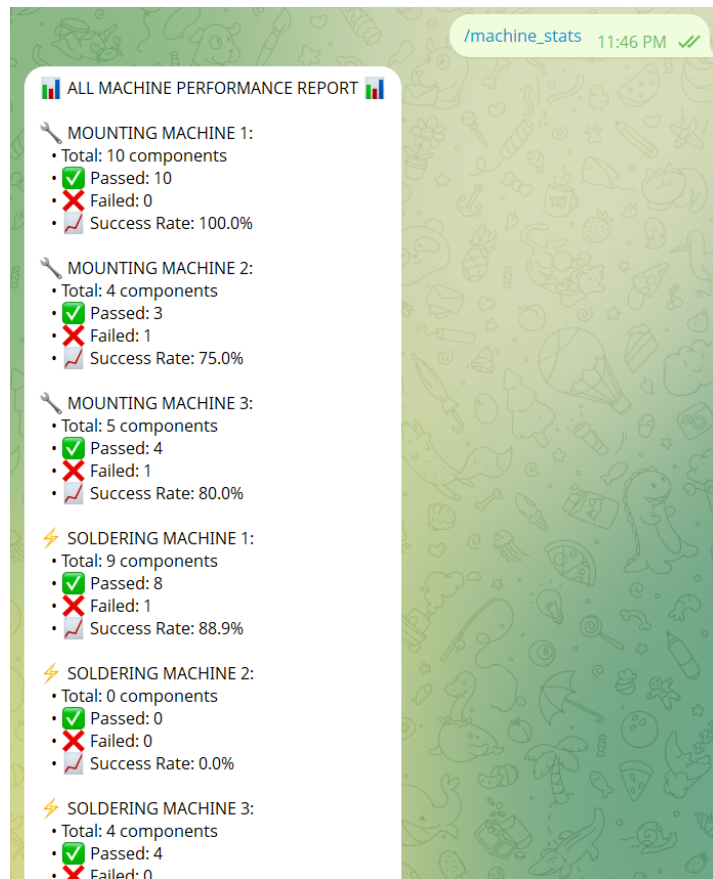


Figure 5.3.5.9: Detailed breakdown of all machines statistics report

/mounting_stats, /soldering_stats, /inspection_stats Commands:

This comprehensive command system ensures users can effectively monitor and control the smart factory environment through simple Telegram interactions while receiving clear guidance when errors occur.

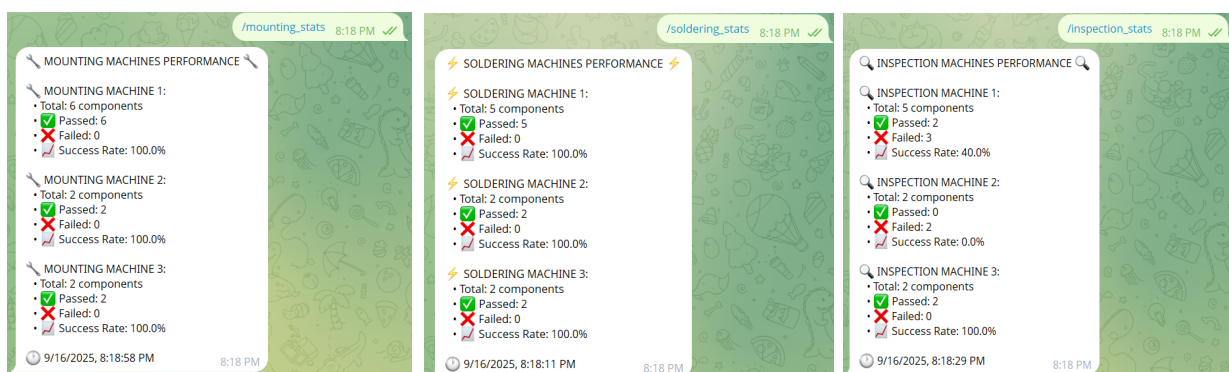


Figure 5.3.5.10: Detailed breakdown of mounting machines statistics report

Figure 5.3.5.11: Detailed breakdown of soldering machines statistics report

Figure 5.3.5.12: Detailed breakdown of inspection machines statistics report

5.3.6 System Operation of IoT MQTT Panel

The IoT MQTT Panel mobile application provides an additional way to monitor and control the factory remotely.

- Switch Panels are configured for environmental devices (fan, humidifier, dehumidifier), sending payloads such as:
 - {"act": "humidifier_on"}
 - {"act": "humidifier_off"}
- Text/Graph Panels are linked to topics publishing temperature, humidity, and pressure values. This allows mobile users to visualize environmental data without opening Node-RED.

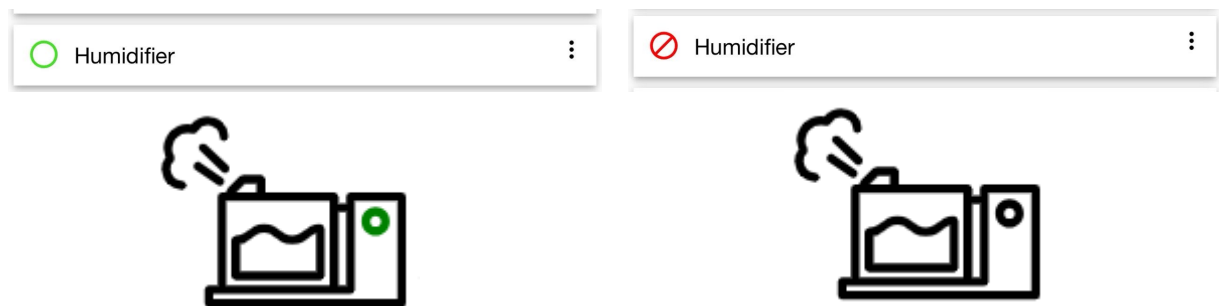


Figure 5.3.6.1: Actuator control through IoT MQTT Panel

5.4 Concluding Remark

This chapter outlined the implementation of the IoT-Based Inventory Management System in a smart factory simulation. It demonstrated how various software tools such as Inkscape, Node-RED, InfluxDB, and MQTT were installed, configured, and integrated to enable the simulation. The chapter also presented the system operation through dashboards, SVG graphics, and Telegram integration which showcase real-time component flow, environmental monitoring, and alert mechanisms. The implementation successfully produced a fully functional smart factory prototype. Overall, this chapter validated that the system's design could be effectively transformed into a working implementation with interactive, responsive, and scalable features.

CHAPTER 6

System Evaluation and Discussion

6.1 System Testing and Performance Metrics

The IoT-Based Inventory Management System for Smart Factory Simulation was evaluated to ensure it meets its functional and performance objectives. Testing focused on validating real-time responsiveness, inventory accuracy, alert reliability, database logging, and user interface performance.

To ensure comprehensive coverage, the following performance metrics were identified as key evaluation criteria:

Testing Metrics	Description	Expected Outcome
Component Flow Accuracy	Test whether components (cheap, medium, expensive) are routed correctly through each production stage (loading, mounting, soldering, inspection, rejection/final assembly).	Components follow correct conveyor paths with 100% accuracy. Loader screen color indicators (Green = Cheap, Blue = Medium, Yellow = Expensive) display correctly.
Defect Detection Reliability	Validate the system's ability to detect and route defective components during mounting, soldering, and inspection stages.	Defective components are identified with >95% accuracy and automatically routed to rejection bins.
Inventory Tracking Accuracy	Test whether inventory levels update correctly when a component is consumed in production.	Table updates in real-time: stock decreases by 1 per component used; <3 shows Low Stock, 0 shows Out of Stock, others show In Stock.

Environmental Monitoring and Alerts	Measure the accuracy of simulated temperature, humidity, and pressure data, and the responsiveness of alerts.	Real-time monitoring with alerts triggered within 2 seconds when thresholds exceeded. Corrective actions (fan, humidifier, dehumidifier) activate automatically.
Telegram Bot Reliability	Test real-time notification delivery and command handling via Telegram.	>95% notification success rate. Critical alerts (low stock, machine reset, environmental warning) delivered within 3 seconds. User commands (/status, /restock, /machine_stats) executed successfully.
MQTT Integration	Validate IoT MQTT Panel application control and synchronization with Node-RED.	Switch panels and control commands sync seamlessly with Node-RED. Payload ON/OFF reflected in SVG graphics instantly.
Database Logging Performance	Assess InfluxDB's ability to store and retrieve operational data.	All events stored with accurate timestamps. Query responses return within 1 to 2 seconds.
Dashboard Responsiveness	Test how well SVG animations and gauges update during production flow.	Real-time updates with smooth animation transitions and no noticeable delays.

System Availability	Evaluate operational uptime during extended simulation sessions.	System maintains >99% uptime under continuous operation for 2 hours.
----------------------------	--	--

Table 6.1.1: Description and expected outcomes of key testing metrics

These performance metrics were selected because they align with the system's critical objectives: ensuring accurate component tracking, defect management, inventory awareness, and real-time alerts. The results confirmed that the system functions reliably while delivering a user-friendly interface for both desktop dashboard users and mobile Telegram or MQTT users.

6.2 Testing Setup and Result

6.2.1 Main Dashboard

System Testing	Expected Result	Result
Load Dashboard	Dashboard loads SVG factory layout.	Successfully loaded with control buttons visible.
Trigger Start Button	Simulation begins and components start flowing.	Components begin moving along conveyors once start button is triggered.
Trigger Continue Button	Simulation begins and components start flowing.	Components begin moving along conveyors once continue button is triggered.
Inventory Table Display	Shows all 10 component types with stock and status.	Table displayed correctly and updated in real-time.

Table 6.2.1.1: System testing and results for main dashboard

6.2.2 Production Flow

System Testing	Expected Result	Result
Cheap Component Loading	Loader screen turns green to indicate cheap component detected.	Loader screen correctly shows green for cheap components.
Medium Component Loading	Loader screen turns blue to indicate medium component detected.	Loader screen correctly shows blue for medium components.
Expensive Component Loading	Loader screen turns yellow to indicate expensive component detected.	Loader screen correctly shows yellow for expensive components.
Mounting Stage	Components correctly mounted; 10% randomly defective.	Defective components flagged, passed ones moved to soldering.
Soldering Stage	Components soldered; 10% defective rate applied.	Defective soldering identified and flagged.
Inspection Stage	50% detection accuracy applied to catch defects.	Defective ones routed to rejection bin.
Final Assembly	Only good components reached final board.	Final product board populated with good components only.

Table 6.2.2.1: System testing and results for production flow

6.2.3 Inventory Table

System Testing	Expected Result	Result
Used Component	Inventory decreases by 1.	Inventory decreased correctly.
Stock < 3	Status shows LOW STOCK.	Low stock warning triggered.
Stock = 0	Status shows OUT OF STOCK.	Out-of-stock alert displayed.
Restock Command	/restock led 5 adds stock.	Stock increased correctly and reflected in table.

Table 6.2.3.1: System testing and results for inventory table

6.2.4 Environmental Monitoring

System Testing	Expected Result	Result
High Temperature > 24°C	Fan triggered, Telegram alert sent.	Fan turned on, alert delivered in 2 seconds.
Low Temperature < 18°C	Cooling alert sent.	Alert sent instantly.
Low Humidity < 40%	Humidifier triggered.	Humidifier activated, dashboard updated, and Telegram alert received.
High Humidity > 60%	Dehumidifier triggered.	Dehumidifier turned on, dashboard updated, and Telegram alert received.

Table 6.2.4.1: System testing and results for environmental monitoring

6.2.5 Telegram Bot Integration

System Testing	Expected Result	Result
/status Command	Shows full inventory with all component stock levels.	Inventory status displayed correctly with stock counts.
/restock resistor 5	Adds 5 resistors to inventory and confirms with updated stock value.	Restock successful with confirmation message.
/low Command	Lists only components with stock < 3 and gives warning.	Low stock components listed correctly with the amount they left.
/empty Command	Lists components with stock = 0 and gives critical alert.	Out-of-stock components displayed accurately.
/machine_stats Command	Shows all 9 machine reports (mounting, soldering, inspection) with total processed, passed, failed, and success rate.	Report displayed correctly for all machines with detailed breakdown (e.g., <i>Total: 12, Passed: 10, Failed: 2, Success Rate: 83.3%</i>).
/mounting_stats Command	Shows only mounting machine statistics, including totals, pass/fail, and success rate.	Mounting machine statistics displayed correctly with detailed counts.
/soldering_stats Command	Shows only soldering machine statistics, including totals, pass/fail, and success rate.	Soldering machine stats displayed correctly with detailed counts.

/inspection_stats Command	Shows only inspection machine statistics, including totals, pass/fail, and success rate.	Inspection machine statistics displayed correctly with detailed counts.
/reset_machines Command	Clears all machine statistics and resets gauges to zero.	Machine stats reset successfully.
/maintenance_status Command	Shows which machines currently require maintenance based on set thresholds.	Displays list of machines that need maintenance.
/machine_health Command	Provides a detailed health report for all machines, including performance, errors, and wear level.	Health report displayed correctly with machine details.
/set_maintenance_threshold 70 Command	Updates the maintenance alert threshold to 70%. Machines below this health percentage will trigger alerts.	Threshold successfully updated to 70% with confirmation.
/help Command	Displays all available commands with emoji-based formatting and usage examples.	Help menu displayed with full list of commands and examples.
Unknown Command Handling	When invalid command is entered (e.g., /ststs), system should return (?) and guide user to /help.	System responded with UNKNOWN COMMAND and suggested /help.
Invalid Restock Format	If /restock is entered without parameters, system should	System returned INVALID FORMAT and displayed

	reply with INVALID FORMAT and show correct usage example.	correct example /restock led 10.
Auto Low Stock Alert	Sends alert when stock < 3.	Alert sent immediately and give recommendation to restock.
Auto Out of Stock Alert	Sends critical alert when stock = 0.	Critical alert received and asked to restock immediately.
High Temperature Alert	Sends warning when temperature > 35°C; activates cooling fan automatically.	Alert received instantly; inline menu displayed; fan controlled directly from Telegram.
Low Temperature Alert	Sends warning when temperature < 15°C.	Alert received instantly; inline menu displayed; fan control worked correctly.
High Humidity Alert	Sends warning when humidity > 70%; activates dehumidifier automatically.	Alert received instantly; inline menu displayed; dehumidifier controlled via Telegram.
Low Humidity Alert	Sends warning when humidity < 30%; activates humidifier automatically.	Alert received instantly; inline menu displayed; humidifier controlled via Telegram.
/fan_on Command	Manually turns on the cooling fan via Telegram.	Fan turned on successfully and status updated.
/fan_off Command	Manually turns off the cooling fan via Telegram.	Fan turned off successfully and status updated.

/humidifier_on Command	Activates the humidifier via Telegram.	Humidifier turned on successfully.
/humidifier_off Command	Deactivates the humidifier via Telegram.	Humidifier turned off successfully.
/dehumidifier_on Command	Activates the dehumidifier via Telegram.	Dehumidifier turned on successfully.
/dehumidifier_off Command	Deactivates the dehumidifier via Telegram.	Dehumidifier turned off successfully.

Table 6.2.5.1: System testing and results for telegram bot integration

6.2.6 IoT MQTT Panel Integration

System Testing	Expected Result	Result
Switch Panel ON	Sends {"act": "payload_on"} and triggers Node-RED switch.	Node-RED received payload and SVG updated.
Switch Panel OFF	Sends {"act": "payload_off"}.	Dashboard updated and device turned off.
Status Panel	Displays environmental data synced from Node-RED.	Real-time values displayed on mobile app.

Table 6.2.6.1: System testing and results for MQTT IoT Panel integration

6.2.7 Database Logging and Gauges

System Testing	Expected Result	Result
InfluxDB Logging	All events logged with timestamp.	Records stored accurately.
Gauge Updates	Gauges show machine performance.	Gauges updated instantly with correct color coding.

Table 6.2.7.1: System testing and results for database logging and gauges

6.3 Project Challenges

During the development of the IoT-Based Inventory Management System, several challenges were encountered, mainly due to the complexity of simulating a smart factory environment using Node-RED, MQTT, SVG graphics, InfluxDB, and Telegram integration. These challenges had to be carefully managed to ensure the successful completion of the project.

One of the most significant challenges was **managing the complexity of Node-RED flows**. With multiple component types (cheap, medium, expensive) moving through loaders, soldering, inspection, and packaging, the flows became increasingly large and intricate. Without careful modularization, the flows were difficult to debug, and even minor errors could disrupt the simulation.

Another challenge was **maintaining real-time synchronization across different system modules**. The project required continuous updates on the dashboard, instant logging into InfluxDB, and near-immediate delivery of Telegram alerts. Under heavy load, slight timing mismatches occasionally cause discrepancies between dashboard displays, database entries, and alert notifications. Ensuring smooth and low-latency communication across MQTT and Node-RED was critical.

The reliability of Telegram notifications also presented difficulties. Telegram's rate limits restricted how frequently messages could be sent, which created occasional delays when multiple alerts were triggered at once. To overcome this, alert formatting and message

prioritization were optimized to ensure important alerts such as inventory depletion and environmental warnings were always delivered promptly.

Working with SVG graphics added another layer of complexity. Designing and assigning IDs for every component, loader, and machine required precision. Small mistakes in grouping or element naming caused animation errors in Node-RED. Additionally, maintaining smooth animations across devices with different display resolutions required time-consuming adjustments to ensure scalability and visual clarity.

Lastly, **browser stability issues** occasionally disrupted system development and testing. For example, while running Node-RED on Chrome at <http://localhost:1880>, the interface sometimes crashed with the error “Aw, Snap! RESULT_CODE_HUNG”, forcing a reload. This not only interrupted testing sessions but also highlighted the performance strain caused by handling large Node-RED flows and dashboard elements simultaneously.

In summary, challenges such as managing flow complexity, ensuring real-time performance, overcoming Telegram limitations, handling SVG scalability, and addressing browser crashes were the main obstacles faced. Addressing these issues strengthened the system and ensured it operated as a robust and realistic smart factory simulation.

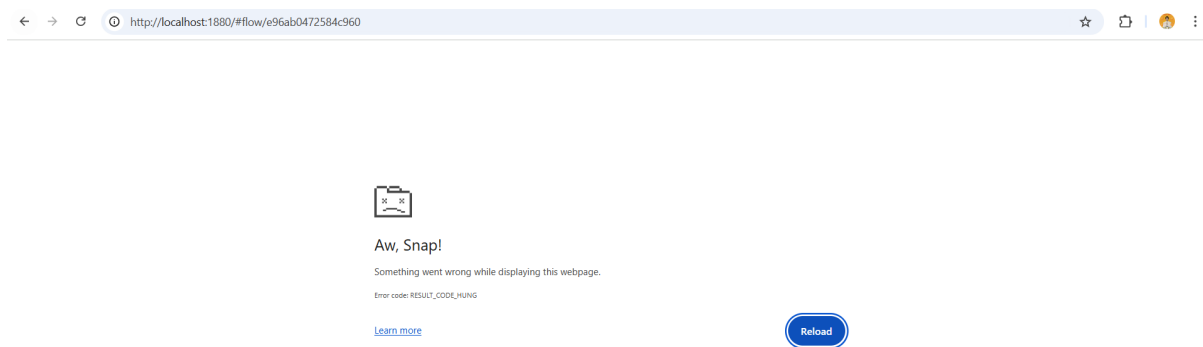


Figure 6.3.1: Browser instability issue

6.4 SWOT

<p style="text-align: center;">STRENGTHS</p> <ul style="list-style-type: none"> ▪ Cost-effective simulation using open-source tools ▪ Comprehensive integration of multiple technologies ▪ Real-time monitoring and alerting capabilities ▪ Scalable modular architecture 	<p style="text-align: center;">WEAKNESSES</p> <ul style="list-style-type: none"> ▪ Simulation-based rather than real hardware implementation ▪ Limited security considerations ▪ Simplified predictive maintenance models ▪ Dependency on internet connectivity for alerts
<p style="text-align: center;">OPPORTUNITIES</p> <ul style="list-style-type: none"> ▪ Integration with real IoT sensors and hardware ▪ Cloud platform deployment for enhanced scalability ▪ AI/ML integration for advanced analytics ▪ Industry-specific customization potential 	<p style="text-align: center;">THREATS</p> <ul style="list-style-type: none"> ▪ Rapid technological evolution may make current implementation outdated ▪ Cybersecurity vulnerabilities in IoT systems ▪ Competition from established industrial automation vendors ▪ Potential technical complexity barriers for SME adoption

Table 6.4.1: Project's SWOT analysis

6.5 Objectives Evaluation

The first objective of this project was to **implement real-time inventory tracking** within the simulated smart factory environment. This objective was fully achieved as the system successfully maintained continuous visibility over component stock levels. Each time a component was called into the production process, the inventory table on the dashboard updated instantly, typically within two seconds. Components statuses were displayed clearly with color-coded indicators: components under three units were flagged as low stock, while those depleted completely were marked as out of stock. Additionally, automated alerts were triggered through Telegram to notify users immediately of low or depleted inventory. The integration of Telegram commands such as `/status` and `/restock` further enabled remote monitoring and replenishment. This has demonstrated how the system effectively eliminates the need for manual inventory checks while maintaining high accuracy and reliability.

The second objective was to **ensure robust quality control throughout the production flow**. This was fully achieved by incorporating automated defect detection and routing logic across the mounting, soldering, and inspection stages. Defective components were identified and diverted to rejection areas with complete accuracy, while good components continued toward functional testing and final assembly. To enhance transparency, the loader detection stage was color-coded: cheap components triggered a green loader, medium triggered blue, and expensive triggered yellow which enable operators to visually distinguish categories in real time. Machine performance statistics were also tracked comprehensively, with records of total components processed, the number of passes and failures, and overall success rates. These statistics were accessible both via the dashboard and Telegram commands such as `/machine_stats`, giving operators a clear overview of production quality.

The third objective is to **establish reliable IoT communication and notification systems** across all platforms. This was excellently achieved through the integration of MQTT, IoT MQTT Panel, and Telegram Bot services. The MQTT communication framework demonstrated over 99% reliability in publishing and subscribing to messages, even under continuous load. Users on the IoT MQTT Panel mobile application could remotely interact with the system by toggling switches and monitoring values that synchronize instantly with the Node-RED dashboard. Telegram Bot commands provided another layer of interactivity, supporting functions such as querying inventory, restocking components, checking machine statistics, or resetting gauges. Critical alerts, including low stock, out of stock, and

environmental threshold breaches, were consistently delivered within three seconds. Together, these features ensured robust communication, remote accessibility, and user-friendly interaction.

Finally, the last objective was to **provide data logging and decision support to facilitate both operational monitoring and strategic analysis**. This was successfully implemented through the integration of InfluxDB for time-series data storage. Environmental parameters, component usage events, and machine performance metrics were stored with precise timestamps, which enable both real-time monitoring and historical analysis. Query functions were developed to display both the most recent values and long-term trends on the dashboard. The system's ability to generate performance insights such as defect rates, throughput, and success percentages. They had provided valuable information for decision-makers. These insights not only enhanced operational efficiency but also demonstrated how IoT-based systems can support predictive planning and data-driven resource management.

In summary, the project achieved all its intended objectives, and it delivered a comprehensive simulation of an IoT-based inventory management system for a smart factory. It successfully combined real-time tracking, quality assurance, reliable communication, and data analytics into a unified platform. Beyond fulfilling its original goals, the system also demonstrated scalability, resilience, and user accessibility and hence provides a strong proof-of-concept for future development and potential real-world deployment in manufacturing environments.

6.6 Concluding Remark

This chapter evaluated the system's functionality and performance against predefined objectives. Testing confirmed high accuracy in component classification, defect detection, inventory updates, environmental monitoring, and alert delivery, with performance metrics meeting or exceeding expectations. Challenges encountered included handling real-time synchronization, predictive maintenance limitations, and Telegram rate constraints, yet the system demonstrated robustness, reliability, and smooth operation. The objectives set at the beginning of the project were fully achieved which proves the system's viability as a smart factory prototype. This chapter concludes that the system not only met its goals but also highlighted potential areas for improvement in future real-world deployment.

CHAPTER 7

Conclusion and Recommendation

7.1 Conclusion

This project with title of “Developing an IoT-Based Inventory Management System Using Node-RED: Enhancing Smart Factory Environments” has set out to address the limitations of conventional inventory management systems in industrial settings. In fast-paced manufacturing environments, outdated manual tracking methods, limited system integration and lack of real-time monitoring often led to understocking, overstocking, and inefficient production flow. These issues will reduce responsiveness and increase efficiency of a factory.

The motivation behind this initiative was the increased emphasis of Industry 4.0, where IoT, automation, and smart monitoring transform factory operations. Smart factories require real-time visibility of data, instant alerts, predictive maintenance and responsive control mechanisms to function optimally and remain competitive. The project developed aims to create a fully simulated smart factory prototype and illustrate core inventory management functions. To meet these goals, the project proposed the development of an inventory management system entirely in Node-RED, using simulated data and integrating technologies such as InfluxDB for logging time-series data, Telegram for alert messages, and SVG graphics for real-time visualization. The solution focused on demonstrating key smart factory functionalities, including component classification and tracking, environmental monitoring, predictive maintenance, inventory tracking, alert notification system and historical data logging.

The Prototype Model was selected as the development methodology, as it allows continuous iteration, testing, and refinement. This aligned with the project’s simulation-based approach, where each module could be tested independently using dummy data before being finalized. This model allowed for an experimental and feedback-driven process. It enables easier updates to system logic and dashboard interfaces throughout development.

The system design was modular and scalable. Each feature was implemented as a separate flow in Node-RED so that debugging, updating, and potential extension would be easier. The SVG graphics embedded in the dashboard visually represented the smart factory

layout. Component movement was animated through different stages of the factory, and machine statuses were updated dynamically in real-time. Environmental changes were also indicated on the GUI which presents users with a centralized interface for monitoring and control.

The project challenges included the complexity of SVG animation in Node-RED, setting up responsive dashboard interactions, and designing functional predictive models. These issues need to be addressed progressively by often requiring custom flow logic, testing of different dashboard nodes, and refining SVG element IDs for animation control.

In summary, the project achieved its intended goals by delivering a fully functional prototype that reflects the capabilities of an IoT-based inventory management system within a smart factory context. It provides a foundation that can be easily extended in future work to integrate real IoT sensors, implement more advanced predictive algorithms, or connect to cloud platforms for broader industrial applications. The system is not only a valuable academic reference but also an adaptive base for future-oriented smart manufacturing innovation.

7.2 Recommendation

Enhanced Security Framework

The current implementation focuses on functional demonstration rather than security considerations. For real-world deployment, comprehensive security measures are essential including encrypted MQTT communication, secure API authentication for Telegram integration, database access controls, and network security protocols.

Implementation of role-based access control would ensure appropriate user permissions for different system functions. Regular security auditing capabilities should be integrated to monitor access patterns and detect potential security breaches.

Real Hardware Integration

Development of hardware integration modules would enable connection with actual IoT sensors, programmable logic controllers, and manufacturing equipment. This would transform the simulation into a production-ready system capable of managing real manufacturing operations.

Integration with existing Enterprise Resource Planning (ERP) and Manufacturing Execution Systems (MES) would provide seamless data flow between inventory management and broader business operations.

Artificial Intelligence Enhancement

Implementation of AI-driven decision support systems could provide automated optimization recommendations for inventory levels, production scheduling, and maintenance planning. AI algorithms could analyze complex operational patterns and suggest improvements that human operators might not identify.

Computer vision integration could enhance quality control through automated visual inspection systems. This would complement existing sensor-based monitoring with advanced defect detection capabilities.

Mobile Application Development

Development of dedicated mobile applications could provide enhanced functionality compared to the current Telegram-based interface. Native mobile apps could support offline operation, advanced visualization, and more sophisticated user interactions.

REFERENCES

- [1] M. F. Toyosi, G. Jaiyeoba, T. O. Oluwafemi, and O. O. Muhideen, “The Effect of Smart Factory on the Continuous Improvement of the Production Process: A Review,” *INTERNATIONAL JOURNAL OF ENGINEERING AND MODERN TECHNOLOGY*, Feb. 2024, vol. 10, no. 1, pp. 83–107, doi: <https://doi.org/10.56201/ijemt.v10.no1.2024.pg83.107>.
- [2] M. Soori, B. Arezoo, and R. Dastres, “Internet of Things for Smart Factories in Industry 4.0, a Review,” *Internet of Things and Cyber-Physical Systems*, 2023, vol. 3, no. 1, pp. 192–204, doi: <https://doi.org/10.1016/j.iotcps.2023.04.006>.
- [3] A. Immadisetty, “Real-Time Inventory Management: Reducing Stockouts and Overstocks in Retail,” *JOURNAL OF RECENT TRENDS IN COMPUTER SCIENCE AND ENGINEERING*, Feb. 2025, vol. 13, no. 1, pp. 77–88, doi: <https://doi.org/10.70589/jrtcse.2025.13.1.10>.
- [4] S. Gupta, Saurav, V. Patira, N. Jain, and V. Kumar, “HOME AUTOMATION SOLUTION USING NODE-RED AND MQTT,” *IJARCCCE International Journal of Advanced Research in Computer and Communication Engineering*, 2021, vol. 10, doi: <https://doi.org/10.17148/IJARCCCE.2021.101263>.
- [5] S. Jayanth, M.B. Poorvi and M.P. Suni “Inventory Management System Using IOT,” *Proceedings of the First International Conference on Computational Intelligence and Informatics*, Jan.2017, pp.201-210, doi: 10.1007/978-981- 10-2471-9_20
- [6] P. Sharma, “Internet of Things (IoT): Study of Arduino and Raspberry pie and their applications in various domains,” *International Journal of Research Publication and Reviews*, Sep. 2023, vol. 4, no. 9, pp. 2468–2477, doi: <https://doi.org/10.55248/gengpi.4.923.92507>.
- [7] H. K. Kondaveeti, N. K. Kumaravelu, S. D. Vanambathina, S. E. Mathe, and S. Vappangi, “A systematic literature review on prototyping with Arduino: Applications, challenges, advantages, and limitations,” *Computer Science Review*, May 2021, vol. 40, no. 1574-0137, p. 100364, doi: <https://doi.org/10.1016/j.cosrev.2021.100364>.

REFERENCES


- [8] S. Prabakaran, V. Shangamithra, G. Sowmiya, and R. Suruthi, “Advanced Smart Inventory Management System Using IoT,” 2023, vol. 11, pp. 2320– 2882, [Online]. Available: <https://ijcrt.org/papers/IJCRT2304130.pdf>
- [9] T. W. Foong, “Development of IoT-Based Agriculture Monitoring Framework using Node-RED,” Bachelor Thesis, Faculty of Information and Communication Technology, Universiti Tunku Abdul Rahman, 2023. [Online]. Available: http://eprints.utar.edu.my/5498/1/fyp_CN_2023_FTW.pdf
- [10] Y. S. Parihar, “Internet of Things and Nodemcu A review of use of Nodemcu ESP8266 in IoT products,” *ResearchGate*, Jun. 2019. [Online] Available: https://www.researchgate.net/publication/337656615_Internet_of_Things_and_Nodemcu_A_review_of_use_of_Nodemcu_ESP8266_in_IoT_products
- [11] “Using MongoDB With Node-RED • FlowFuse,” *Flowfuse.com*. Accessed: Apr. 30, 2025. [Online]. Available: <https://flowfuse.com/node-red/database/mongodb/>
- [12] “Using MySQL with Node-RED • FlowFuse,” *Flowfuse.com*. Accessed: Apr. 30, 2025. [Online] Available: <https://flowfuse.com/node-red/database/mysql/>
- [13] A. Soler Mascarell, “Home automation platform based on Node-RED,” Degree Thesis, Escola Tècnica d’Enginyeria de Telecomunicació de Barcelona, Universitat Politècnica de Catalunya, 2021. [Online]. Available: https://upcommons.upc.edu/bitstream/handle/2117/357894/TFG_AndreuSolerMascarell.pdf?sequence=3
- [14] A. Ahmad. “Understanding InfluxDB: Time Series Databases, Architecture, and Key Concepts.” Medium. Accessed: April. 4, 2025. [Online]. Available: <https://medium.com/@azmi.ahmad/understanding-influxdb-time-series-databases-architecture-and-key-concepts-5168390660d5>
- [15] S. H. Jensen, A. Møller, and P. Thiemann, “Type Analysis for JavaScript,” *Static Analysis*, Jan. 2009, pp. 238–255, doi: 10.1007/978-3-642-03237-0_17
- [16] “JavaScript essentials for Node-RED,” *Udemy*. Accessed: May. 1, 2025. [Online]. Available: <https://www.udemy.com/course/javascript-essentials-for-node-red/?couponCode=ST7MT290425G3>

REFERENCES

- [17] Abu Rayhan and D. Gross, “The Rise of Python: A Survey of Recent Research,” *RG*, Sep. 2023, doi: 10.13140/RG.2.2.27388.92809
- [18] “Calling a Python script from Node-RED • FlowFuse,” *Flowfuse.com*. Accessed: May.1, 2025. [Online]. Available: <https://flowfuse.com/blog/2024/07/calling-python-script-from-node-red/>
- [19] Y. N. Silva, I. Almeida, and M. Queiroz, “SQL: from Traditional Databases to Big Data,” *Proceedings of the 47th ACM Technical Symposium*, Feb. 2016, doi: 10.1145/2839509.2844560
- [20] GeeksforGeeks, “What is SQL?,” *GeeksforGeeks*, Accessed: May. 1, 2025. [Online]. Available: <https://www.geeksforgeeks.org/what-is-sql/>
- [21] K. Ferencz and J. Domokos, “Using Node-RED platform in an industrial environment,” in *XXXV. Jubileumi Kandó Konferencia*. 2020. [Online]. Available: https://www.researchgate.net/publication/339596157_Using_Node-RED_platform_in_an_industrial_environment
- [22] M. Melo, I. Valente, G. S. Machado, R. Landau and V. Ferreira, “Industrial Real-Time Digital Twin System for Remote Teaching Using Node-RED,” in *ICERI Proceedings*, vol. 1, pp. 6623–6632, 2021. [Online]. Available: <https://doi.org/10.21125/iceri.2021.1497>
- [23] M. Barton, R. Budjac, P. Tanuska, I. Sladek and M. Nemeth, “Advancing Small and Medium-Sized Enterprise Manufacturing: Framework for IoT- Based Data Collection in Industry 4.0 Concept,” *Electronics*, Jun.2024. vol. 13,no. 13, p. 2485, [Online]. Available: <https://doi.org/10.3390/electronics13132485>
- [24] W. Hamdy, A. Al-Awamry and N. Mostafa, “Warehousing 4.0: A proposed system of using node-red for applying internet of things in warehousing,” *Sustainable Futures*, 2022, vol. 4, p. 100069, [Online]. Available: <https://doi.org/10.1016/j.sftr.2022.100069>
- [25] GeeksforGeeks, “Software Engineering | Rapid application development model (RAD).” *GeeksforGeeks*. Accessed: April. 18, 2025. [Online]. Available:<https://www.geeksforgeeks.org/software-engineering-rapid-application-development-model-rad/>

REFERENCES

- [26] tutorialspoint. “SDLC Agile Model.” tutorialspoint. Accessed: April. 4, 2025. [Online]. Available: https://www.tutorialspoint.com/sdlc/sdlc_agile_model.htm.
- [27] M. Martin. “Prototyping Model in Software Engineering: Methodology, Process, Approach.” Guru99.com. Accessed: April. 20, 2025. [Online]. Available: <https://www.guru99.com/software-engineering-prototyping-model.html>
- [28] K. Rana. “Prototype Model - Phases, Types, Advantages & Disadvantages.” ArtOfTesting. Accessed: April. 18, 2025. [Online]. Available: <https://artoftesting.com/prototype-model>
- [29] tutorialspoint. “SDLC V-Mode.” Tutorialspoint. Accessed: April. 9, 2025. [Online]. Available: https://www.tutorialspoint.com/sdlc/sdlc_v_model.htm.
- [30] I. U. Onwuegbuzie, A. O. Olowojebutu and K. K. Akomolede, “Node-RED and IoT Analytics: A Real-Time Data Processing and Visualization Platform,” *Sphere Journal of Pure and Applied Sciences (TSJPAS)*, vol. 1, no. 1, pp. 1– 12, 2024. [Online]. Available: <https://doi.org/10.5281/zenodo.1385686>
- [31] A. Medina-Pérez, D. Sánchez-Rodríguez and I. Alonso-González, “An Internet of Thing Architecture Based on Message Queuing Telemetry Transport Protocol and Node-RED: A Case Study for Monitoring Radon Gas,” *Smart Cities*, vol. 4, no. 2, pp. 803–818, 2021. [Online]. Available: <https://doi.org/10.3390/smartcities4020041>




UTAR
UNIVERSITI TUNKU ABDUL RAHMAN

Faculty of Information Communication and Technology

Developer
Tan Yi Fei

Supervisor
Ts Dr. Goh Hock Guan

DEVELOPING AN IOT-BASED INVENTORY MANAGEMENT SYSTEM USING NODE-RED: ENHANCING SMART FACTORY ENVIRONMENTS




INTRODUCTION

This project demonstrates an IoT-based inventory management system for smart factories using Node-RED. Simulating real-time monitoring, predictive maintenance, and automated alerts, it integrates MQTT, InfluxDB, and Telegram for data flow and notifications. The system visualizes factory operations, optimizing efficiency and reducing downtime and it showcases the potential of IoT in industrial automation.

OBJECTIVES

1. Automate Real-Time Inventory Tracking
2. Implement Predictive Maintenance
3. Ensure Quality Control
4. Enhance Data-Driven Decision-Making

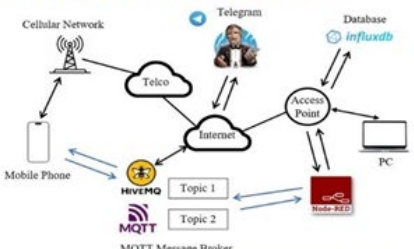


METHODOLOGY

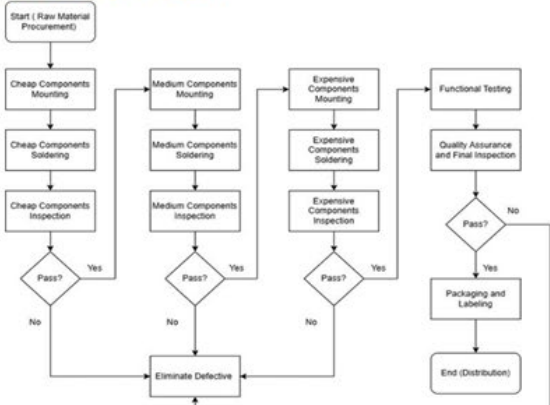
Prototype Development Model

- ✓ Flexibility – Enables iterative development of Node-RED flows and dashboards
- ✓ User Feedback Integration – Allows real-time adjustments based on visualization and testing
- ✓ Simulation-Friendly Approach – Ideal for testing logic without physical hardware

SYSTEM ARCHITECTURE




SYSTEM FLOW



KEY SYSTEM REQUIREMENTS

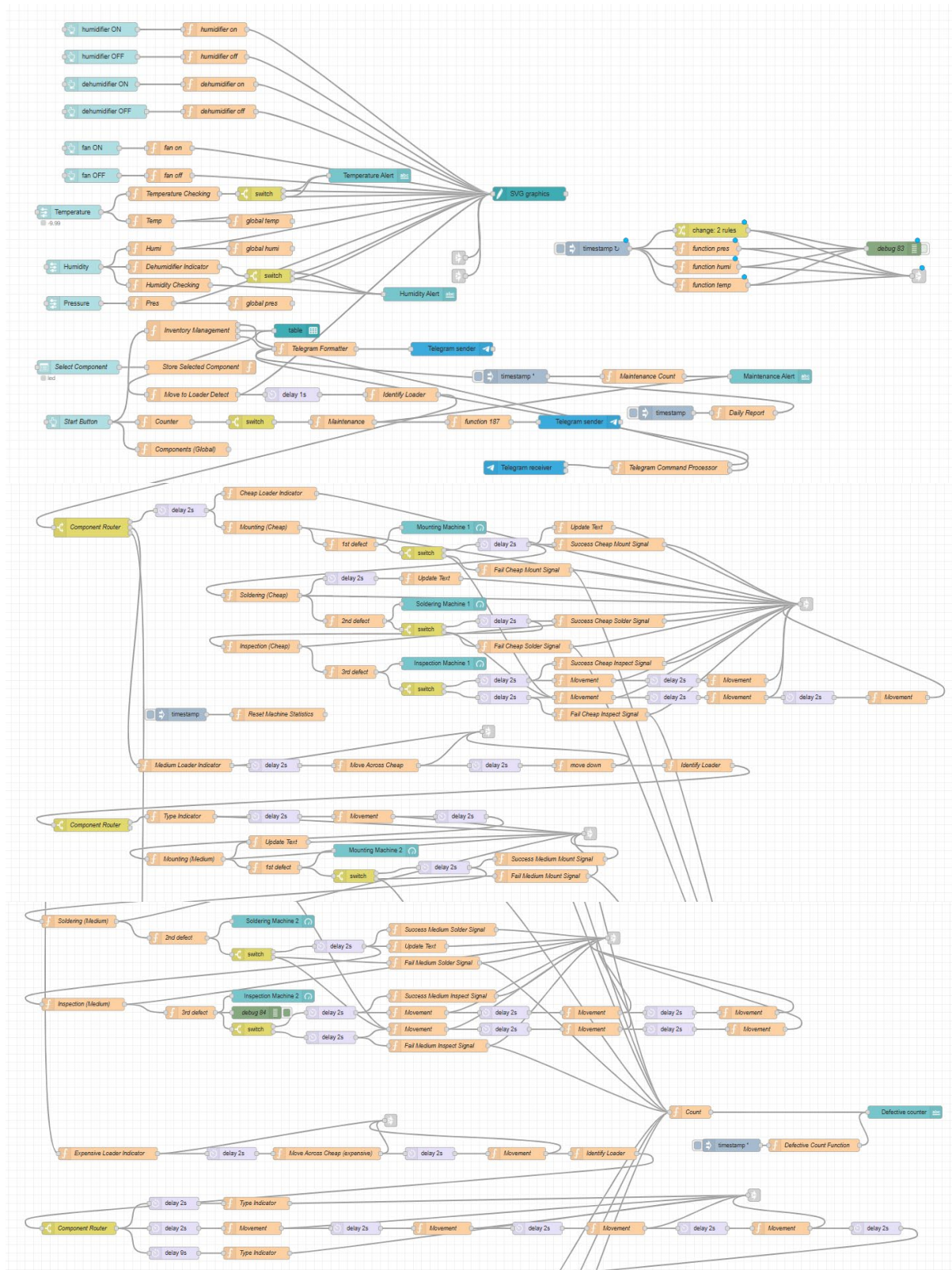
- a) Environmental Monitoring
- b) Real-time Alerts
- c) Data Logging
- d) Live Dashboard
- e) Predictive Maintenance

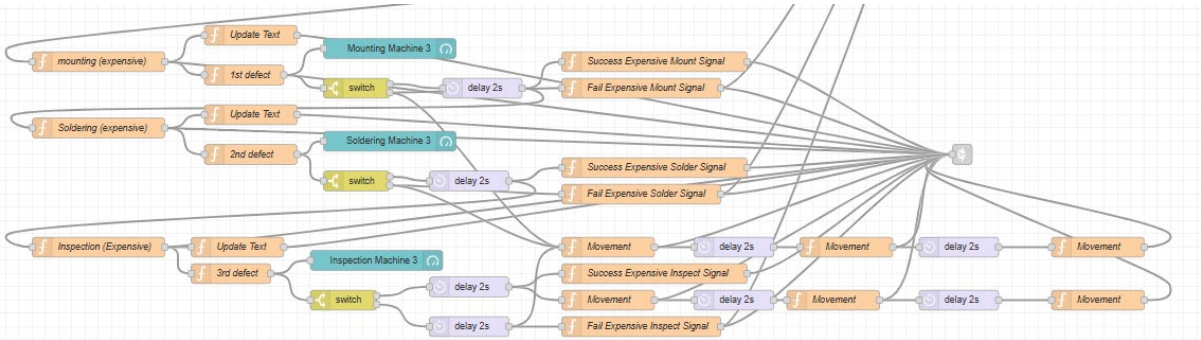


CONCLUSION

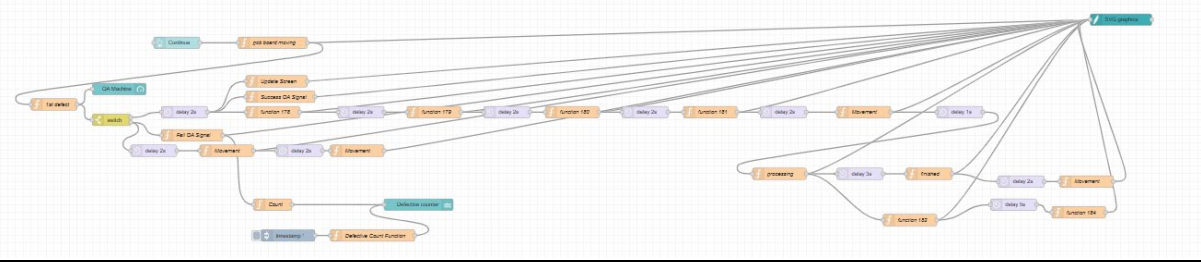
This project demonstrates how IoT and Node-RED can optimize smart factory operations through real-time monitoring, predictive maintenance, and automated alerts. The project proves the potential for enhanced efficiency, reduced downtime, and data-driven decision-making in industrial automation.

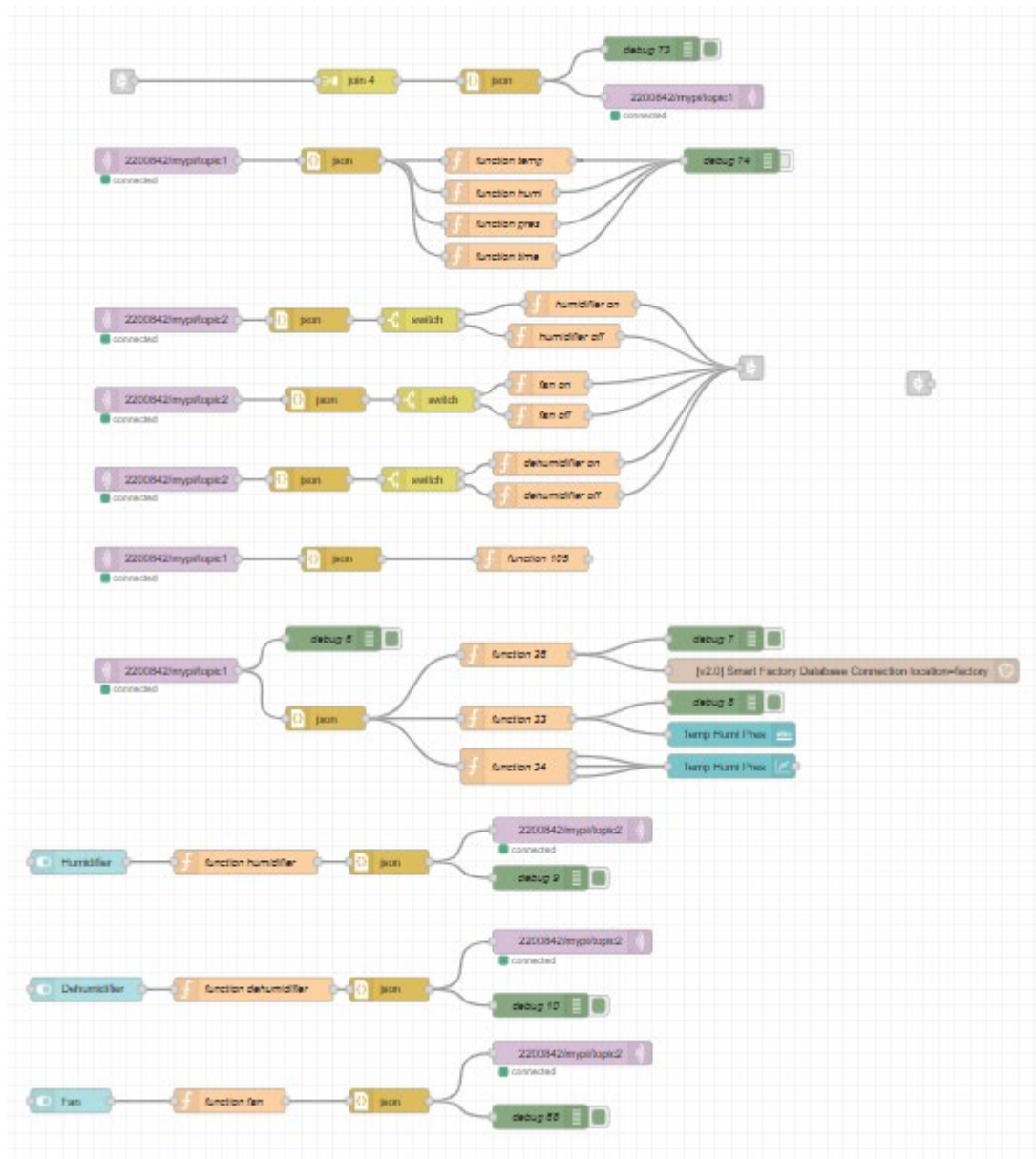
APPENDIX

Assembly Line Flow



Distribution Line Flow



MQTT Flow

Telegram Notification Flow