AIN SHAMS UNIVERSITY
FACULTY OF ENGINEERING
MECHATRONICS DEPARTMENT

Graduation Project

Integrating IIOT and Digital Twin with Industrial Automation

Submitted by:

Ahmed Ebrahim Elsayed Hassan

Mohamed Abd El Hakeem El said

Mohamed Elsadek Metwally Ebrahim

Omar Hamdy Gbr Mohamed

Supervised By:

Prof.Dr. Mohamed Ibrahim

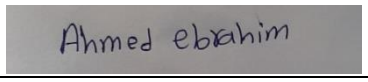
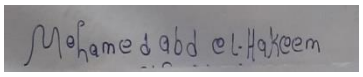
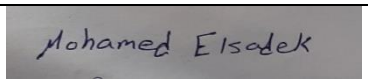
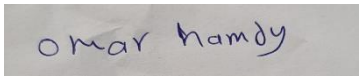
Eng. Ahmed Salah

June 2024

Declaration

We/I hereby certify that this Project submitted as part of our/my partial fulfilment of BSc in (Mechatronics Engineering) is entirely our/my work, that we/I have exercised reasonable care to ensure its originality, and does not to the best of our/my knowledge breach any copyrighted materials, and have not been taken from the work of others and to the extent that such work has been cited and acknowledged within the text of our/my work.

Signed: by all students

Name: Ahmed Ebrahim El Sayed	Signature: 
Name: Mohamed Abd El Hakeem El Said	Signature: 
Name: Mohamed El Sadek Metwally	Signature: 
Name: Omar Hamdy Gbr	Signature: 

Date: 24/6/2024

Acknowledgment

We would like to take this opportunity to express our heartfelt gratitude to Prof. Mohamed Ibrahim Awad for allowing us this great opportunity to work on this project and for his constant guidance and support throughout the year. It was a great honor to work under his supervision and have him entrust us. To our parents for their unwavering support and encouragement, for all their motivation and inspirational words and actions throughout this long academic journey. We would also like to extend our gratitude to ENG: Ahmed Salah for their continuous suggestions, and constructive criticism.

Abstract

This report represents the progress throughout the graduation project concerning the SCADA and Digital Twin & IIoT automation of industrial process control. The main target outcome of this project is understanding the system functionality, components, and control design, which was successfully achieved by studying the system flow and the role of each component and tracing the highly complex wire connections inside the control panel and the controller box. After running some tests, some components were found to be damaged either during the moving process of the system or due to being left for a long time without maintenance, therefore they were replaced or repaired. Some issues were only discovered after powering up the system and observing the system's behavior. The report also includes the next phase, which was improving the system by adding PLC, SCADA, and IIoT, the problems that were faced, how it was solved, and step-by-step how to reach the goal. For the PLC part, three PLCs were used. For the SCADA part, due to problems faced in communication, all the PLCs are connected to the same computer at the same time. For the IIoT part, the communication between PLCs with each other and trying to make the master control that can take any action on the system and reading the data from the system and then the many types of protocols that PLCS have them from this side, on another side ESP32 Can it handle any of these Protocols for transferring the data from PLCs to the cloud? , also the little resources available for the idea, and finally trying to find a free cloud to make Iiot of our system on it, where the most is the commercia. We used A digital twin to make a virtual representation of system designed to reflect a physical object accurately. It spans the object's lifecycle, is updated from real-time data, and uses simulation, machine learning, and reasoning to help make decisions. We use Azure platform, the Azure cloud platform is the superior choice due to its comprehensive suite of services, which includes Azure IoT Hub, Azure Functions, Azure Event Hubs, and Azure Digital Twins, among others. In the following sections, we will detail each of these services.

Contents

List of figures

List of Tables

List of Symbols

D1	Tank
D2	**Feed tank**
C1	**Column**
E1	**Heat exchanger**
M1	**The motor of the heat exchanger fan**
M2	**Stirrer of the tank**
J1	**Electrical resistance**
TA1	**Thermostat for temperature in the tank**
TI1	**RTD for temperature measurement after cooling**
CE1	**Conductivity indicator**
G1	**Centrifugal pump feeding column**
G2	**Centrifugal pump recirculating back to the tank**
TI2	**RTD for thermostat**
WT1	**Two load cells**
FT1	**Flow-rate transmitter**
LT1	**Differential-pressure transmitter**
PA1	**Minimum pressure switch**
PA2	**Maximum pressure switch**
FC1	**Pneumatic control valve for flow controls out of the tank**
LC1	**Pneumatic control valve for the level inside the column**
TT1	**Temperature transmitter**

1. Introduction

The multi-process control system, specifically the Mod. UNIPRO/EV model made by Electronica Veneta -an Italian company- encompasses various components and functionalities, designed about 30 years ago. The system comprises a glass column that is supplied with liquid through a centrifugal pump. The pump is interconnected with a tank incorporating an electric resistor for heating. Additionally, another centrifugal pump is responsible for recirculating the water from the column to the tank through an air-cooled heat exchanger.

To regulate and monitor the system, a PID controller known as Digitric 500, which is manufactured by ABB Instrumentation, is employed. Along with a special microprocessor made for the conductivity sensor, and the switches and indicators fixed on the side of the control panel.

And that was it for this super old-fashioned and out-of-date system and it was left rusting and aging with no proper use whatsoever. It needed help and restoration to its original form. Therefore, some modernization was critical for the system to make it more tempting for others to work with, in addition to improving its capabilities and accuracy.

The system's control capabilities have been enhanced by installing three XBCDR30SU programmable logic controllers (i.e., PLC) controllers. The controlling technique can easily be switched between the old controller or the PLCs with a click of a button. One can easily monitor the system -when controlled by the PLC system- using the XP-builder supervisory control and data acquisition (i.e., SCADA) software package. For remote monitoring from anywhere or any device, the TeamViewer application is utilized with Info for a user-friendly experience. Also, as an alternative means of control, the ESP32 WROOM module has been integrated, enabling the utilization of the Industrial Internet of Things (i.e., IIOT) platform known as Ubidots Cloud for data uploading and monitoring in the cloud.

This system will serve a respected and important mission as it would be used as an educational kit to help other students from different levels with their automation courses and most importantly, save the system with its valuable components from being thrown away.

Figure 1:Whole System

2. System Overview

The system is essentially an educational kit equipped with mechanical, electrical, and control elements allowing students to grasp the concept of multi-process control. The system's model dates to the 90s, with very limited documentation available which made understanding the system's operation and functionality quite challenging.

The road for system development has been paved in the previous semester through the following points: Understanding the system flow, control loops, and each component function, Inspection of components, connections, and maintenance/replacement of faulty ones.

2.1 SYSTEM FLOW AND COMPONENTS

The process starts with a borosilicate glass tank filled with water connected to another small feed tank that contains the solution. The two fluids can be mixed by a stirrer and heated to a specific temperature by electrical resistance.

A conductivity indicator with a probe that can withstand up to 100℃ is used to measure the mixture's conductivity. Both the mixing and heating elements can be controlled by P.C. or a manual switch on the control panel.

The temperature inside the tank is maintained at a specified set point through a thermostat. RTD pt100 sensor is used for measuring the fluid temperature coupled with a temperature transmitter that produces a 4-20 mA output signal.

A centrifugal pump feeds another borosilicate glass column, where the input flow to the column is controlled by a flow-rate transmitter and pneumatic control valve. The column is mounted on two load cells and a differential-pressure transmitter is inserted at the bottom of the column for level measurement.

Another centrifugal pump is placed after the column to recirculate the fluid into the tank. The output flow from the column is controlled by minimum and maximum pressure switches and pneumatic control valves. The fluid is passed by a heat exchanger (motor fan with tachometer generator) for cooling. The output temperature from the heat exchanger is measured by the RTD pt100 sensor.

The following diagram visualizes the process flow in detail:

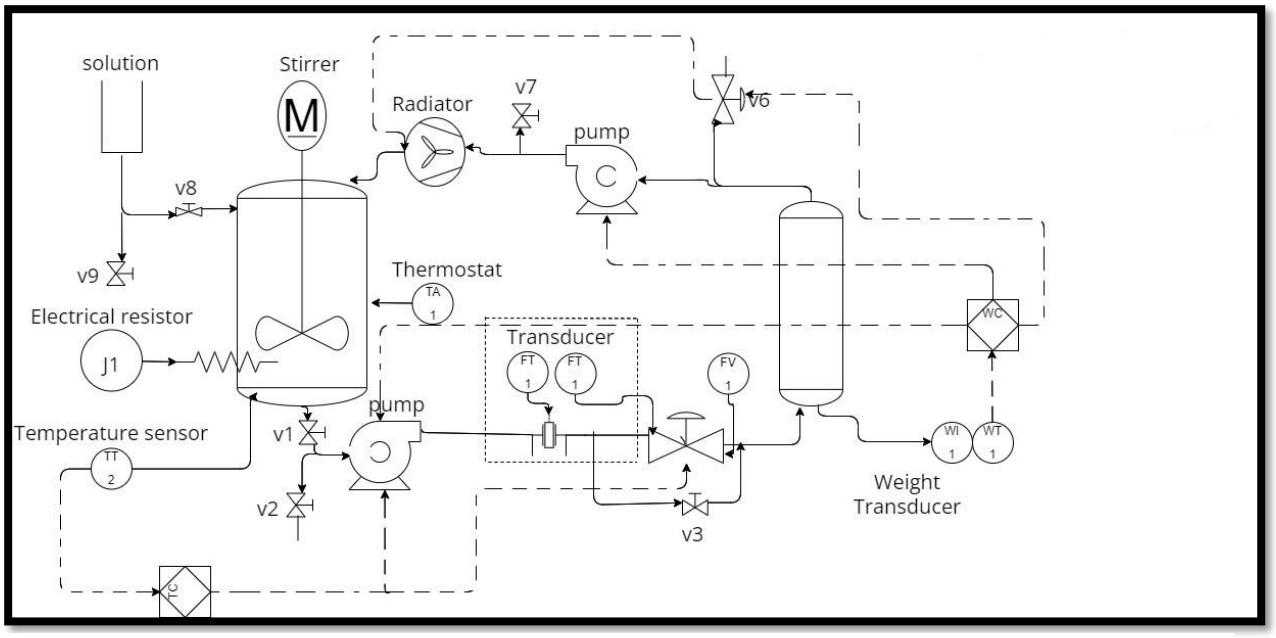


Figure 2::P&Id diagram

2.2 Wiring diagram

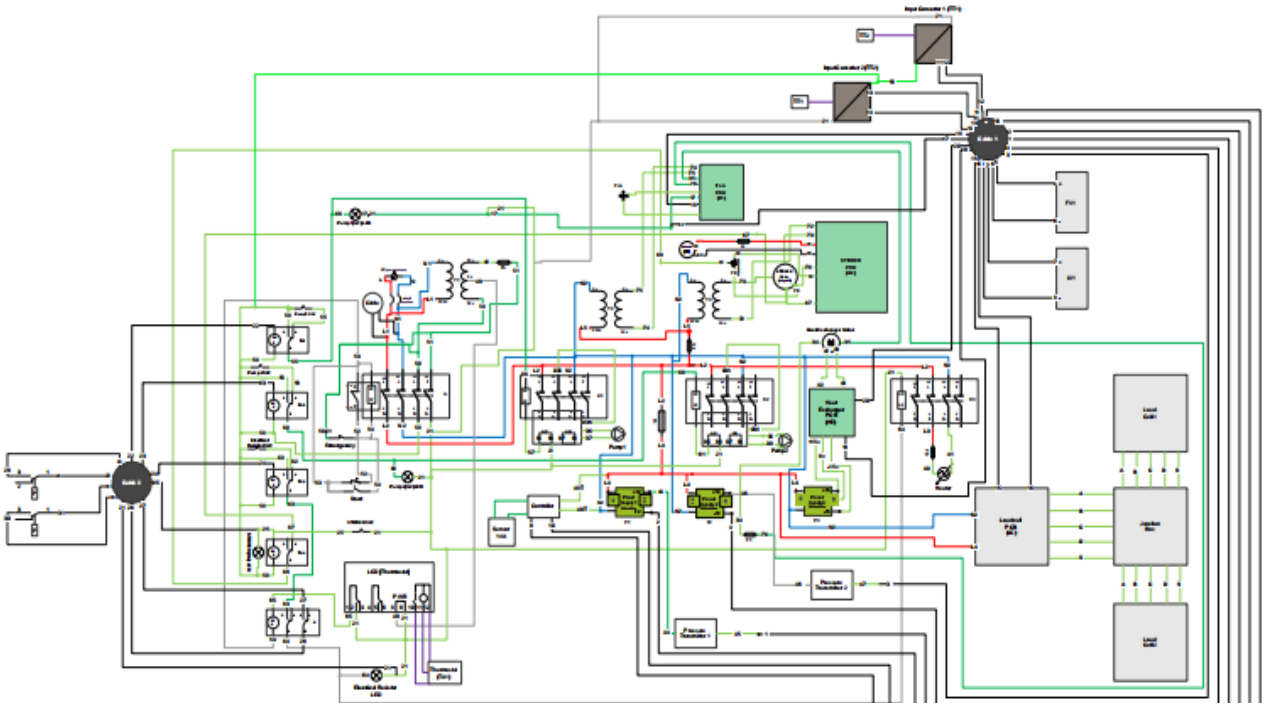


Figure 3:wiring diagram

2.3 QR-code for wiring diagram

2.4 Bill of materials

Part Name	Quantity	Specification
Old System		
Tank	1	Borosilicate Glass (20L Capacity)
Auxiliary Tank (Feed)	1	Borosilicate Glass (1L Capacity)
Column	1	Borosilicate Glass (10L Capacity)
Centrifugal Pump	2	-AISI 304 stainless steel – Qmax = 4.25 m3 /h -Hmax = 32.5 m
Air-cooled heat exchanger	1	- Motor fan speed = 1500 rpm - Tachometer generator and electronic card signal: 0-1300 rpm / 4-20 mA
Electronic flow-rate transmitter	1	- AISI 316 stainless steel - Range: 0-0.6 m3/h - Output signal: 4-20 mA
Electronic differentialpressure transmit	1	- AISI 316 stainless steel - Range: 0-500 mm H2O - Output signal: 4-20 mA
Pneumatic Control Valve	2	- AISI 316 stainless steel - Cv = 2.5
Electro-pneumatic converters	2	- 4-20 mA / 0.2-1 bar
Load Cell	2	- Range: 0-10 kg - Output signal: 4-20 mA mv/Volt indicator
Electronic Thermostat	1	- Programmable from 0-100 ℃
Conductivity transmitter/indicator	1	- Range: 0-2000 µS - Probe can withstand up to 120℃
Pressure Switch	2	- Minimum: 1.5 bar

		- Maximum 3.5 bar
RTD Sensor	**2**	**- Pt 100 - AISI 304 stainless steel**
Stirrer	**1**	**- AISI 304 stainless steel - Variable speed reducer - Reduction ratio 6:1**
Electrical Resistance	**1**	**- AISI 304 stainless steel - P = 3kW**
Temperature Transmitter	**2**	**- AISI 304 stainless steel - Output signal: 4-20 mA**
Electronic Microprocessor PID Controller	**2**	**- With 4-line LCD - Serial card**
Control Panel	**1**	**- IP55**
Transformer	**3**	**- 2transformers 230/24 V - 1 transformer 230/12 V**
Power Supply	**3**	**- 2 power supplies 12-Vdc - 1 power supply 15-Vdc**
PLC&SCADA		
XBC-DR30SU	**3**	**----------**
XB0-AD02A	**4**	**----------**
XB0-AD02A	**2**	**----------**
USB to V3	**3**	**----------**
USB to rs485	**3**	**----------**
USB hub	**1**	**----------**
IIOT		
ESP32	**1**	**----------**
PCB	**1**	**----------**
Micro USB cable	**1**	**----------**
Installation		
Palette	**1**	**----------**
Wires 1mm	**Yellow: 20 meters Blue: 20 m Red: 5 m**	**----------**
Wires 2mm	**Black: 5 m**	**----------**
Terminal Block	**40 2 for power, 6 for communication, and the rest are for the actuators and sensors**	**----------**
Rail	**1 meter**	**----------**
Duct	**1.5 meter**	**----------**

Table 1:bill of material

3. Maintenance

After powering up the system it was noticed that some components were not functioning as intended and had to be replaced/fixed. The following list summarizes these changes:

3.1 Replacing some ICs, and fixing the voltage regulator in fan PCB

- **We find some problems with this driver when testing it by Avometer.**
- **there are five circles mentioned to the components that we replace or fix them.**
- **voltage regulator in red circle while ICs in orange circle.**

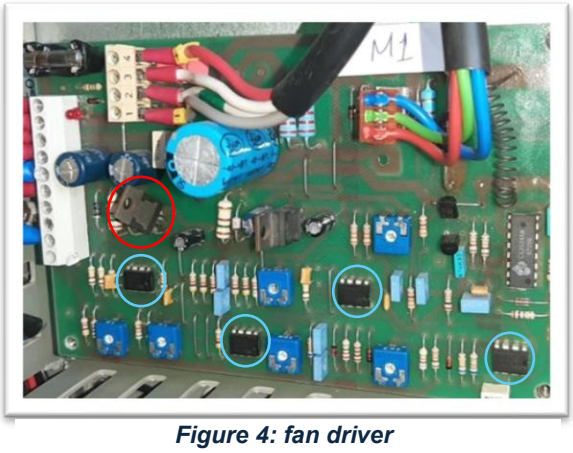


Figure 4: fan driver

- **After repairing this board, we were able to adjust the different fan speeds**

3.2 Radiator maintenance and cleaning

- **After checking the radiator, we find**

  The issue in the cooling rate so we decided that it need for repairing.
- **After maintenance, the cooling rate was increased and it began to operate very efficiently**

Figure 5: radiator

3.3 Stirrer repairing

- **Problem**
  - **The method of the coupler fixation was not correct or strong enough due to wrong fixation.**
  - **Solution**
- **We disassembled the coupler and considered creating a level surface on the motor shaft. We then attached the screw to it and fixed it between shaft of motor and shaft of the stirrer.**

Figure 6:coupler

3.4 Tank repairing

- This was perhaps the most challenging part of the hardware maintenance process. Therefore, we tried a lot to implement an effective repairing method.

Problem

There is leakage in the bottom of this tank due to break in it.

solutions

1- Make a mechanical drawing of the tank to manufacturing a new one but we find it is too expensive.
2- Make a connection between tank and pipes made by ٣d printing but it wasn't sufficient because there is a leakage again.
3- Put a Silicon in between this connection and the tank, and between this connection and the pipe but it is not good enough.
4- We reduce the length of this connection and putting a silicon also and we find that there is no leakage until temperature 40C
5- Finaly we used material named pattex PL150 it prevent leakage until 80C.

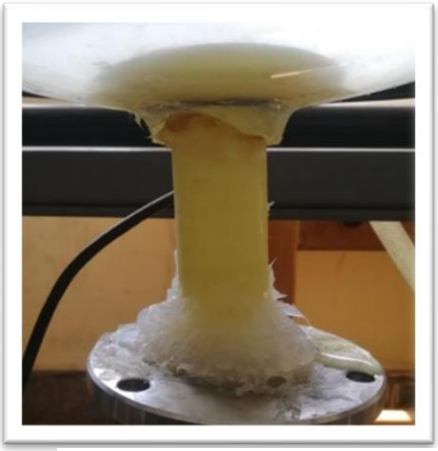


Figure 7:tank repairing 3

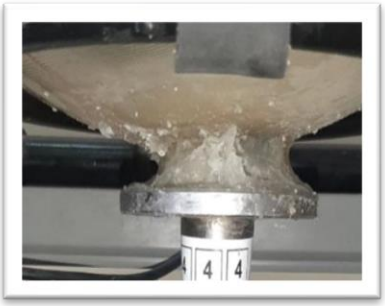


Figure 9:Magnification of Tank repairing 4

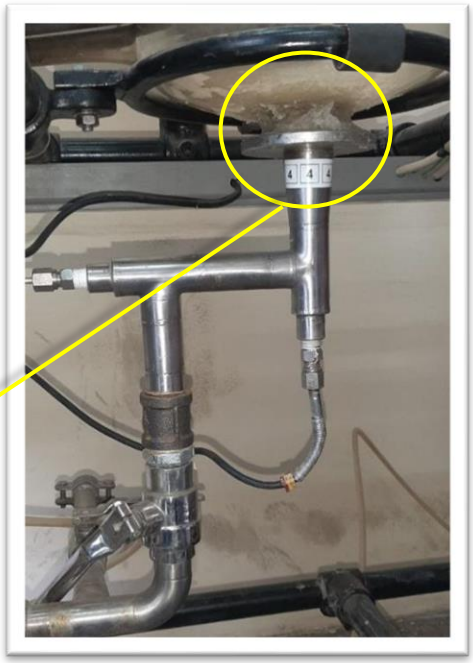


Figure 8:tank repairing 4

Figure 10: Tank** repairing **5

3.5 Calibration for sensors

3.5.1 Load cell

Hysteresis is a phenomenon where the output of a sensor not only depends on its current input but also on the history of its previous inputs. This can lead to different readings for the same input value, depending on whether the input value is increasing or decreasing. Hysteresis is an important factor to consider in the design and application of sensors, as it can affect the accuracy and reliability of measurements.

Firstly we make calibration for load cell by take reading while project is ON we take readings every 2 seconds during loading and unloading and draw Hysteresis curve

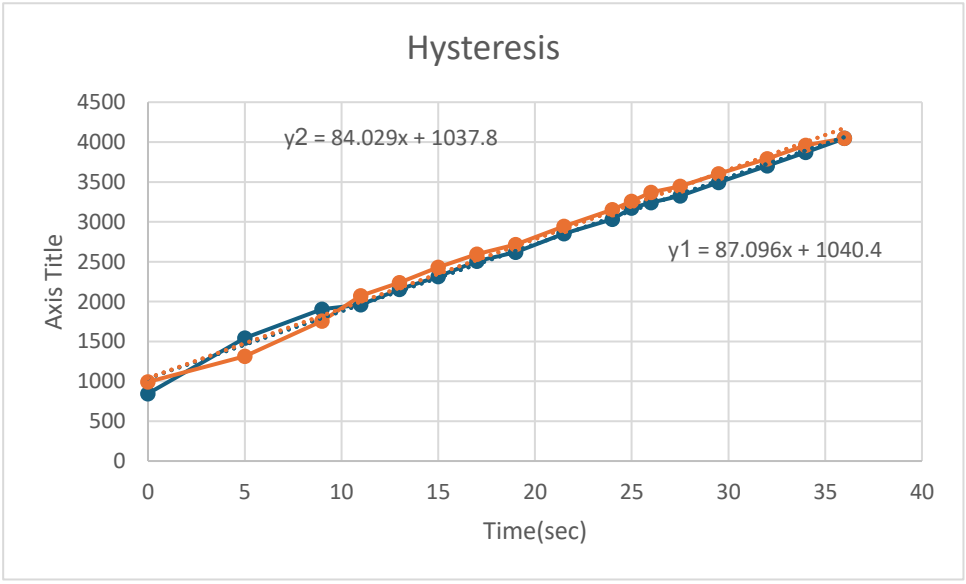


Figure 11:Hysteresis curve

3.5.2 Temperature sensor

We get thermometer and take samples of water at different temperatures and used it to take accurate measurements from temperature sensor.

4. PICs

4.1 Introduction

At this phase, the system is to be controlled by programmable logic controllers (PLCs) and using Analog Digital Converters [ADC] to communicate with the analog sensors, heat exchanger motor, and the two pneumatic valves. The PLCs from LS Electric of model XBCDR30SU, one of them is the Master and the others are Slaves. Two ADCs connected to each PLC.

A PLC or a programmable logic controller is an industrial computer that automates electromechanical processes like factory assembly lines. PLCs are anticipated to function perfectly for years in hazardous industrial environments. It mainly consists of a CPU, memory, and I/O modules and works with a suitable programming device. Compared to old controlling systems, A PLC has the advantage of a smaller circuit with fewer components, an easier troubleshooting mechanism, and exceptional computation abilities.

4.2 System Specs

The system requires the following:

- **7 analog input channels (TT1, TT2, FT1, LT1, WT1, ST1, CE1)**
- **3 analog output channels (SC1, TC1, FC1, LC1)**
- **2 digital input channels (PA1, PA2)**
- **4 digital output channels (G1, G2, D1, J1)**

The available PLC was a LS-electric product -originated from South Korea-, model XBC-DR30SU which has the following specifications:

Rated Voltage (UL warranty voltage)	*AC 100 ∽ 240V*
Input voltage range	***AC85∽ 264V (-15%, +10%)***
Inrush current	***50APeak or less***
Input current	***0.5A or less (220V), 1A or less (110V)***
Efficiency	***65%***
Permitted momentary power failure	***10ms or less***
Power supply status indication	***LED On when the power supply is normal***
Rated Output	***1.5A @ DC5V; 0.3A @ DC24V***
Voltage Ripple	***DC5V (±2%)***
Cable specification	***0.75 ∽ 2 mm2***

Digital input	***0.18 (AWG24) - 1.5 (AWG16)***
Digital output	***0.18 (AWG24) - 2.0 (AWG14)***
Analogue I/O	***0.18 (AWG24) - 1.5 (AWG16)***
Communication	***0.18 (AWG24) - 1.5 (AWG16)***
Main power	***1.5 (AWG16) - 2.5 (AWG12)***
Protective grounding	***1.5 (AWG16) - 2.5 (AWG12)***
Terminal Strip Torque	***3.7 - 5.1 in/lb***
Dimensions in. (LxHxD)	***5.3 x 3.5 x 2.5"***

Table 2:PLC General Specifications

Operation Voltage Range	*20.4–28.8VDC (Within Ripple Rate 5%)*
Rated Input Voltage	***24VDC***
Rated Input Current	***4mA (Contact Point 0–1: 16mA, 2–7: 10mA)***
Input Resistance	***5.6kΩ (P00–P07: 2.7kΩ)***
On Voltage/On Current	***19VDC or Higher/3mA or Higher***
Off Voltage/Off Current	***6VDC or Less/1mA or Less***
Response Time	
Off ➔ On / On ➔ Off	***1/3/5/10/20/70/100ms (Set by CPU Parameter) Default: 3ms***
Insulation Voltage	***560VAC rms/3 Cycle (Altitude 2000m)***
Insulation Resistance	***10MΩ or Higher***
Relay Outputs	
Rated Load Voltage/Current	***24VDC 2A/220VAC 2A, 5A/COM***
Min. Load Voltage/Current	***5VDC/1mA***
Max. Load Voltage	***250VAC, 125VDC***
Off Leakage Current	***0.1mA (220VAC, 60Hz)***
Max. On/Off Frequency	***3,600 Times/Hour***
Response Time	

Off ➔ On	**10ms or Less**
On ➔ Off	**12ms or Less**
Service Life	
Mechanical	***Rated Load Voltage/Current 100,000 Times or More***
Electrical	***200VAC/1.5A, 240VAC/1A 100,000 Times or More*** ***200VAC/1A, 240VAC/0.5A 100,000 Times or More*** ***24VDC/1A, 100VDC/0.1A 100,000 Times or More***

Table 3:I/O Specifications

Program Control Method	Fixed Scan Time, Constant Scan
I/O Control Method	***Scan Synchronous Batch Processing Method (Refresh Method), Directed by Program Instruction***
Program Language	**Ladder Diagram, Instruction List**
Processing Speed	**94μs /1K Boolean**
Program Capacity	**15k steps**
Expansion Capability	***Expansion Modules and Option Boards (Max. 7 Combined)***
Data Area	
P	**P0000–P1023F (16,384 points)**
M	**M0000–M1023F (16,384 points)**
K	**K00000–K4095F (65,536 points)**
L	**L00000–L2047F (32,768 points)**
F	**F000–F1023F (16,384 points)**

T	***100ms, 10ms, 1ms: T0000–T255 T1023 (1,024 point) (Adjustable by Parameter Setting)***
C	***C000–C1023 (1,024)***
S	***S00.00–S127.99***
D	***D0000–D10239 (10,240 word)***
U	***U00.00–U0A.31 (Analog Data Refresh Area: 352 words)***
Z	***Z000–Z127 (128 word)***
R	***R0000–R10239 (10,240 word)***
Total Programs	***128***
Tasks Available	
Initial Task	***1***
Cyclic Task	***8***
I/O Task	***8***
Internal Device Task	***8***
Operation Mode	***Run, Stop, Debug***
Self-Diagnosis Function	***Detects Errors of Scan Time, Memory, I/O***
Program Ports	***Mini DIN 6-Pin RS-232C, USB 1 Channel***
Back-up Method	***Latch Area Setting in Basic Parameter***
Built-in Functions	
PID Control	***Control by Instruction, Auto-Tuning, PWM Output, Manual Mode, Adjustable Operation Scan Time Setting, Anti-Windup, Delta MV, SV-Ramp Function, Max. 16 Loops***

Serial	
Protocol	***Dedicated Protocol, Modbus Protocol, User Defined Protocol***
Channel	***RS-232C 1 port, RS-485 1 port***
High-Speed Counter	
Performance	***1 Phase: 100kHz, 2 Channel/20kHz, 6 Channel 2 Phase: 50kHz, 1 Channel/8kHz, 3 Channel***
Counter Mode	***Based on Input Pulse and INC/DEC method – 1 Phase Pulse Input: Addition/Subtraction Counter 2 Phase Pulse Input: Addition/Subtraction Counter 2 Phase Pulse Input: Addition/Subtraction by Pulse Phase Change? (Encoder)***
Function	***Internal/External Preset, Latch Counter, Compare Output, No. of Rotation per Unit Time***
Positioning	
Control Axis/Method	***2 Axis, Position/Speed Control***
Control	***Pulse, Positioning Data: 80 Data/Axis (Operation Step No. 1–80)***
Operation	***End/Keep/Continuous, Single, Repeated Operation***
Positioning Method	***Absolute/Incremental, Address Range: –***

	2,147,483,648 to 2,147,483,647
Speed	***Max. 100kpps (Setting Range 1–100,000pps)***
Acceleration/Deceleration	***Trapezoidal Method***
Additional Functions	***Inching Operation, Speed Synchronizing, Position Synchronizing, Linear Interpolation***
Pulse Catch	***10µs 2 point (P0000–P0001), 50µs 6 point (P0002–P0007)***
Input Filtering Time Setting Range	***1, 3, 5, 10, 20, 70, 100ms***
External Interrupt	***10?: 2 Points (P00000–P00001)***
50µs:6 Points (P00002–P00007)	
RTC	***Option Board Required***

Table 4:performance specification

Since there are 0 analog input or output pins present on the PLC, 4 XBO-AD02A and 2 XBO-AD02A extension modules were used, and as a single PLC can hold up to two modules only it was inevitable to use three PLCs. The communication protocols will later be discussed

Number of Channels	2
Range Voltage	**0 to 10VDC (Input Resistance: 1MΩ minimum)**
Current	**4 to 20mA, 0 to 20mA (Input Resistance 250Ω)**
Signal Resolution	**12 bit**
Unsigned Value	**0 to 4000**
Signed Value	**-2000 to 2000**
Precise Value Voltage	**0 to 1000 (0 to 10VDC)**
Current	**400 to 2000 (4-20mA), 0 to 2000 (0 to 20mA)**
Percentile Value	**0 to 1000**
Maximum Resolution Voltage	**2.5mV (0 to 10VDC)**
Current	**5µA (0 to 20mA), 6.25µA (4 to 20mA)**

Accuracy	**±1% maximum**
Maximum Conversion Speed	**1ms / Channel + Scan Time**
Maximum Absolute Input Voltage	**DC±15V**
Current	**DC±25mA**
Average Function	**Count Average (2 to 64,000 times)**
Gain Adjustment Function	**-40 to 40**
Insulation Method	**None**

Table 5:XBO-AD02A Module Specs

Number of Channels	2
Range Voltage	**0 to 10VDC (Input Resistance: 1MΩ minimum)**
Current	**4 to 20mA, 0 to 20mA (Load 450Ω or less)**
Setting	**In User Program or I/O Parameter per Channel**
Signal Resolution	**12 bits**
Unsigned Value	**0 to 4000**
Signed Value	**-2000 to 2000**
Precise Value Voltage	**0 to 1000 (0 to 10VDC)**
Current	**400 to 2000 (4-20mA), 0 to 2000 (0 to 20mA)**
Maximum Resolution Voltage	**2.5mV (0 to 10VDC)**
Current	**5μA (0 to 20mA), 6.25μA (4 to 20mA)**
Accuracy	**±1% or less**
Maximum Conversion Speed	**1ms / Channel + Scan Time**
Insulation Method	**None**

Table 6:XBO-DA02A Module Specs

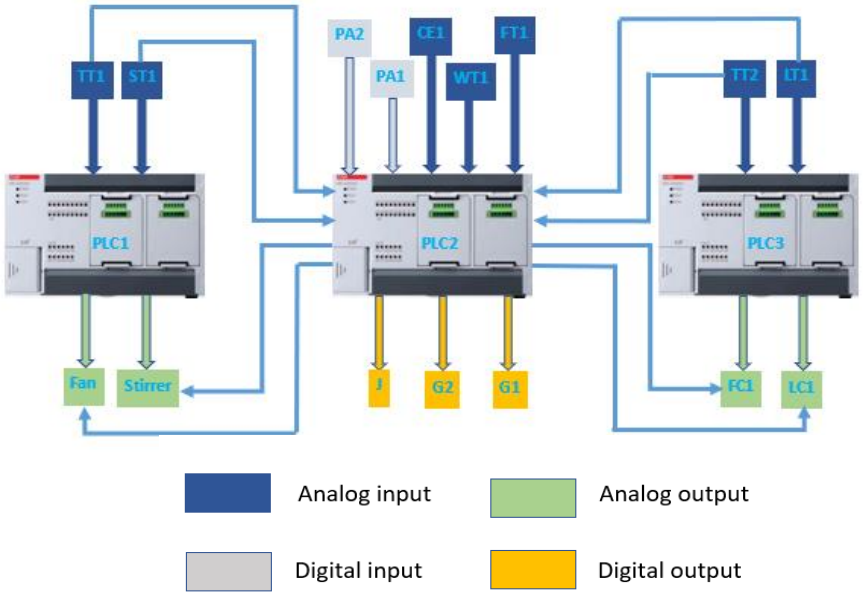


Figure 12:PLCs Inputs &Outputs

Hint: Figure 12 shows the connection between three PLCs and the illustration of how the data transfer between them will explained in detail in the communication chapter.

Sensor symbol	Sensor/Actuator	Sensor name	PIC		
			type	PIC No.	status
PIC1					
TT1	Sensor	Temp. Sensor 1	Analog	1	In
ST1	Sensor	Taco meter for fan	Analog	1	In
Fan	Actuator	Fan	Analog	1	out
Stirrer	Actuator	Stirrer	Analog	1	out
PIC2					
PA1	Sensor	Pressure sensor 1	Digital	2	In
PA2	Sensor	Pressure sensor 2	Digital	2	In
FT1	Sensor	Flow sensor	Analog	2	In
WT1	Sensor	Two load cells	Analog	2	In

CE1	**Sensor**	**Conductivity indicator**	**Analog**	**2**	**in**
G1	**Actuator**	**Pump 1 for large tank**	**Digital**	**2**	**Out**
G2	**Actuator**	**Pump 2 for small tank**	**Digital**	**2**	**Out**
J1	**Actuator**	**Electrical resistance**	**Digital**	**2**	**Out**
PIC3					
TT2	**Sensor**	**Temp. sensor 2**	**Analog**	**3**	**In**
LT1	**Sensor**	**Differential pressure**	**Analog**	**3**	**In**
FC1	**Actuator**	**Control valve (down)**	**Analog**	**3**	**Out**
LC1	**Actuator**	**Control valve(up)**	**Analog**	**3**	**out**

Table 7:Name of abbreviations for each sensor & actuator

PLC 1							
MODULE 9 (INPUT)				**MODULE 10 (OUTPUT)**			
Channel 0		**Channel 1**		**Channel 0**		**Channel 1**	
TT1		**ST1**		**Fan (TC1,SC1)**		**stirrer**	
13	**14**	**19**	**20**	**17**	**18**	**----------**	**----------**
PLC 2							
MODULE 9 (INPUT)				**MODULE 10 (INPUT)**			
Channel 0		**Channel 1**		**Channel 0**		**Channel 1**	
FT1		**WT1**		**CE1**		**----------**	
1	**2**	**15**	**16**	**9**	**10**		

PLC 3							
MODULE 9 (INPUT)				**MODULE 10 (OUTPUT)**			
Channel 0		**Channel 1**		**Channel 0**		**Channel 1**	
TT2		**LT1**		**FC1**		**LC1**	
11	**12**	**3**	**4**	**5**	**6**	**7**	**8**

Table 8:analog connections

PLC 1			
No Digital Connections.			
PLC 2			
Inputs:			
Sensor/actuator	**PORT**	**(+) labeling**	**(-)**
PA1	**P03**	**E4P**	**28**
PA2	**P04**	**E4N**	**31**
Outputs:			
G2	**P40**	**23**	**21**
G1	**P41**	**22**	**21**
J1	**P42**	**24**	**21**
PLC 3			
No Digital Connections.			

Table 9:Digital connections

4.3 Programming Software

LS Electric PLCs work with the XG5000 software tool for programming either using ladder diagrams or another language. The XG5000 was also developed by LS Electric to program and troubleshoot the XGT PLC series and has a user-friendly interface for creating, editing, and debugging ladder logic programs. It supports the following programming languages:

Figure 13:XG500

- **Ladder Diagram (LD)**
- **Instruction List (IL)**
- **Function Block Diagram (FBD)**
- **Structured Text (ST)**

XG5000 also features a simulation mode and supports online monitoring and debugging of running programs.

PLC1			
Read		**Write From PLC 2**	
TT1	**D03**	**Fan**	**M00**
ST1	**D04**	**Stirrer**	**M01**
PLC2			
Read		**Write**	
WT1	**D14**	**To PLC 1**	
FT1	**D08**	**Fan**	**D09**
CE1	**D02**	**Stirrer**	**D10**
PA1	**P03**	**To PLC 3**	
PA2	**P04**	**FC1**	**D11**
G1	**P41**		
G2	**P40**		
J	**P42**		
From PLC 1		**LC1**	**D12**
TT1	**M00**		
ST1	**M01**		
From PLC 3			
TT2	**M02**		
LT1	**M03**		
PLC3			
Read		**Write**	
TT2	**D01**	**Write From PLC 2**	
LT1	**D09**	**FC1**	**M00**
		LC1	**M01**

Table 10:connection between 3 PLCs

Memory	Function
M20	**change the mode from automatic to manual**
M15	**coil indication for modes**
M21	**manual control for G1**
M22	**manual control for G2**
M23	**manual control for J**
M24	**manual control for fan**
M25	**manual control for stirrer**
M26	**Valve1 FC1**
M27	**Valve 2LC1**

Table 11:Manual control for hmi

Memory	Function
M13	**change the mode from automatic to manual**
M06	**Manual_website_ control for G1**
M07	**Manual_website_control for G2**
M08	**Manual_website_ control for J**
M09	**Manual_website_control for fan**
M10	**Manual_website_ control for stirrer**
M11	**Manual_website_control for Valve_1(FC1)**
M12	**Manual_website_control for Valve_2(LC1)**

Table 12:Manual control from website

4.4 Process

4.4.1 Objective:

Take samples at different temperatures (45,35)

4.4.2 Challenge

The system doesn't have an output source, so the valve used to take the output

4.4.3 Overview of the Process Steps

The process starts with the activation of the heater to raise the temperature. Once the temperature reaches 45 degrees Celsius, water transfer begins from the large tank to the small tank. After reaching a specified volume in the small tank, a sample is collected, and the system enters a cooling phase. Subsequently, a second sample is collected when the temperature decreases to 35 degrees Celsius.

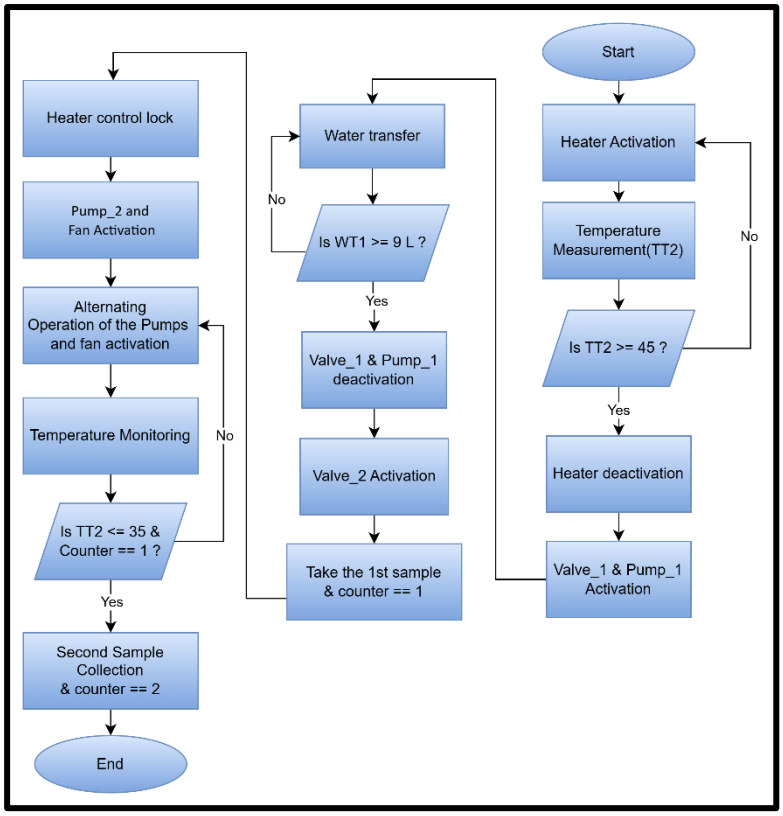


Figure 14: Flow chart of the Process

4.4.4 Detailed Description

- **Initialization:**

  The process begins with the activation of the heater.

- **Temperature Measurement:**

  The Temperature Sensor 2 monitors the temperature of the water until it reaches 45 degrees Celsius.

- **Heater Control:**

  Once the temperature reaches 45 degrees Celsius, the heater is deactivated

- **Valve 1 Activation:**

  Valve 1 (FC1) is opened.

- **Pump 1 Activation:**

  Pump 1 is turned on to transfer water from the first tank (Large tank) to the second tank (Small tank).

- **Water Transfer:**

Water is transferred until the load cell sensor in the small tank detects a volume corresponding to 9 liters.

- **Pump 1 and Valve 1 Deactivation:**

  Pump 1 and Valve 1 are both turned off.

- **Valve 2 Activation:**

  Valve 2 is opened for 3.5 seconds.

- **Sampling:**

  Counter Up (CTU1) counts one count, indicating the first sample has been taken.

- **Heater Control Lock:**

  A condition prevents the heater from being activated again, initiating the cooling phase.

- **Pump 2 and Fan Activation:**

  Pump 2 and the fan are activated after the first sample is taken.

- **Alternating Operation of the Pumps:**

  Pump 1 operates for 3 seconds, followed by Pump 2 for 4 seconds in an alternating manner.

- **Temperature Monitoring:**

  The temperature is continuously monitored.

- **Second Sample Collection:**

  When the temperature reaches 35 degrees Celsius, preparations for the second sample collection are initiated, and then take the second sample

- **End of Process:**

  Once the second sample is collected and the counter reaches 2, the process concludes. This indicates that the required samples have been successfully obtained, and there are no further actions required within the defined scope of the process.

4.5 Simulation

After finishing the ladder diagrams for the 3 PLCs, a simulation was done on XG5000 to ensure the system is working properly. After the simulation, the ladder diagram was modified a bit to reach the best performance

The video in the attached QR shows the simulation process using the updated ladder diagrams.

5. SCADA

SCADA, which stands for Supervisory Control and Data Acquisition, is a comprehensive control system architecture used in various industries for monitoring, controlling, and managing complex processes and facilities. It provides a centralized platform that enables real-time data acquisition, visualization, and control of industrial processes.

5.1 Key components and features of SCADA system

- **Supervisory Control: SCADA allows operators and supervisors to monitor and control industrial processes remotely from a central location. This supervision involves overseeing the performance of machinery, processes, and other relevant parameters.**

- **Data Acquisition: SCADA systems collect data from sensors, instruments, and devices distributed throughout the industrial environment. This data includes information on variables such as temperature, pressure, flow rate, and more.**

- **Human-Machine Interface (HMI): SCADA systems typically include an HMI that provides a graphical representation of the industrial processes. The HMI allows operators to interact with the system, view real-time data, and respond to alarms or events.**

- **Control Logic: SCADA systems implement control logic to regulate and automate processes based on the acquired data. This ensures that industrial systems operate efficiently and within specified parameters.**

- **Communication: SCADA systems use various communication protocols to establish connections between the central system and remote devices or field equipment. This enables the exchange of data and control commands.**

- **Alarming and Event Logging: SCADA systems monitor for abnormal conditions and trigger alarms when predefined thresholds are exceeded. They**

also maintain event logs for recording important occurrences and system events.

- Security: Given the critical nature of industrial processes, SCADA systems incorporate security measures to protect against unauthorized access, cyber threats, and ensure the integrity of the data and control commands.

5.2 XP-Builder

XP-Builder is software that allows you to create and manage projects for machine control devices. You can use XP-Builder to create projects for the XGT Panel. XP-Builder includes multiple features that allow you to design and edit projects conveniently, such as:

- Customizable toolbars and hotkeys
- Customizable tool, project, and editing panes
- Functions to import and export common data
- Tabs for viewing multiple screens easily
- Previews of project screens
- Customizable image and object libraries
- Scripts and advanced functions, such as alarms, and logs.
- Support for multiple languages

5.3 Building the model

The Scada model was built in XP-builder software as it is an open-source software and the next screens explain the steps of building this model:

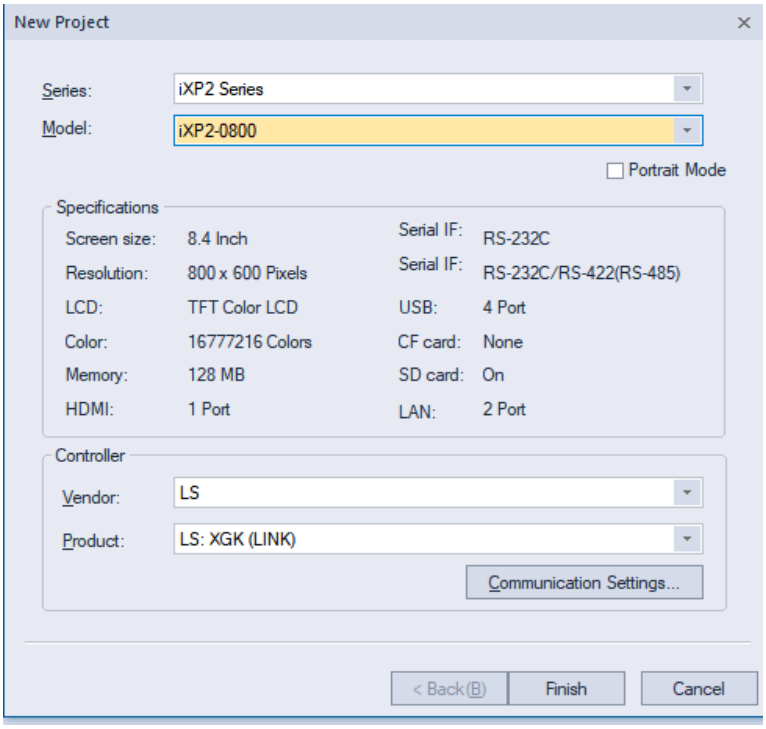


Figure 15:initialization of building Scada model

Home screen:

This screen contains the buttons that allow us to navigate to the next screens which is:

1-Automatic control 2-Manual control 3-Alarms

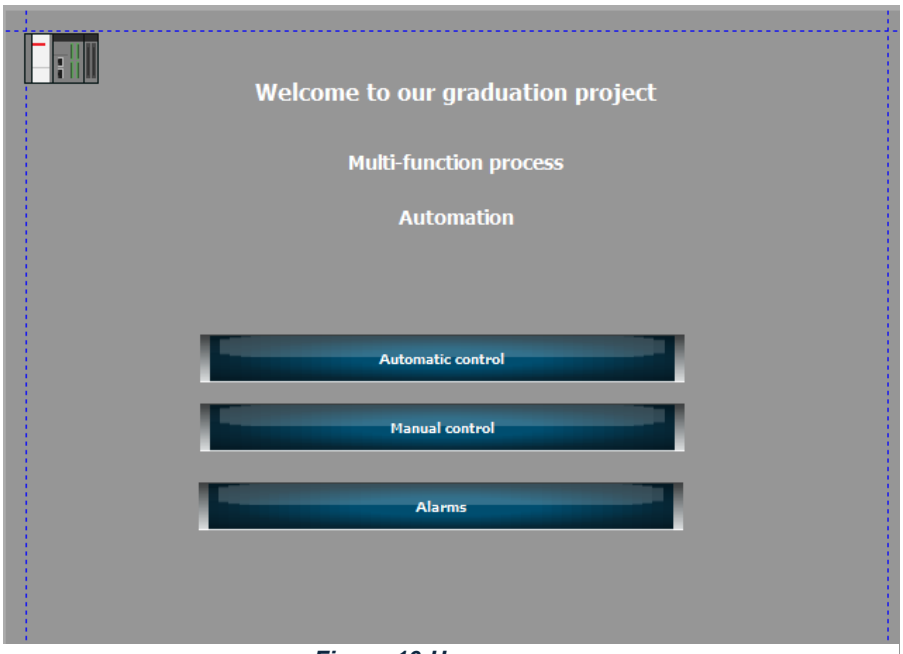


Figure 16:Home screen

Automatic Control Screen

This screen displays all system sensors and actuators, and it is accessed when the automatic process is applied

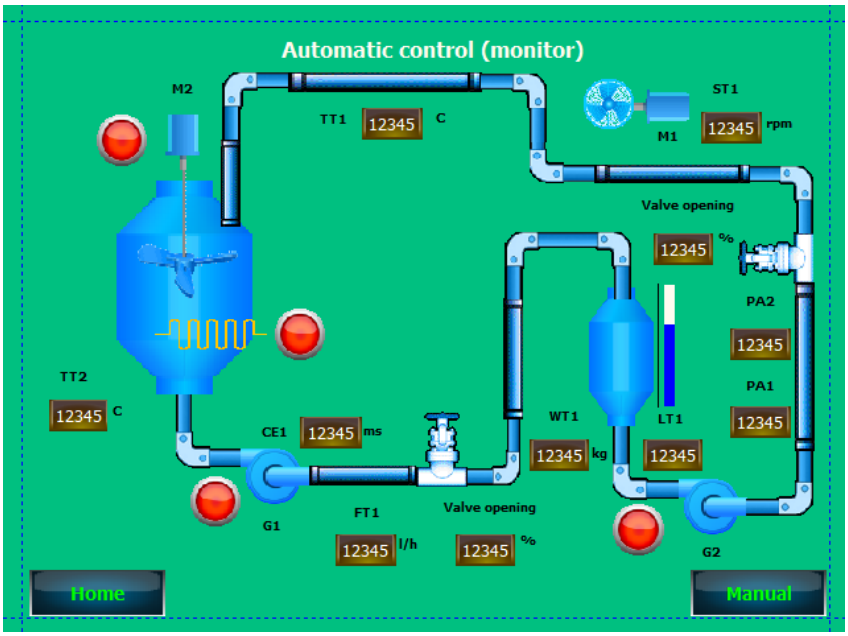


Figure 17: Automatic Screen

Manual Control Screen

This screen displays all system sensors and actuators. You access this screen when manual control of the system is necessary. To activate this mode, press the 'Manual Control' button at the bottom screen, thereby switching from automatic control to manual control. This option is designed to facilitate system testing during failures or maintenance needs. In manual control mode, you have the ability to manually adjust and control:

- **Pump 1**
- **Pump 2**
- **Valve 1**
- **valve 2**
- **Heater**
- **Stirrer**
- **Fan**

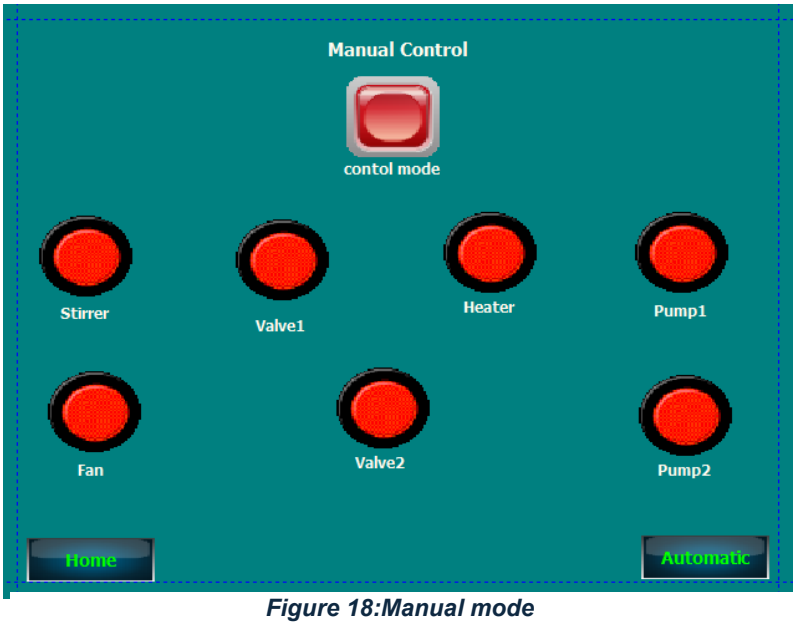


Figure 18:Manual mode

Alarm Screen

We create alarms to safeguard our system from failures and enhance its performance by tracking the frequency of issues. After designing the alarm screen, a table appears at the bottom screen, comprising:

1. **Occurrence Time**
   - **Definition: The occurrence time signifies when a specific alarm condition was initially detected or occurred within the system.**
   - **Usage: It provides a timestamp for when the alarm event was first identified, aiding in chronological analysis of system events.**
2. **Message**
   - **Definition: The message in the alarm table typically contains a brief description or information about the specific alarm condition or event.**
   - **Usage: It helps operators and users understand the nature of the alarm, facilitating quick and informed decision-making.**
3. **Group**
   - **Definition: The group in the alarm table categorizes or groups related alarms together based on common characteristics or causes.**
   - **Usage: Grouping alarms helps in organizing and prioritizing responses, especially when dealing with multiple alarms simultaneously.**
4. **Frequency**
   - **Definition: The frequency in the alarm table indicates how often a particular alarm condition has occurred over a specific period.**
   - **Usage: It provides insights into the recurring nature of specific issues, aiding in preventive maintenance and system optimization.**

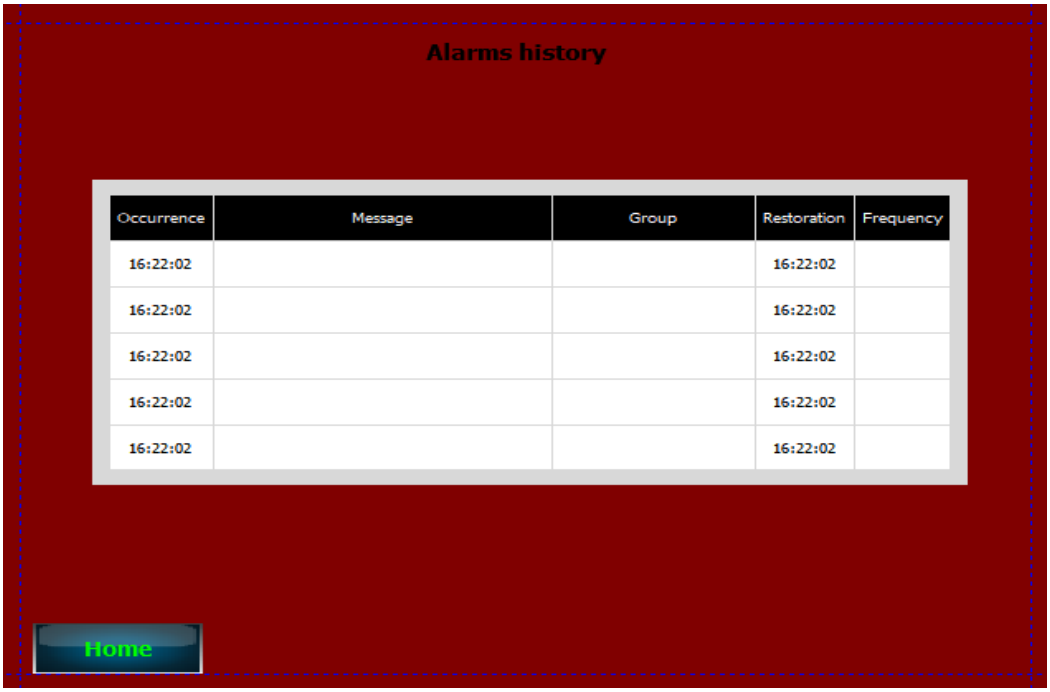


Figure 19:Alarm screen

Memory	Function
M20	**change the mode from automatic to manual**
M15	**coil indication for modes**
M21	**manual control for G1**
M22	**manual control for G2**
M23	**manual control for J**
M24	**manual control for fan**
M25	**manual control for stirrer**
M26	**Valve1 FC1**
M27	**Valve 2LC1**

Table 13:Tags between PLC and XP-Builder

5.4 Test simulation

6. Communication

we would remember the challenges that face us for make communication among the plcs with each other and form another side, the system transfers their data to cloud through the used microcontroller for building internet connect with cloud is called “ESP32”, and then we would move to the accurate hardware and connection and diagram for make this communication.

6.1 Challenges

- Current Mirror Issue
- No Synchronoss
- No Mastering

6.1.1 Current Mirror

1. Introduction

For the PLC and the IIoT to be used simultaneously the output current signal of all the sensors should be passed to both the PLC and the controller of the IIoT. Hence, there need to be two copies of each signal. The current mirroring circuit is used to copy the flow of current in one active device and control the flow of current in another device by maintaining the output current stable instead of loading. Theoretically, a perfect current mirror is an inverting current amplifier. The main function of this amplifier is to reverse the direction of the flow of current. The main function of the current mirror is to provide active loads as well as bias currents to circuits and is also used to form a more practical current source.

Several methods are available to implement a current mirror circuit. Two methods were tried to implement the circuit

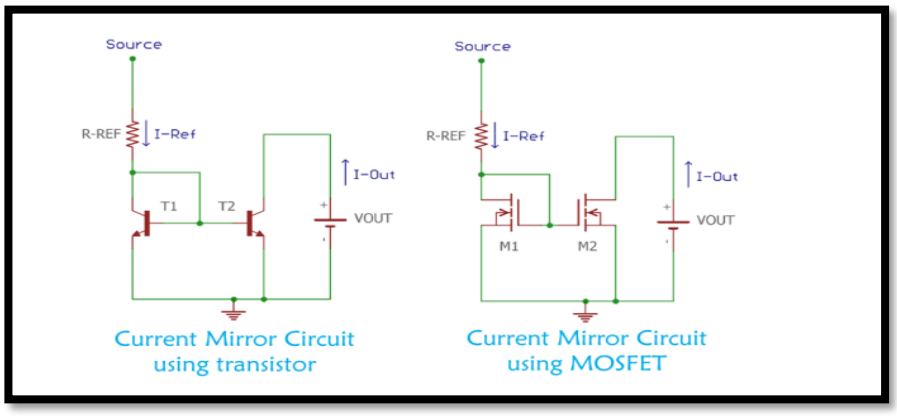


Figure 20:Current Mirror

2. USING MOSFETs

There are 4 MOSFETs in total in the circuit, two P-type and two N-type having similar properties. Here's a brief explanation of how it works:

In a basic MOSFET current mirror circuit, the input current is applied to the gate of a MOSFET, called the input or reference MOSFET. The drain of the input MOSFET is connected to the gate of a second MOSFET called the output MOSFET. The drain of the output MOSFET is connected to a current source or load resistor.

When a voltage is applied to the gate of the reference MOSFET, it establishes a current flowing through the MOSFET from the drain to the source. This current is mirrored in the output MOSFET and flows through the drain and source of the output MOSFET. The ratio of the two currents is determined by the size (W/L ratio) of the two MOSFETs and their operating conditions.

By adjusting the size and bias of the MOSFETs, the current mirror can be made to operate in a variety of configurations, such as cascade, folded cascade, or improved output impedance.

The simulation is done to check the output and the error for both 4 and 20 mA input and the results are the following:

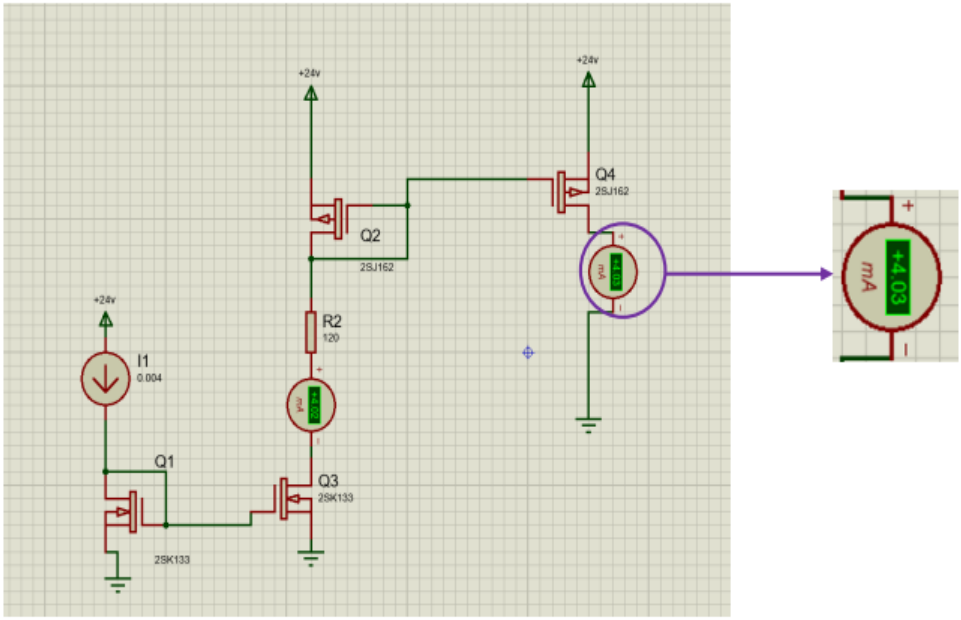


Figure 21:Current Mirror Connections 1

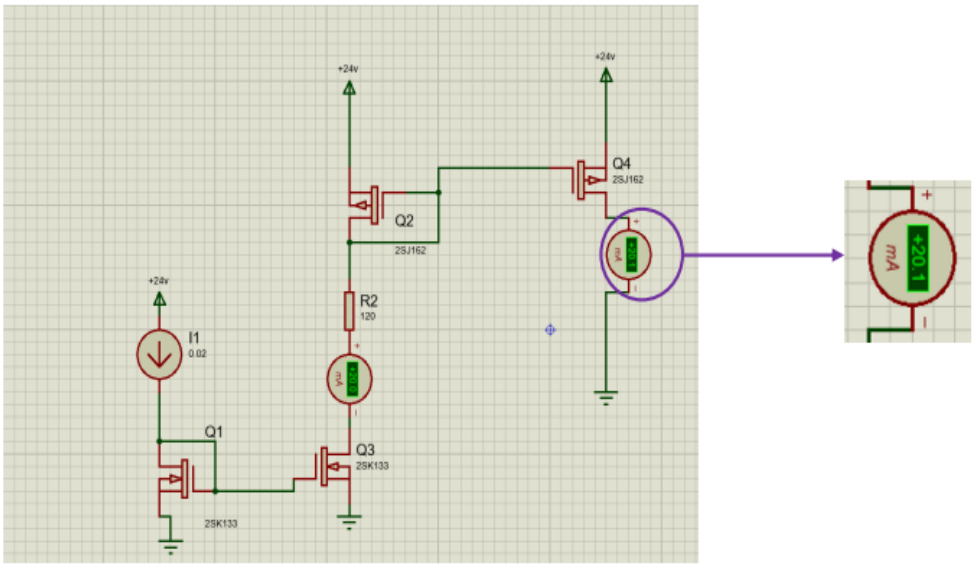


Figure 22:Current Mirror Connection 2

3. Issues Encountered

The shortage in the P-type MOSFETs in the market.

Wide variety in prices.

Hard to find P-type and N-type MOSFETs with similar characteristics.

The Previous issues made the signals that would esp32 receive them be noisy and distorted and thus the plcs and then this method was not very efficient in communication process.

6.1.2 No Synchronous & No Mastering

1. Introduction

The current mirror circuit is used to make the Esp32 take the signals of sensors without there is mastering or controlling by PLCs. In this case, the esp32 is used only for transferring the data from the system to the cloud without taking action from the cloud to control and manage the system.

While the Plc receives data and takes action regardless of whether the manager wants to take action as an alarm action or not, then the problems of synchronization and mastering must be taken into consideration for accurate communication.

Brainstorming

Now we find the current mirror and making esp32 is separated of the plcs in transferring data would make the communication is inefficient, then we thought that

to solve these problems we make master control in all processes from first the communication between PLCs getting to communication with the Cloud.

the solution that we found out that would make ESP32 Master, PLCs Slaves would be explained in detail in the part Esp32 Master, but before speaking about the Mastering of ESP32, let us speak more about the types of available protocols that plcs have and what would be the best types of protocols for applying this idea.

6.2 PLC Protocols

- **Ethernet/IP: An advanced version of standard Ethernet, developed by Rockwell Automation.**
- **Profibus: A protocol owned by Siemens Automation.**
- **Modbus: A protocol used for transmitting information over serial lines [Modbus RTU] or Ethernet [Modbus Tcp], developed by Mod icon (now Schneider Electric).**

1- Ethernet/IP

-In This Part from Topic, we thought to make the plc Is Master and synchroniser through Communicating with cloud directly without needing to any helper as esp32.

XGB Fast Enet I/F module supports open Enet. It provides network configuration that connects LSIS and other company PLC, PC on network.

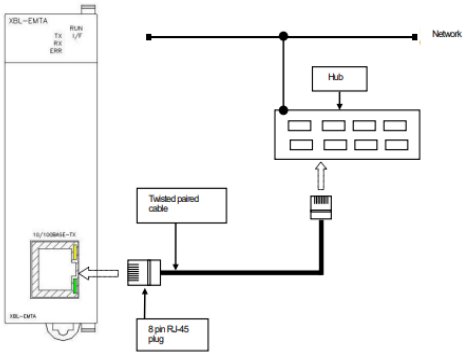


Figure 23:Ethernet/Ip

Examples of System Configuration:

-Combination network configuration

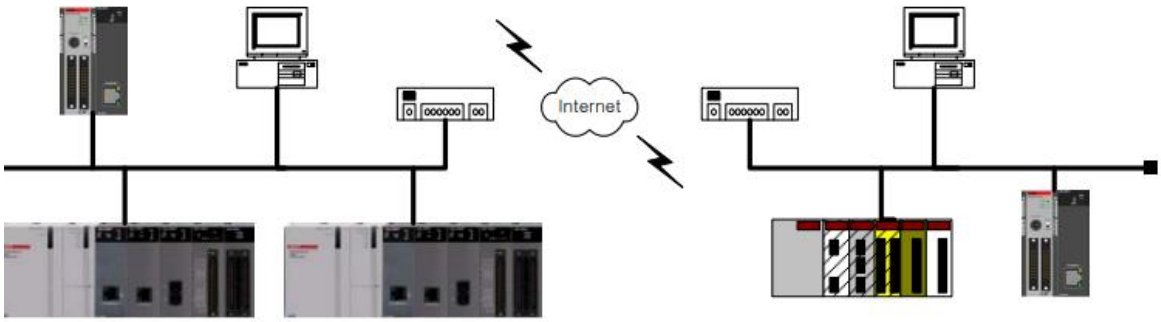


Figure 24:Combination Network configuration

XGB Fast Enet I/F module provides system configuration by using main communication, Modbus TCP/IP, user define frame, HS link communication connecting LSIS PLC with other LSIS PLC, PC on network.

- Network configuration using XGB

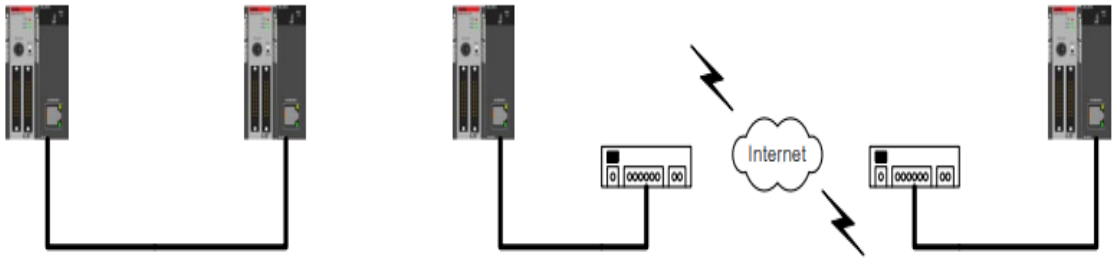


Figure 25:Network Configuration XGB Driver

Communication between XGB Fast Enet I/F modules is available to perform 1:1 communication by using cross cable or 1: N communication by connecting network. It provides data sending/receiving by using the dedicated service, Modbus TCP/IP, user define frame and HS link communication.

And there are many examples that certain the high ability of plc for communicating by Ethernet Protocol, but the encountered issue is that LSIS cannot handle the signals with web through http protocols, so we moved to the Modbus protocol that is the most famous in Automation World to be the best solution for our mission.

2- Modbus Protocol

- **Introduction**

  Modbus protocol is specified open protocol used between client-server, which executes reading/writing data according to function code. Communication between devices that use Modbus protocol uses Client-server function in which only one client processes the data.

- **Kind of Modbus protocol**

  There are two communication modes of Modbus, ASCII and RTU.

Characteristic		ASCII mode	RTU mode
Coding method		ASCII code	8 bit binary code
No. of data per one character	Start bit	1	1
	Data bit	7	8
	Parity bit	Even,Odd,None	Even,Odd,None
	Stop bit	1 or 2	1 or 2
Error check		LRC(Longitudinal Redundancy Check)	CRC (Cyclical Redundancy Check)
Start of frame		Colon (:)	3.5 Character no response time

Table 14:Types of Modbus

- **Data and expression of address**

  - **To express data and address of Modbus protocol, the characteristic is as follows. (1) It used hexadecimal as basic form. (2) In the ASCII mode, Hex data is converted into ASCII code. (3) RTU mode uses Hex data. (4) Each function code has the following meaning.**

Code(Hex)	Purpose	Used area	address	Max. response data
01	Read Coil Status	Bit output	0XXXX	2000bit
02	Read Input Status	Bit input	1XXXX	2000bit
03	Read Holding Registers	Word output	4XXXX	125word
04	Read Input Registers	Word input	3XXXX	125word
05	Force Single Coil	Bit output	0XXXX	1bit
06	Preset Single Register	Word output	4XXXX	1word
0F	Force Multiple Coils	Bit output	0XXXX	1968bit
10	Preset Multiple Registers	Word output	4XXXX	120word

Table 15:expression of address

and Because of Modbus RTU is often considered better than Modbus ASCII for several reasons:

Efficiency: Modbus RTU uses binary encoding, which allows for more compact and efficient data transmission compared to Modbus ASCII. The binary nature of Modbus RTU reduces the transmission overhead, enabling faster and more efficient communication.

Transmission Speed: Due to its binary encoding and simpler frame structure, Modbus RTU can achieve higher transmission speeds compared to Modbus ASCII. This is particularly beneficial in applications that require fast and time-sensitive communication.

Compatibility: Modbus RTU enjoys broader support and compatibility with devices and software applications compared to Modbus ASCII. Many industrial devices and communication modules are specifically designed to work with Modbus RTU, making it a more practical choice in most industrial automation environments.

Error Checking: Both Modbus RTU and Modbus ASCII include error checking mechanisms, such as CRC (Cyclical Redundancy Check). However, the binary nature of Modbus RTU allows for more robust error checking, resulting in better data integrity and error detection capabilities.

Noise Immunity: Modbus RTU, being a binary-based protocol, is more resistant to noise and interference compared to Modbus ASCII. The binary nature of Modbus RTU signals allows for better noise immunity, making it suitable for industrial environments with high levels of electrical noise.

Implementation Simplicity: Modbus RTU has a simpler frame structure compared to Modbus ASCII, which includes additional character framing. This simplicity makes it easier to implement and troubleshoot Modbus RTU-based systems.

Performance: Due to its efficiency, higher transmission speed, and better error checking, Modbus RTU generally offers better overall performance compared to Modbus ASCII.

So, we used the Modbus RTU for communicating among the plcs with each other and esp32 with them. but how we would make this idea.

We would make the esp32 master Modbus RTU while the PLCS would be the slaves Modbus RTU, then the esp32 would have ability to request from any PLC and then PLC response to it ,in this way we can make synchronization and mastering in

transferring data without any noisy or until distortion data and from the another side , esp32 would be bridge between the PLCS and Cloud ,this diagram is mentioned below is the description of the solution.

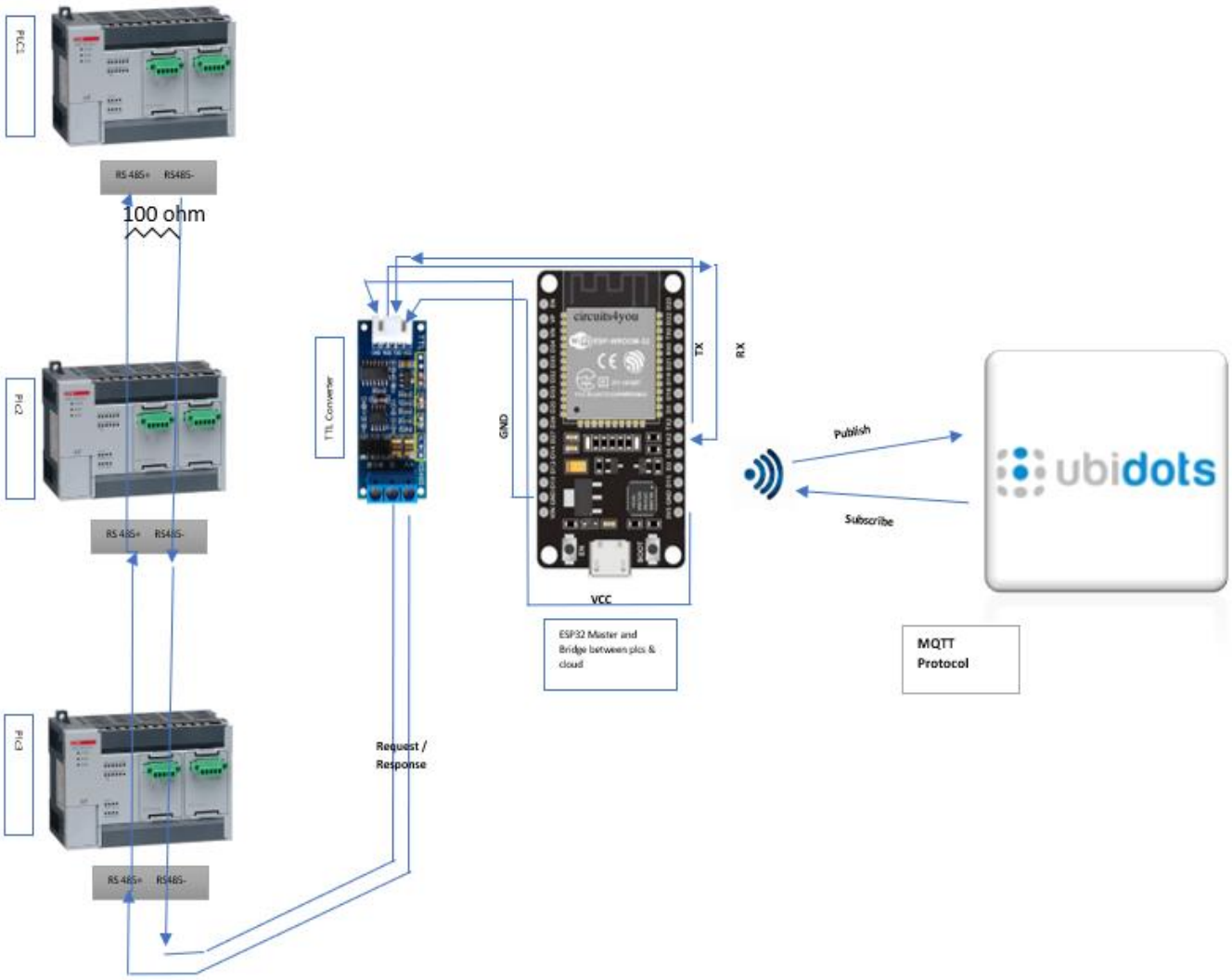


Figure 26:Communication Diagram

Now we will explain how to make PLCs as Slaves MODBUS RTU, ESP32 as Master Modbus RTU.

3- Slave Modbus RTU

First, Modbus RTU can communicate over several types of physical media, two of which are built-in the CPU module of the PLC model used in the project: RS232 and RS485. Modbus used to run on RS232 communication predominantly in the past but nowadays it is more commonly used with RS485.

1- RS485 vs RS232

RS485 can be considered the more advanced version of RS232 as it solves most of the problems encountered using RS232.

RS232 / RS485 Comparison		
P.O.C	***RS232***	***RS485***
Speed	**Slower (20 kb/s)**	**Faster (10 Mb/s)**
Distance	**50 ft.**	**4000 ft.**
No. of devices	**1**	**32**
Noise	**More susceptible (ground reference)**	**Less susceptible**
Complexity	**Simple**	**Complex**
Mode of operation	**Simplex or full duplex**	**Simplex or half/full duplex**

Table 16:comparsion between RS232 and RS485

Due to the previous reasons and the fact that the project has three PLCs running simultaneously, RS485 seemed like the better solution for the process to run fast and efficiently.

2- Modbus memory mapping

Modbus has four basic memory areas depending on the type and direction of data, they can be divided into the following:

Modbus Memory Areas	
Start Address	**Description**
1xxxxx	**Bit read area**
0xxxxx	**Bit write area**
3xxxxx	**Word read area**
4xxxxx	**Word write area**

Table 17:Modbus memory areas

3- Slave Modbus RTU

The Modbus settings need to be adjusted in each PLC to be slave through network configuration, where the PLC registers are mapped into Modbus addresses.

Each PLC also needs to have a unique station number and the baud rate need to be specified. The following are the Modbus settings of each PLC:

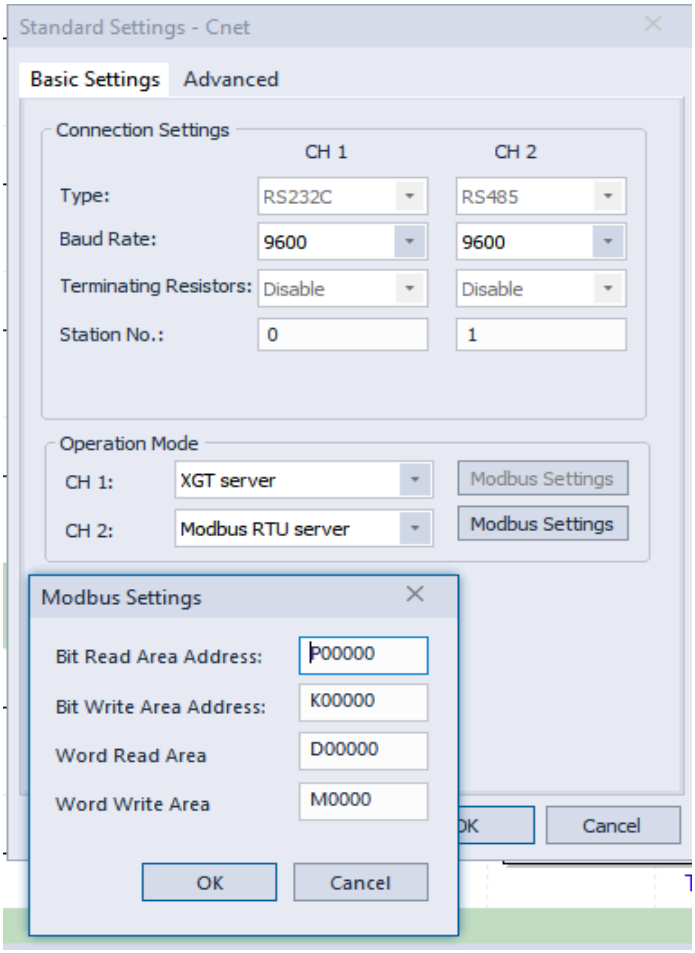


Figure 28:PLC1 Modbus Settings

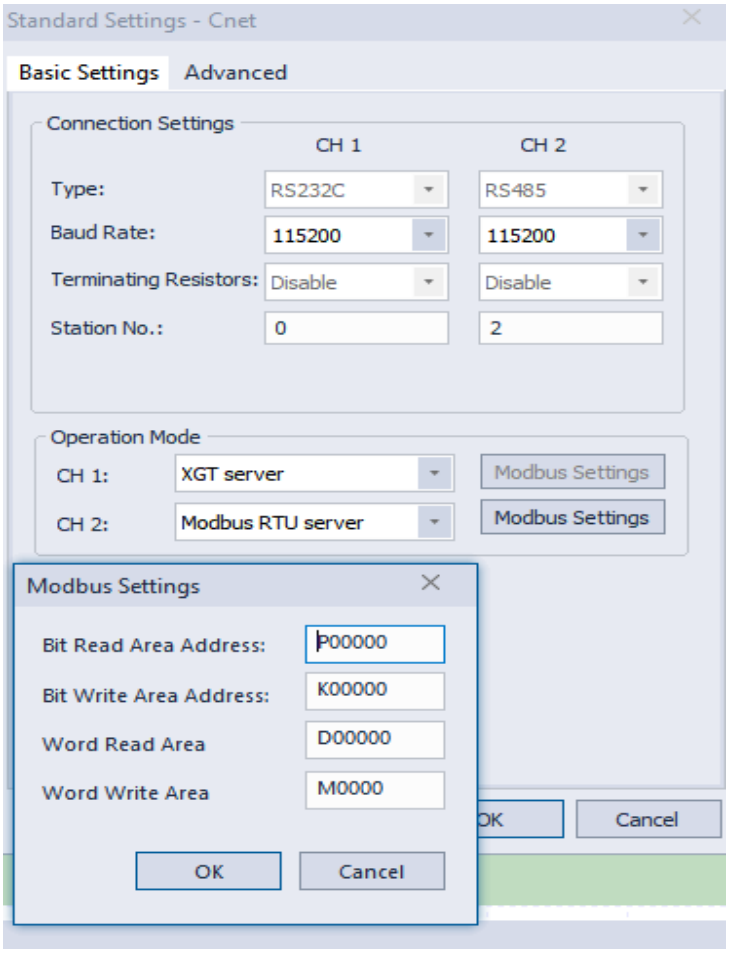


Figure 29:PLC2 Modbus Settings

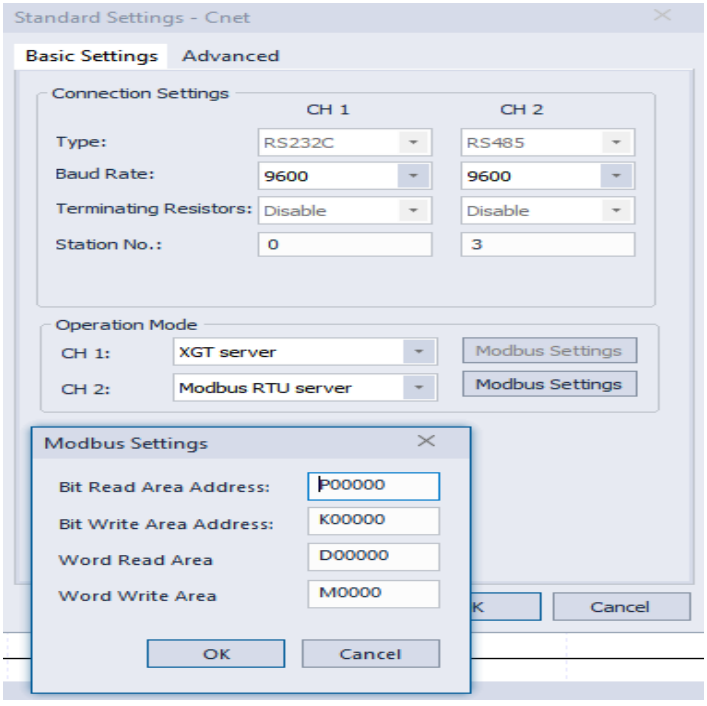


Figure 27:PLC3 Modbus Settings

4- Master Modbus RTU

To make the esp32 master Modbus RTU that can handle data through serial communication RS 485, we needed to converter called “RS 485 TTL CONVERTER” That achieves this mission distinctively, now we would talk about TTL CONVERTER in some detail.

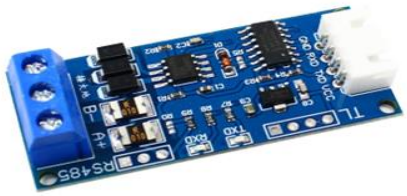


Figure 30:RS 485 TTL

- Introduction

This is a module that allows the TTL interface of the microcontroller to be transferred to the RS485 module. It is generally used for industrial automation.

This module adds lightning protection design and anti-jamming design. When we use it in the field and carry out long-distance transmission, we can Connect the GND end of the module to the ground so that lightning protection and anti-interference can be achieved.

It uses a standard 2.54 mm pitch design. If you want to make the second development will be more convenient. It has 120-ohm matching resistance, and shorted RO can, it is recommended that users in the long-distance transmission shorted.

It supports communication between multiple microcontrollers, allowing up to 128 devices on the bus. It can be hot-swappable, to prevent signal interference, it carries out a large area of copper, and the effect is very good.

- Features

1. 3.3V perfectly compatible with 5.0V power supply
2. Absolute using imported chips, industrial design, anti-interference ability, while using more effective mine design can be used in the industrial field and harsh field environments, the working temperature of -40 °C to + 85 °C transmission distance up to 1 kilometer (made with 850 meters of cable F1.5 tests recommended in the 800 meters, more than 800 m, please add repeaters).
3. Transmission distance up to kilometers (with 850 meters of 2* 1.5 cable to do the test, it is recommended to use in the 800 meters, more than 800 meters, please add repeaters)
4. Semi-hole process design, a thickness of 0.8mm, makes it easy to use as a combo board, it can also be welded for terminal use.
5. Has RXD, and TXD signal lights to observe send and receive status.

6. **The module is fully considered mine design, long-distance transmission signal 485 in the wild, the module "Connected to the ground" terminal is connected to the earth, can play a very good anti-jamming and anti-lightning effect, more secure; it cannot connect with the ground when the indoor short-distance transmission.**
7. **Using the standard 2.54 pitch design, easy secondary development.**
8. **Has a 120-ohm termination resistor, shorted Roto enables the termination resistor.**
9. **Support multi-machine communication, allowing access to up to 128 devices on the bus.**
10. **The module can be hot-swapped, and the signal does not appear dead phenomenon.**
11. **A large area of copper to prevent interference.**

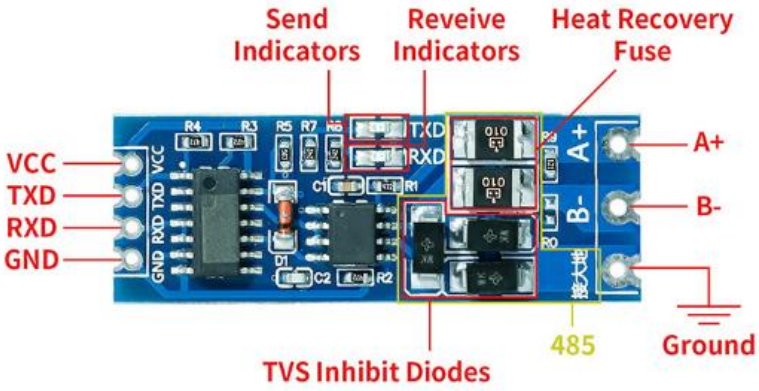


Figure 31:TTL CONVERTER DETAILED

Finally, we need to implement code on Arduino ide to make the esp32 master Modbus RTU which is achieved by a library called “<Modbus Master. H>”, while we need to make the esp32 send data to the cloud “ubidots” that would be explained in detail in “chapter seven” and this is achieved by library is called "UbidotsEsp32Mqtt.h" that handle with MQTT protocol is very efficient in internet of things.

6.3 Testing

7. IIOT

7.1 Platform

IoT and Cloud tools to drive digital transformation, Merge the physical and digital worlds into deployable, end-user ready Applications that deliver insights and solve problems.

Figure 32:ubidots platform

7.2 Device friendly API and SDKs

Connect hardware to Ubidots cloud easily with more than 200 user-proven libraries, SDKs, and tutorials to guide your integration over HTTP, MQTT, TCP,UDP, or by parsing custom/industrial protocols.

Whether onboarding 1 or 1,000 devices, it takes the same amount of effort with Ubidots Device Types. Replicate creating the new device in Ubidots automatically configuring variables, device properties, and appearance each time a new piece of hardware is detected.

Figure 33:connection hardware to ubidots

7.3 Synthetic Variables

Transform raw data into insights with Synthetic Variables that compute complex math formulas and statistical expressions.

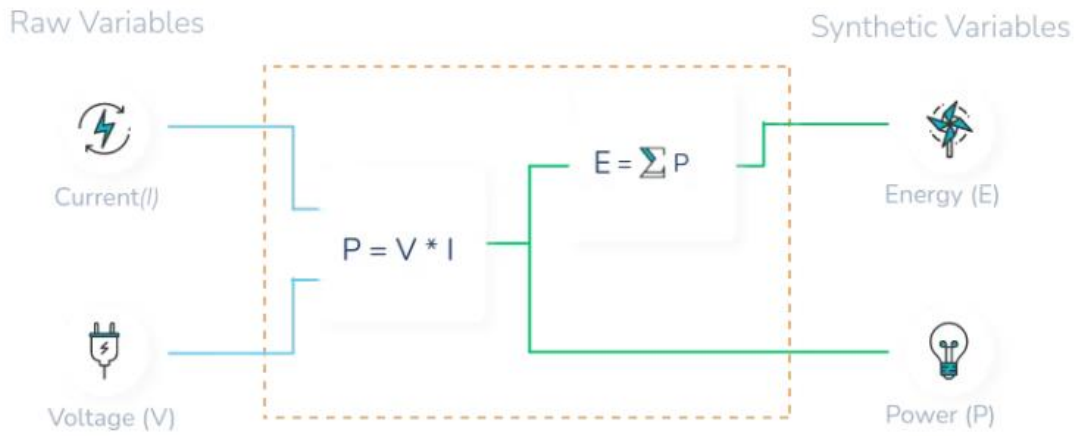


Figure 34:synthetic variables

7.4 Two-year time-series backend and storage

Whether your data is needed every day or every second, Ubidots time-series infrastructure is optimized to receive, compute, and return millions of data points each second across the globe and, with 2 years rolling data retention standard, Ubidots gives business managers and operation analysts a place to store and mine data for added insights and analytic overlays like anomaly detection or predictive maintenance.

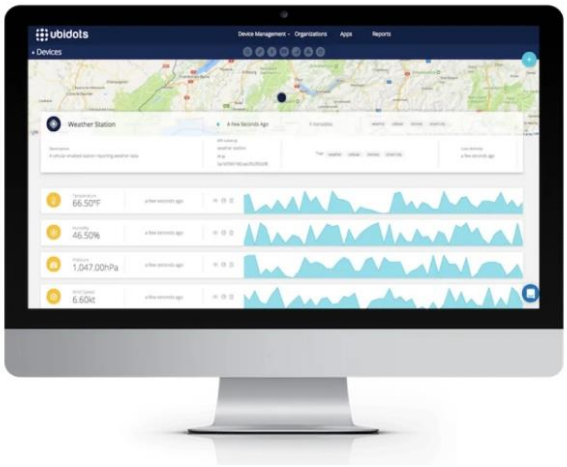


Figure 35:time -series backend and storage

7.5 Live Dashboard

using Ubidots point-and-click application development tools, create real-time dashboards to analyze data and control devices.

Visualize data with Ubidots stock graphs, charts, tables, indicators, maps, metrics, and control widgets — or even develop your own using the HTML Canvas and your own code.

SHARE YOUR DATA THROUGH PUBLIC LINKS OR BY EMBEDDING DASHBOARDS OR WIDGETS INTO PRIVATE WEB AND MOBILE APPLICATIONS

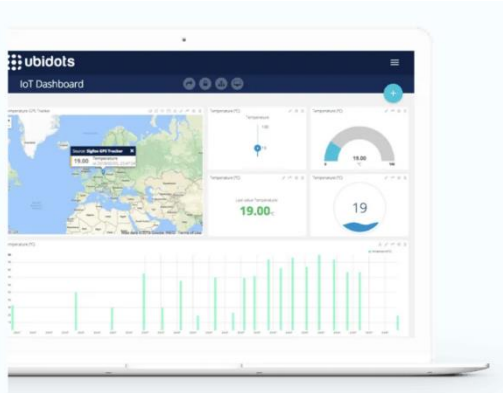


Figure 36:Live dashboard

7.6 Ubidots IoT

Ubidots gives decision makers the exact information needed for critical decisions in real-time, regardless of where the users are. Whether in the office, on the production floor, at home, or on the road, Ubidots keeps the users informed and empowered to consistently make data-driven decisions for improving production.

Today, data at your fingertips is the norm and Ubidots agrees. Access your applications from any Android device and stay informed when on the go.

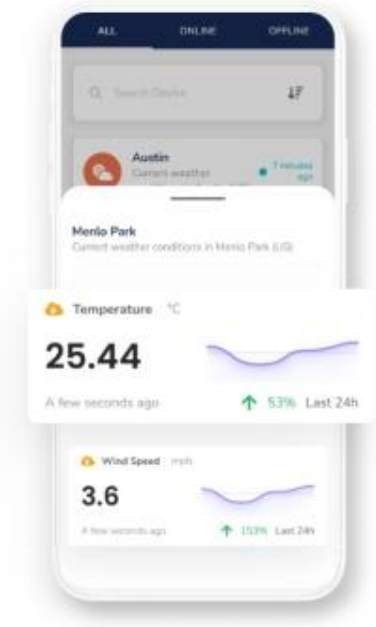


Figure 37:Ubidots iot

7.7 The Ubidots Advantage

1. Global Reliability

Scale from one device to 1,000s with confidence in Ubidots 99.9% uptime SLA made possible by automated failover and replication across IBMs global data centers

2. **Usability Comes First**

Intuitive interfaces and cloud development tools that make Ubidots a pleasure for developers, management, and operators to work with.

3. **Low Barriers to Entry**

Ubidots tools and pricing are designed from the bottom-up to help your business scale its cloud solutions from pilot projects to turn-key solutions.

7.8 Pricing

Flexible pricing to launch and scale your IoT project

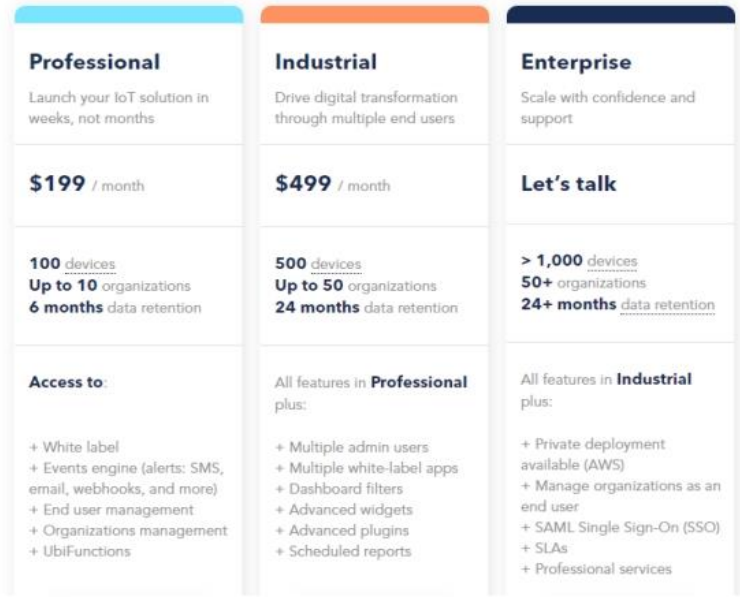


Figure 38:pricing of ubidots

1. Plan Capacity

Professional	Industrial	Enterprise
$199 / month	$499 / month	Customized capacity.
SIGN UP	SIGN UP	CONTACT US
100 Devices $20 per 10 extra (≈$2/device). Up to 1,000 devices	**500** Devices $50 per 50 extra (≈$1/device). Up to 1,000 devices	**+1,000** Devices Starting at $300 per 1,000 extra (≈$0.3/device)
15 M Dots in, stored **2 years** $5 per million extra	**50 M** Dots in, stored **2 years** $5 per million extra	**100 M** Dots in, stored **2 years** $4 per million extra
15 M Dots out $0.1 per million extra	**50 M** Dots out $0.1 per million extra	**100 M** Dots out $0.1 per million extra
6 Dots per second Up to one burst of 3x capacity every hour	**20** Dots per second Up to one burst of 3x capacity every hour	**1,000** End users $300 per 1,000 extra (≈$0.3/user)
1,000 Events executions $10 per million extra	**1,000** Events executions $10 per million extra	**99.9%** SLA
100 Email and Telegram alerts $2 per thousand extra	**100** Email and Telegram alerts $2 per thousand extra	**24h** Guaranteed response time
10 SMS and Voice call alerts Pricing based on **receiving country**	**10** SMS and Voice call alerts Pricing based on **receiving country**	**1** Assigned Technical Account Manager
Up to 1 White-labeled app (Max. 1 App, 20 Organizations, and 20 End users)	**1+** White-labeled app $149 per extra white-labeled app $49 per extra gray-labeled app	**Enhanced security** Single sign-on Log in to Ubidots using the identity provider of your choice
Up to 20 Organizations	**Up to 100** Organizations	**Full customization** White-labeled API, API docs, and mobile app
1,000 UbiFunctions executions $5 per million extra	**1,000** UbiFunctions executions $5 per million extra	**Tailored solutions** Professional services Get purpose-built feature requests on top of the Ubidots platform
Up to 20 End users	**200** End users $25 per 50 extra	**Increased capacity** Provision the right capacity based on your expected amount of devices, variables, dots per second, events, organizations, and users
	10 Admin Users	

Figure 39:Plan capacity

7.9 QR for ubidots website

7.10 Ubidots Objects

1- Sensors used in our process

- load_cell-(wt-1)
- Flow_Sensor (FT-1)
- Diff_Pressure (LT-1)
- Taco_Meter (ST-1)
- temp_sensor-1- (TT-1)
- temp_sensor-2-(TT_2)

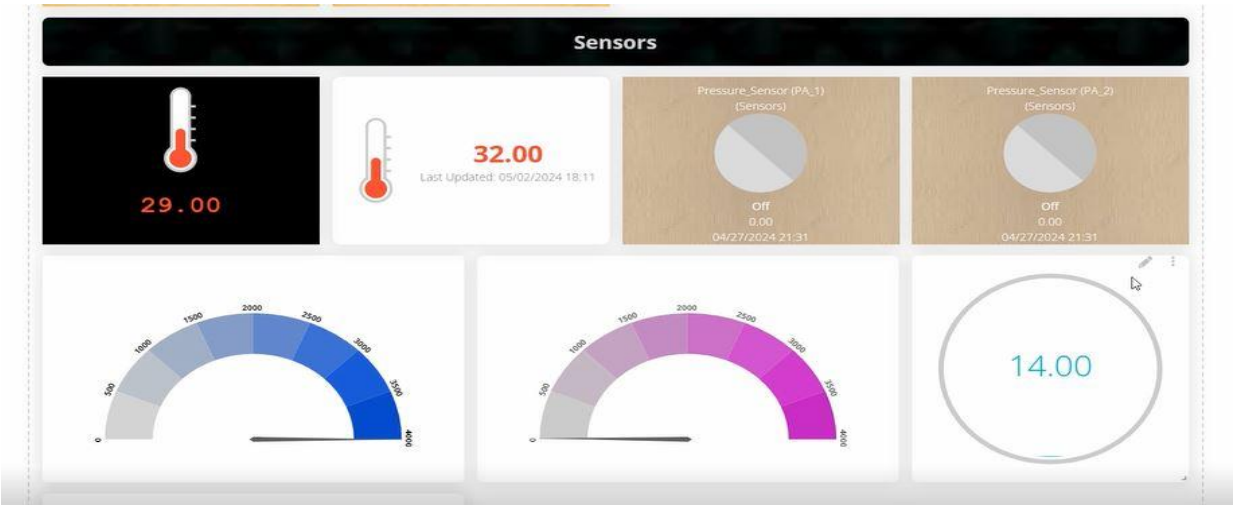


Figure 40:Sensor reading

2- Actuators used in our process

- Switch Mode (Supervisor-Switch)
- pump_1-(G1)
- Pump_2-(G2)

- **Control Valve "U" (LC-1)**
- **Control Valve "D" (FC-1)**
- **heater-(j_1)**
- **Stirrer Motor (M2)-A**
- **Fan Motor (M1)-A**

When the control mode button is pressed, the automatic process fails, and control switches to the website. This allows us to have control over any actuator as needed, enabling remote system control

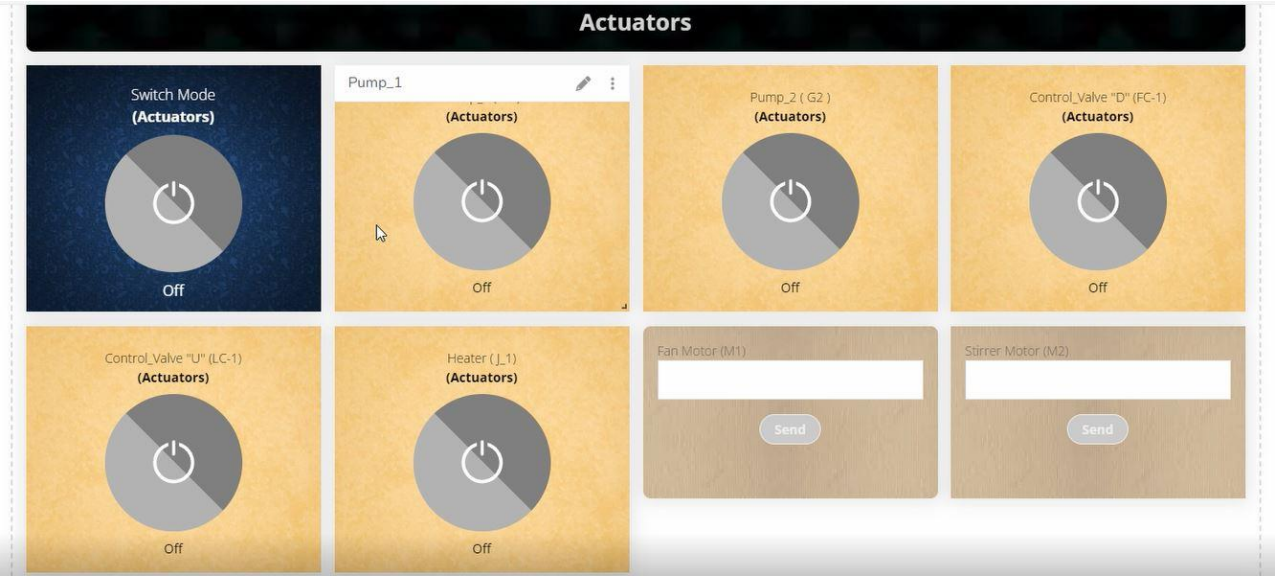


Figure 41:Controlling actuators

7.11 Analysis for Reading Temperature sensor 2

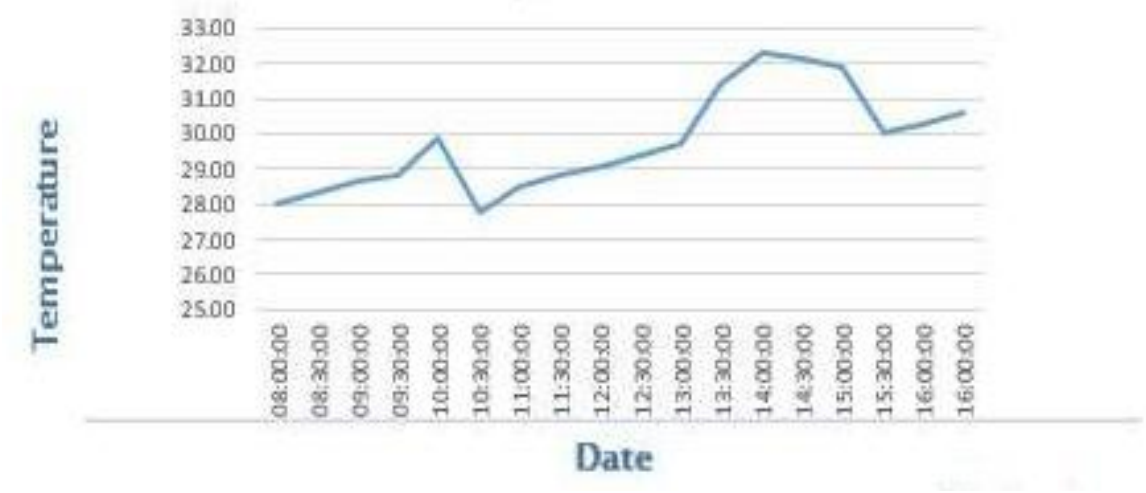


Figure 42:Analysis for temp. sensor 2

7.12 Communication between Ubidots and plc

the communication between the PLCs and upidots was performed using esp32 and TTL converter and this connection was explained in the communication chapter.

7.13 Test

8. Digital Twin (The Future)

8.1 Introduction

8.1.1 What is a Digital twin?

A digital twin is a virtual representation of an object or system designed to reflect a physical object accurately. It spans the object's lifecycle, is updated from real-time data, and uses simulation, machine learning, and reasoning to help make decisions.

8.1.2 How does a digital twin work?

We will show an example for understanding how a digital twin works, the example is a wind turbine — is outfitted with various sensors related to vital areas of functionality. These sensors produce data about different aspects of the physical object's performance, such as energy output, temperature, weather conditions, etc. The processing system receives and actively applies this information to the digital copy.

Figure 43:Wind Turbine

After being provided with the relevant data, the digital model can be utilized to conduct various simulations, analyze performance problems, and create potential enhancements. The ultimate objective is to obtain valuable knowledge that can be used to improve the original physical entity.

- **Digital twins versus simulations:**

Although simulations and digital twins both utilize digital models to replicate a system's various processes, a digital twin is actually a virtual environment, which makes it considerably richer for study. The difference between a digital twin and a simulation is largely a matter of scale: While a simulation typically studies 1 particular process, a digital twin can run any number of useful simulations to study multiple processes.

The differences don't end there. For example, simulations usually don't benefit from having real-time data. But digital twins are designed around a two-way flow of information that occurs when object sensors provide relevant data to the system processor and then happen again when insights created by the processor are shared back with the original source object.

By having better and constantly updated data related to a wide range of areas, combined with the added computing power that accompanies a virtual environment, digital twins can study more issues from far more vantage points than standard simulations can, with greater ultimate potential to improve products and processes.

8.1.3 Types of Digital Twins

Component twins or Parts twins

Component twins are the basic unit of a digital twin, the smallest example of a functioning component. Parts twins are roughly the same thing but pertain to components of slightly less importance.

Asset twins

When two or more components work together, they form what is known as an asset. Asset Twins lets you study the interaction of those components, creating a wealth of performance data that can be processed and then turned into actionable insights.

System or Unit twins

The next level of magnification involves system or unit twins, which enable you to see how different assets come together to form an entire functioning system. System twins provide visibility regarding the interaction of assets and may suggest performance enhancements.

Process twins

Process twins, the macro level of magnification, reveal how systems work together to create an entire production facility. Are those systems all synchronized to operate at peak efficiency, or will delays in one system affect others? Process twins can help determine the precise timing schemes that ultimately influence overall effectiveness.

8.1.4 History of digital twin technology

The idea of digital twin technology was first voiced in 1991, with the publication of Mirror Worlds, by David Gelernter. However, Dr. Michael Grieves (then on faculty at the University of Michigan) is credited with first applying the concept of digital twins to manufacturing in 2002 and formally announcing the digital twin software concept. Eventually, NASA's John Vickers introduced a new term— "digital twin"—in 2010.

However, the core idea of using a digital twin as a means of studying a physical object can be witnessed much earlier. It can be rightfully said that NASA pioneered the use of digital twin technology during its space exploration missions of the 1960s when each voyaging spacecraft was exactly replicated in an earthbound version that was used for study and simulation purposes by NASA personnel serving on flight crews.

8.1.5 Advantages and benefits of digital twins

Better R&D

The use of digital twins enables more effective research and design of products, with an abundance of data created about likely performance outcomes. That information can lead to insights that help companies make needed product refinements before starting production.

Greater efficiency

Even after a new product has gone into production, digital twins can help mirror and monitor production systems, to achieve and maintain peak efficiency throughout the entire manufacturing process.

Product end-of-life

Digital twins can even help manufacturers decide what to do with products that reach the end of their product lifecycle and need to receive final processing, through recycling or other measures. By using digital twins, they can determine which product materials can be harvested.

8.1.6 Applications

According to IBM Document, Digital twins are already extensively used in the following applications:

Power-generation equipment

Large engines—including jet engines, locomotive engines, and power-generation turbines—benefit tremendously from the use of digital twins, especially for helping to establish time frames for regularly needed maintenance.

Structures and their systems

Big physical structures, such as large buildings or offshore drilling platforms, can be improved through digital twins, particularly during their design. Also useful in designing the systems operating within those structures, such as HVAC systems.

Manufacturing operations

Since digital twins are meant to mirror a product's entire lifecycle, it's not surprising that digital twins have become ubiquitous in all stages of manufacturing, guiding products from design to finished product, and all steps in between.

Healthcare services

Just as products can be profiled by using digital twins, so can patients be receiving healthcare services. The same type of system of sensor-generated data can be used to track various health indicators and generate key insights.

Automotive industry

Cars represent many types of complexes and co-functioning systems, and digital twins are used extensively in auto design, both to improve vehicle performance and increase the efficiency surrounding their production.

Urban planning

Civil engineers and others involved in urban planning activities are aided significantly by the use of digital twins, which can show 3D and 4D spatial data in real-time and also incorporate augmented reality systems into built environments.

8.2 Azure Cloud Platform

Up to this point, we have understood the concept and functionality of a digital twin. Our next step is to construct our own digital twin system. In our quest for the ideal platform, we have determined that the Azure cloud platform is the superior choice due to its comprehensive suite of services, which includes Azure IoT Hub, Azure Functions, Azure Event Hubs, and Azure Digital Twins, among others. In the following sections, we will detail each of these services. For now, let's begin by exploring an overview of the Azure platform before we proceed further.

Figure 44:Azure Portal

The Azure cloud platform offers a robust collection of over 200 products and cloud services that enable you to innovate and address current challenges while preparing for the future. With Azure, you can build, run, and manage applications across multiple clouds, on-premises, and at the edge, using the tools and frameworks you prefer1. It's designed to be flexible, and open to all languages and frameworks, and it's continuously evolving to help you keep pace with

technological advancements. Azure also emphasizes security, compliance, and hybrid cloud integration, making it a trusted platform for a wide range of industries.

Now that we have a general understanding of Azure, let's move on to the specifics of each service as they relate to building our digital twin system.

8.2.1 Diagram Of Digital Twin

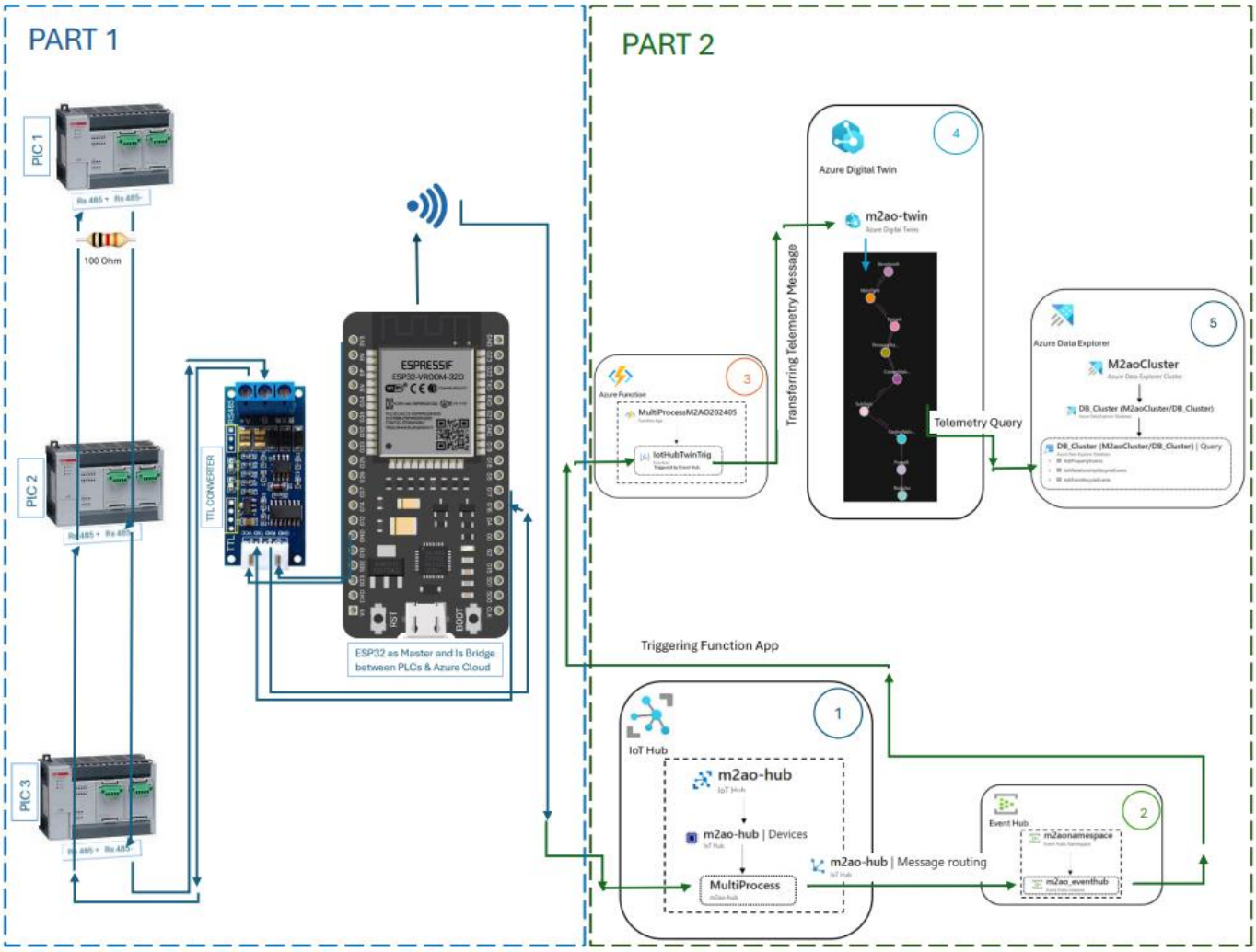


Figure 45:Diagram of Digital Twin

In the above diagram, we outline the sequential steps necessary to construct a digital twin system. The diagram is divided into two main sections: Part 1 and Part 2. Previously, in chapter Six "Communication", we provided an in-depth explanation of establishing communication between the ESP32, TTL Converter RS485, and three PLCs using the Modbus RTU Protocol. Moving forward, we will delve into the specifics of Part 2, detailing the process of creating the digital twin for our system.

The services of Azure Cloud are used for building Digital Twin almost five services:

1. Azure IoT Hub.
2. Event Hub.
3. Azure Function.
4. Azure Digital Twin.
5. Azure Data Explorer.

8.2.2 Azure Services

8.2.2.1 Azure IoT Hub

The Internet of Things (IoT) is a network of physical devices that connect to and exchange data with other devices and services over the Internet or other networks. There are currently over ten billion connected devices in the world, and more are added every year. Anything that can be embedded with the necessary sensors and software can be connected over the internet.

Figure 46:Azure IoT Hub

Azure IoT Hub is a managed service hosted in the cloud that acts as a central message hub for communication between an IoT application and its attached devices. You can connect millions of devices and their backend solutions reliably and securely. Almost any device can be connected to an IoT hub.

Several messaging patterns are supported, including device-to-cloud messages, uploading files from devices, and request-reply to methods to control your devices from the cloud. IoT Hub also supports monitoring to help you track device creation, device connections, and device failures.

IoT Hub scales to millions of simultaneously connected devices and millions of events per second to support your IoT workloads.

We can integrate IoT Hub with other Azure services to build complete, end-to-end solutions. In our case, we will integrate it with Azure Event Hub Service.

Steps for Creating IoT Hub

1- Make an Azure portal account with an "Azure for students "subscription, so you must follow these steps:

- Go to the Azure Portal and select 'Create a Microsoft account' if you don't already have one. You can use any email address for this, including Gmail.
- Once your Microsoft account is set up, sign in to the Azure Portal.
- Look for the "Azure for Students" offer, which provides you with certain Azure services free for 12 months, plus a limited amount of free credit to spend in the first 30 days.
- To verify your academic status, you will need to sign in with your school email account. This is typically provided by your university or educational institution.

2- On the Azure homepage, select the + Create a resource button.
3- From the Categories menu, select Internet of Things, and then select IoT Hub.
4- On the Basics tab, complete the fields as follows in the image, In my account, the Resource group is called “Azure_M2AO”, the IoT hub name is “m2ao-hub”, the region is EAST US.

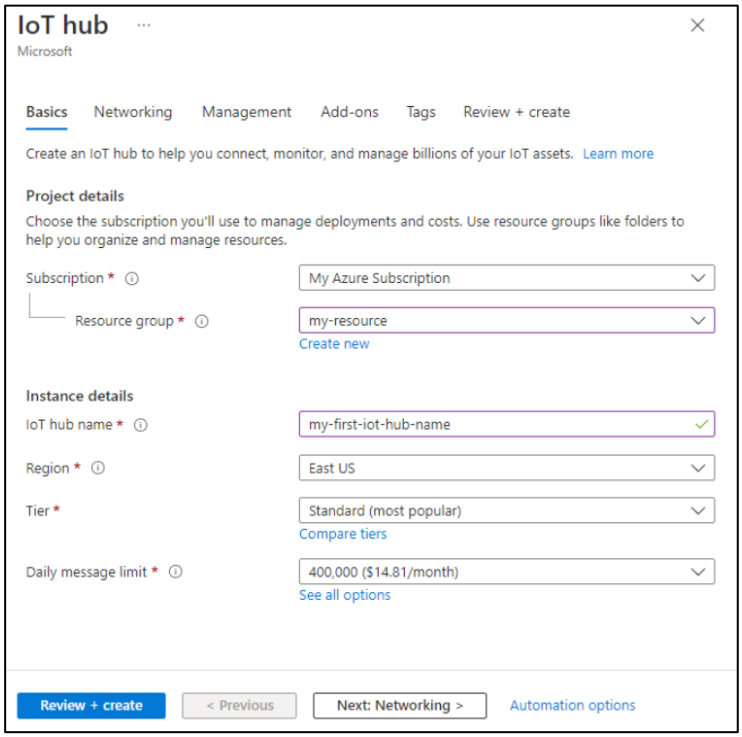


Figure 47:IoT hub Basics Settings

5- Select Next: Networking to continue creating your hub.
6- On the Networking tab, complete the fields as follows:

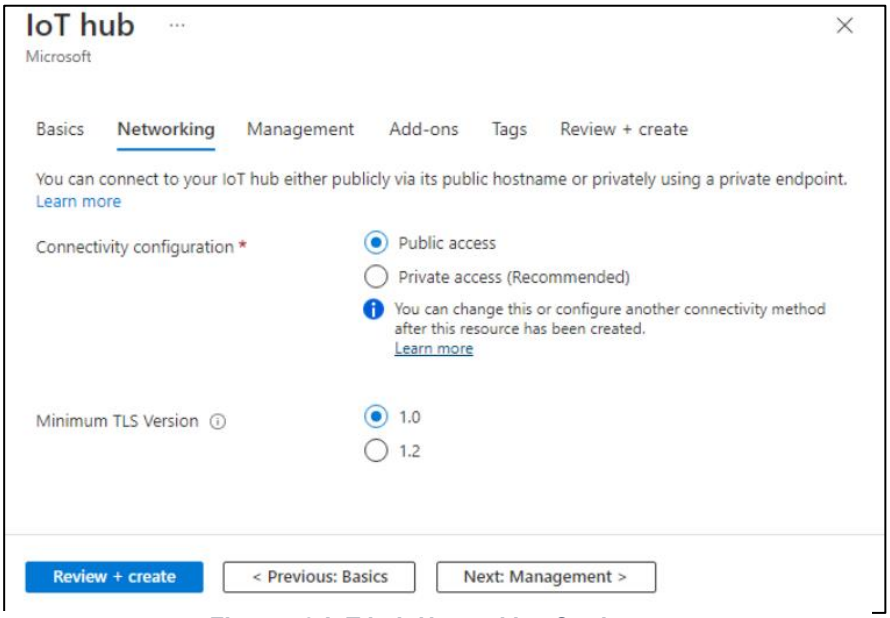


Figure 48:IoT hub Networking Settings

7- **Select Next: Management to continue creating your hub.**
8- **On the Management tab, accept the default settings. If desired, you can modify any of the following fields:**

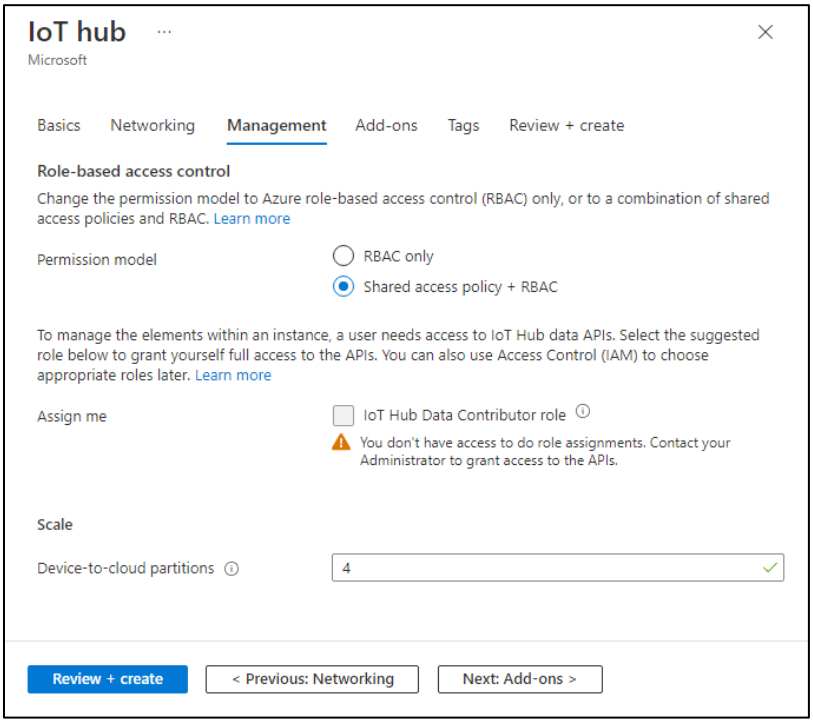


Figure 49:IoT hub Management Settings

9- **Select Next: Add-ons to continue to the next screen.**
10- **On the Add-ons tab, accept the default settings. If desired, you can modify any of the following fields:**

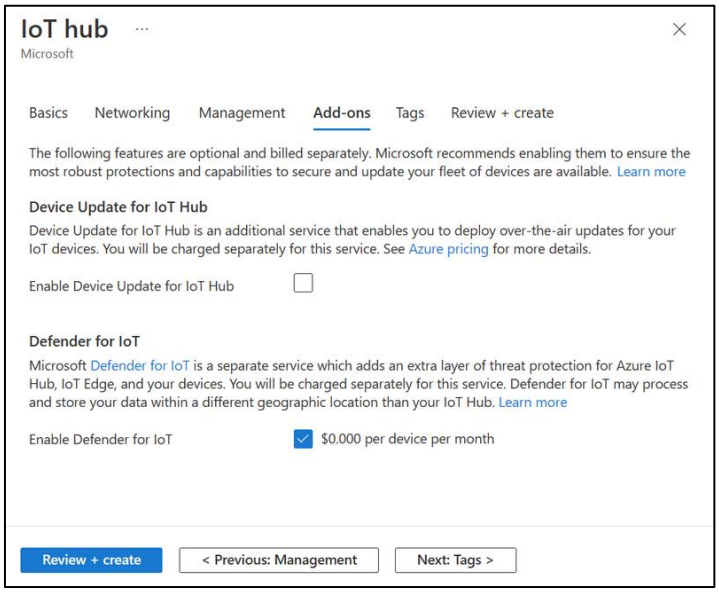


Figure 50:IoT hub Add-ons

11-Select Next: Tags to continue to the next screen.

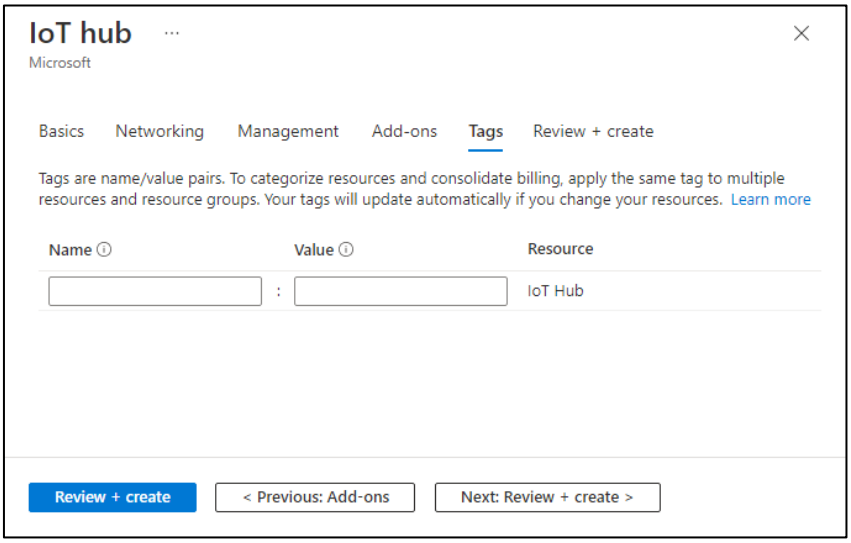


Figure 51:IoT hub Tags

12-Select Next: Review + create to review your choices.

13-Select Create to start the deployment of your new hub. Your deployment will be in progress in a few minutes while the hub is being created. Once the deployment is complete, select Go to Resource to open the new hub.

14-Register a new device in the IoT hub.

- **In your IoT hub navigation menu, open Devices, then select Add Device to add a device to your IoT hub.**

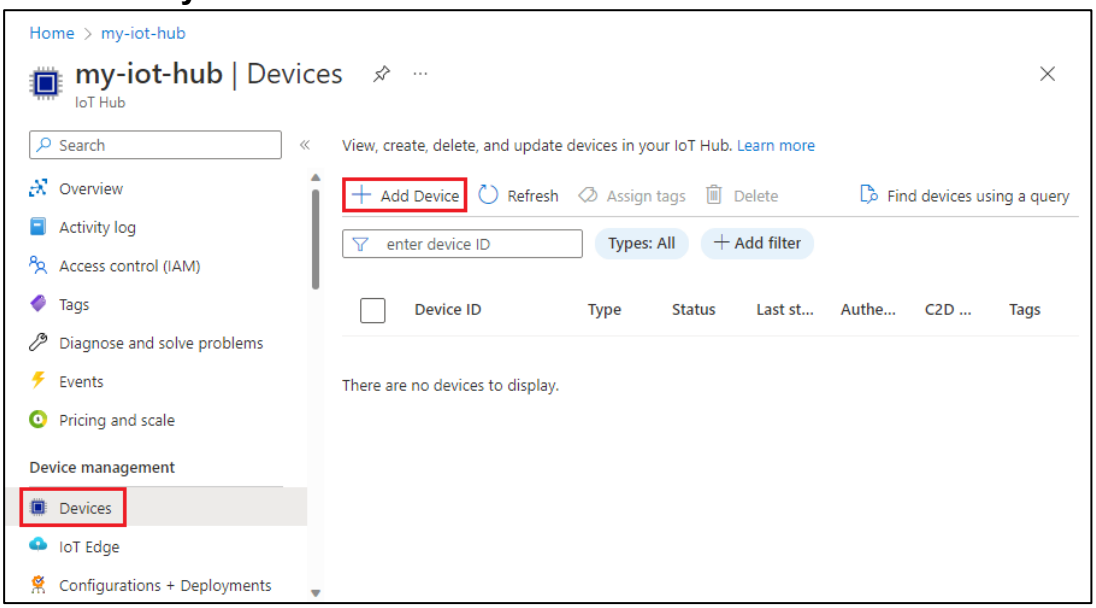


Figure 52:IoT hub [Add Device]

- **In Create a device, provide a name for your new device, such as deviceId, and select Save. This action creates a device identity for your IoT hub. Leave Auto-generate keys checked so that the primary and secondary keys will be generated automatically, also in my account, the Device ID is called "MultiProcess".**

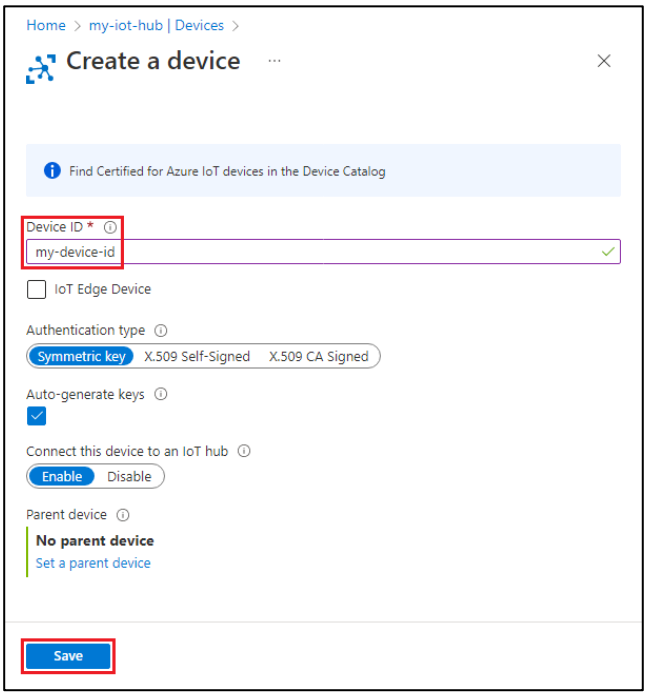


Figure 53:IoT hub [Create a device]

- **After the device is created, open the device from the list in the Devices pane. Copy the value of the Primary connection string. This connection string is used by the device code to communicate with the IoT hub.**
-
- **To send telemetry data of sensors from a real system to azure IoT hub, we use [azure sdk for c] library for doing this task, code of esp32 is attached in Appendix B.**

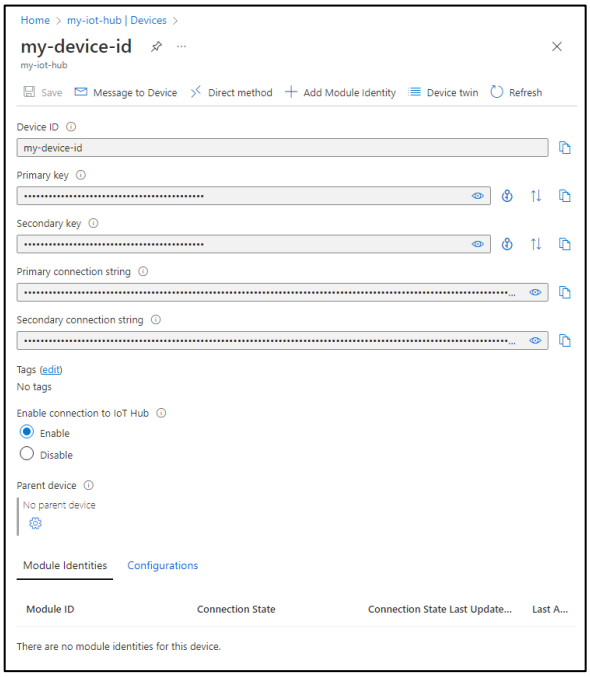


Figure 54:Device Security Data

15-Routing Telemetry message:

- **Route messages to "Event Hub" that I will identify how to create in the next sections.**
- **Click on Message routing.**
- **Click on Add, then convert you to Add a route, you can modify any changes in the following field.**

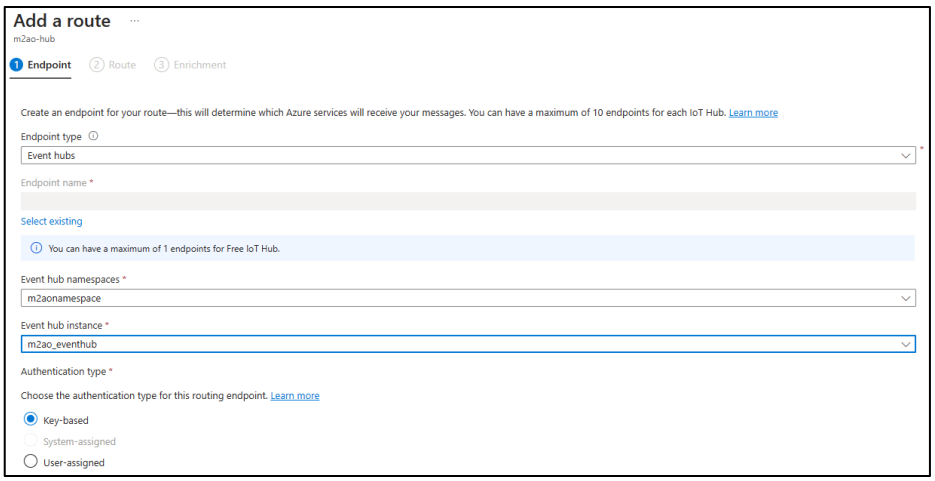


Figure 55:Add Route to Event Hub

- **Finally, you must find the Route added to the IoT Hub as shown in the image.**

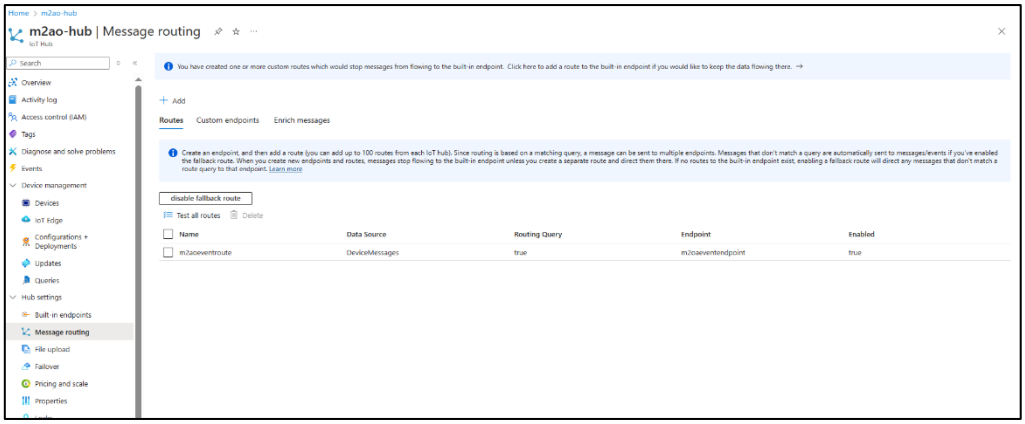


Figure 56:Message routing review

Now, Any Telemetry data sent to Device" MultiProcess" on IoT Hub by ESP32 will be routed to Event Hub Service. Next Step, we will identify how to create an Event Hub on Azure Portal.

8.2.2.2 Azure Event Hub

Figure 57:Event Hub

Azure Event Hubs is a Big Data streaming platform and event ingestion service that can receive and process millions of events per second. Event Hubs can process, and store events, data, or telemetry produced by distributed software and devices. Data sent to an event hub can be transformed and stored using any real-time analytics provider or batching/storage adapters.

1. **Create an Event Hubs namespace:**

An Event Hubs namespace provides a unique scoping container, in which you create one or more event hubs. To create a namespace in your resource group using the portal, do the following actions:

1. **In the Azure portal, Click on Create a resource.**

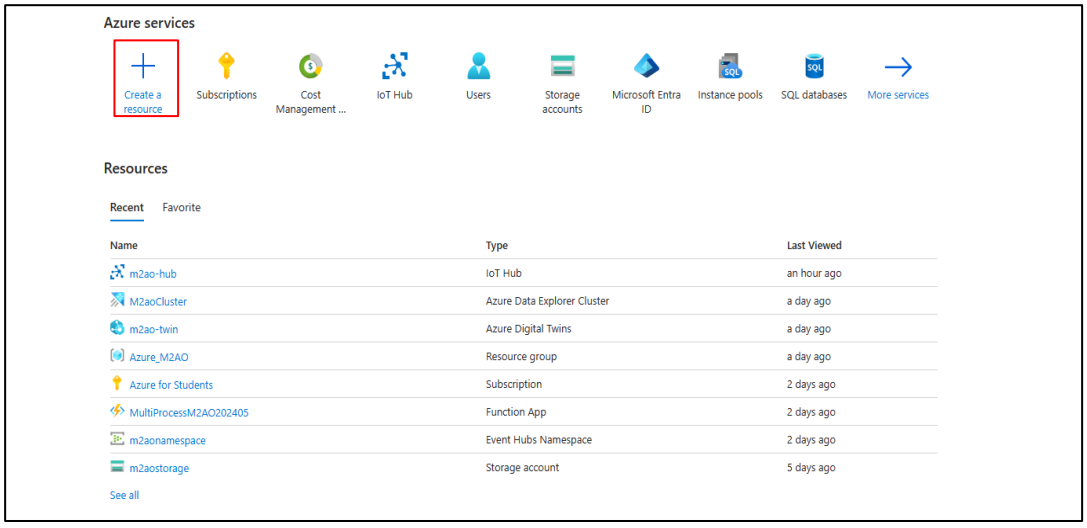


Figure 58:Create a resource [Event Hub]

2. **Select Event Hub.**

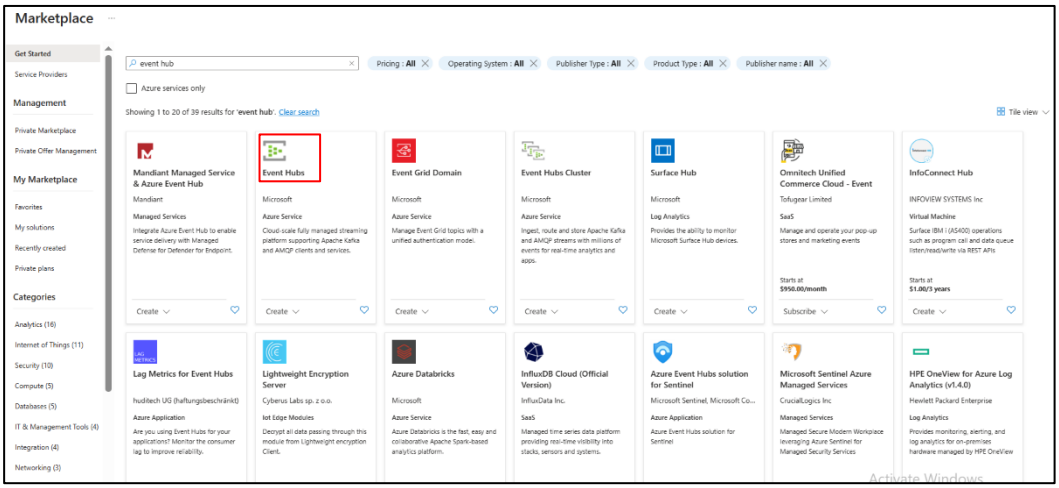


Figure 59:Select Event Hub

3. **Click on Create:**

Figure 60:Create Event Hub

4. **On the Create namespace page, take the following steps:**

   - **Select the subscription, for my account [Azure for Student]**
   - **Select the resource group, [Azure_M2AO].**
   - **Enter a name for the namespace[“m2aonamespace].**
   - **Select a location for the namespace [East US].**
   - **Choose Standard for the pricing tier [Basic].**
   - **Leave the throughput units (for Basic tier) or processing units (for premium tier) settings as it is.**

- **Select Review + Create at the bottom of the page.**

On the Review + Create page, review the settings and select Create. Wait for the deployment to complete.

5. **On the Deployment page, select Go to resource to navigate to the page for your namespace.**

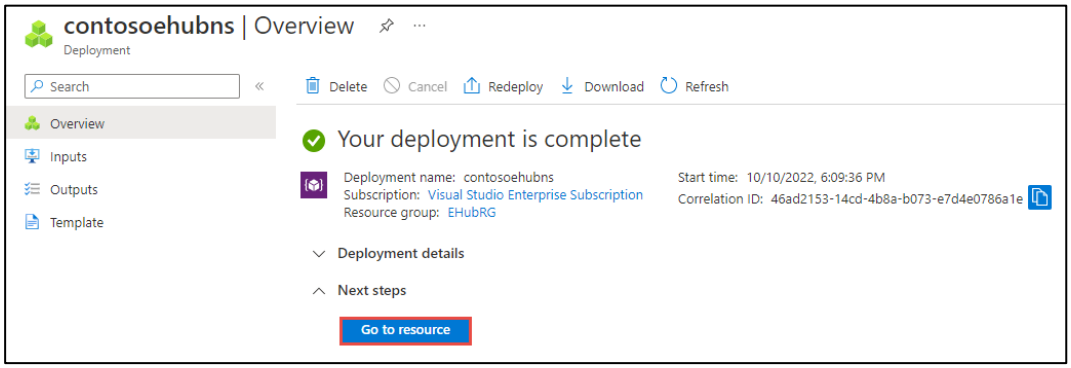


Figure 62: Go to Resource [Event Hub]

6. **Confirm that you see the Event Hubs Namespace page as the following, but I will show the Event Hub Namespace of my account.**

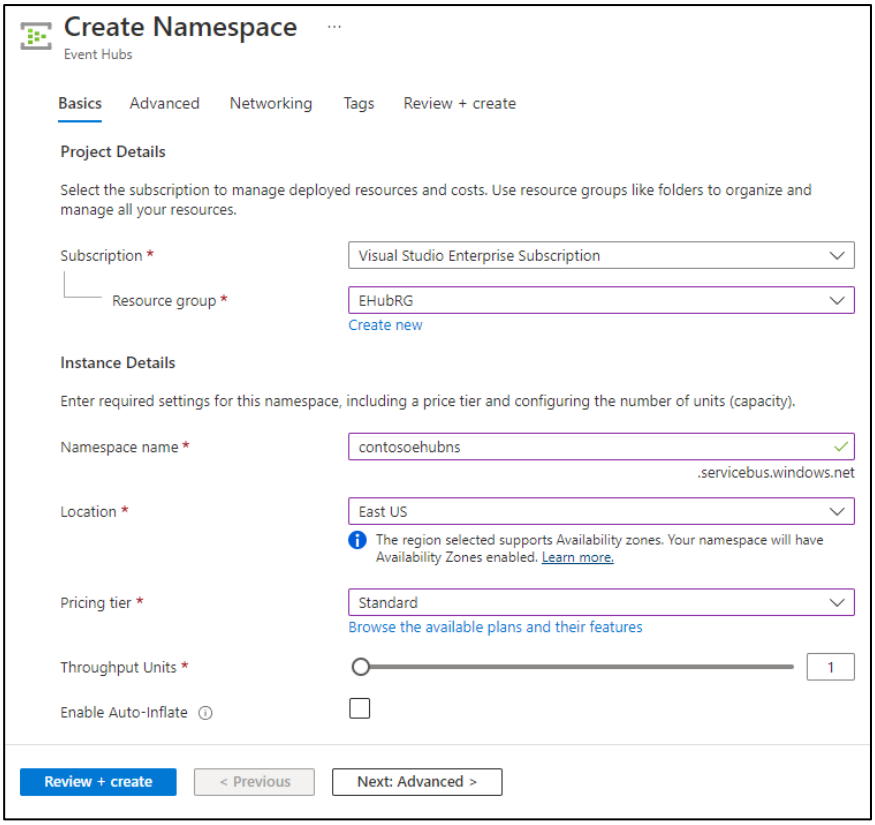


Figure 61:Create Namespace

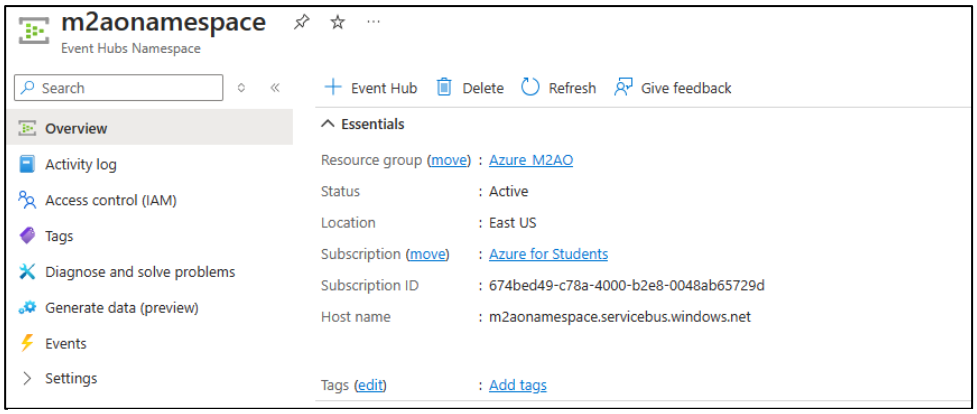


Figure 63:Event Hub Namespace overview.

2. Create an event hub:

To create an event hub within the namespace, do the following actions:

2. **On the Overview page, select + Event hub on the command bar.**

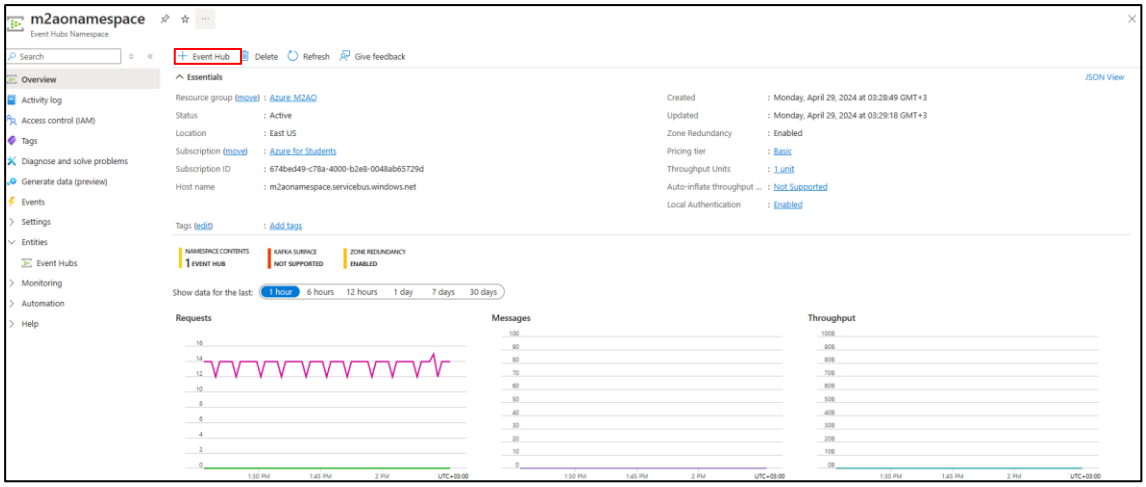


Figure 64:Add Event in Namespace

3. **Type a name for your event hub, then select Review + Create. In My Account, Name is “m2ao_eventhub”.**

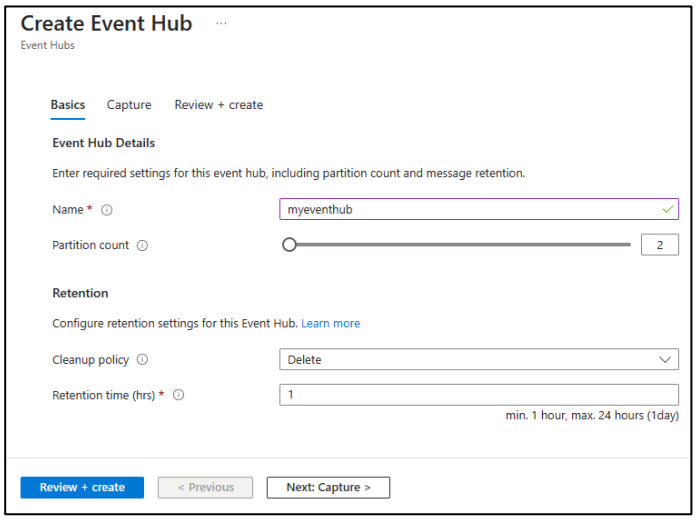


Figure 65:Create Event

The partition count setting allows you to parallelize consumption across many consumers. For more information, see Partitions. The message retention setting specifies how long the Event Hubs service keeps data. For more information, see Event Retention.

4. **On the Review + Create page, select Create.**
5. **You can check the status of the event hub creation in alerts. After the event hub is created, you see it in the list of event hubs, the following is that I created on my account.**

Figure 66:Event Hub Check

8.2.2.3 Azure Function:

Azure Functions is a serverless solution that allows you to write less code, maintain less infrastructure, and save on costs. Instead of worrying about deploying and maintaining servers, the cloud infrastructure provides all the up-to-date resources needed to keep your applications running.

Figure 67:Azure Function

You focus on the code that matters most to you, in the most productive language for you, and Azure Functions handles the rest.

In our case, we will create an Azure function by Microsoft Visual Studio, this function will be responsible for ingesting telemetry data from the IoT Hub Device which is called "MultiProcess" by the Event Hub Trigger through the Event Hub we created in the previous sections.

1- Click on “Create a new project”.

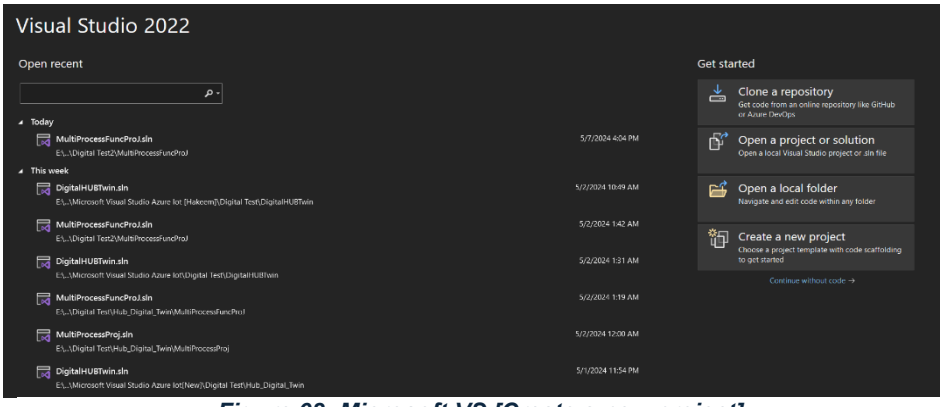


Figure 68: Microsoft VS [Create a new project]

2- Write Azure Functions in the Search tab and click on Azure Function.

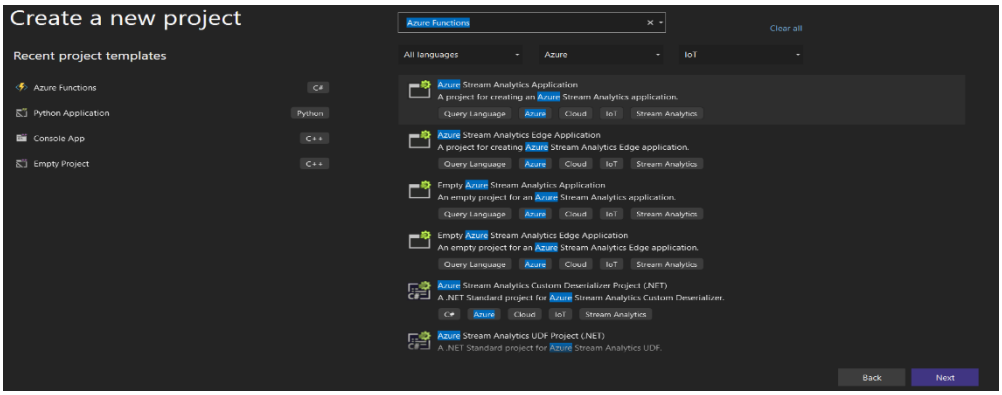


Figure 69:Select Azure Function

3- Put the Name of the Project and Click Next.

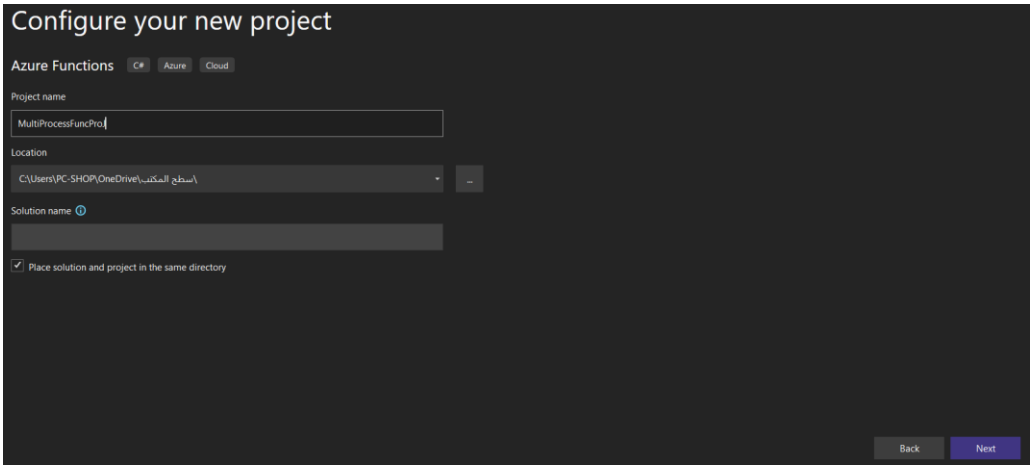


Figure 70:Configuration of Project

4- Select Function Worker and Function as the following image, you must make sure that the Event Hub name is the same as that we created previously on the Azure portal, and the the Connection String setting name save it in any place with you so that we will use it again in the code.

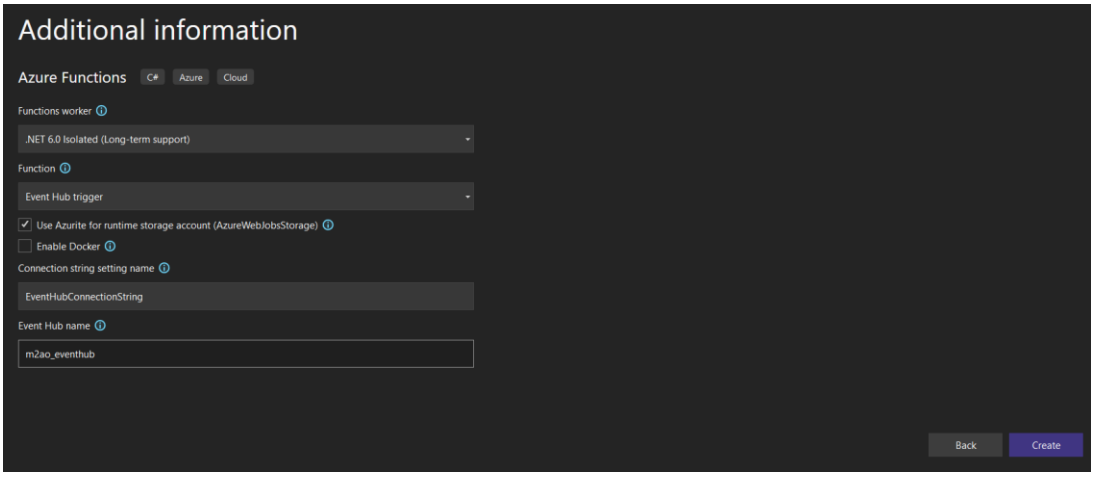


Figure 71:Additional information.

5- After Creating the Function, you will find a file called "local. settings. json", you must add "EventHubConnectionString" which contains on connection string of Event Hub, which you can get from Azure Portal, also you must add "ADT_SERVICE_URL" that In another place will contain on Host Name of Digital Instance.

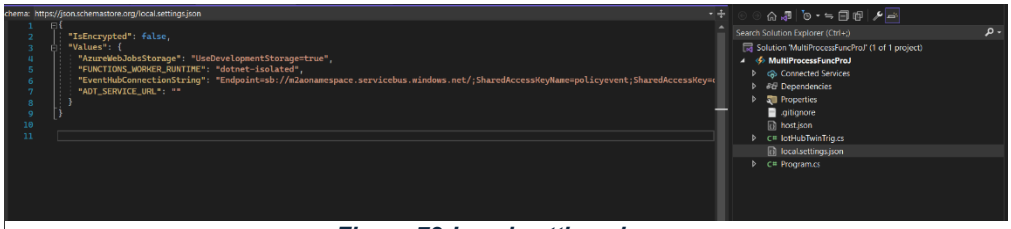


Figure 72:Local.settings.json

6-Write the Code of Function, so you can find it in the Appendix, Add the following packages to your project (you can use the Visual Studio NuGet package manager, or the dotnet add package command in a command-line tool).

- `Azure.DigitalTwins.Core`
- `Azure. Identity`
- `Azure.Messaging.EventHubs`
- `Newtonsoft.Json.Linq`

7-Publish this Function to your Azure Portal by clicking on Publish.

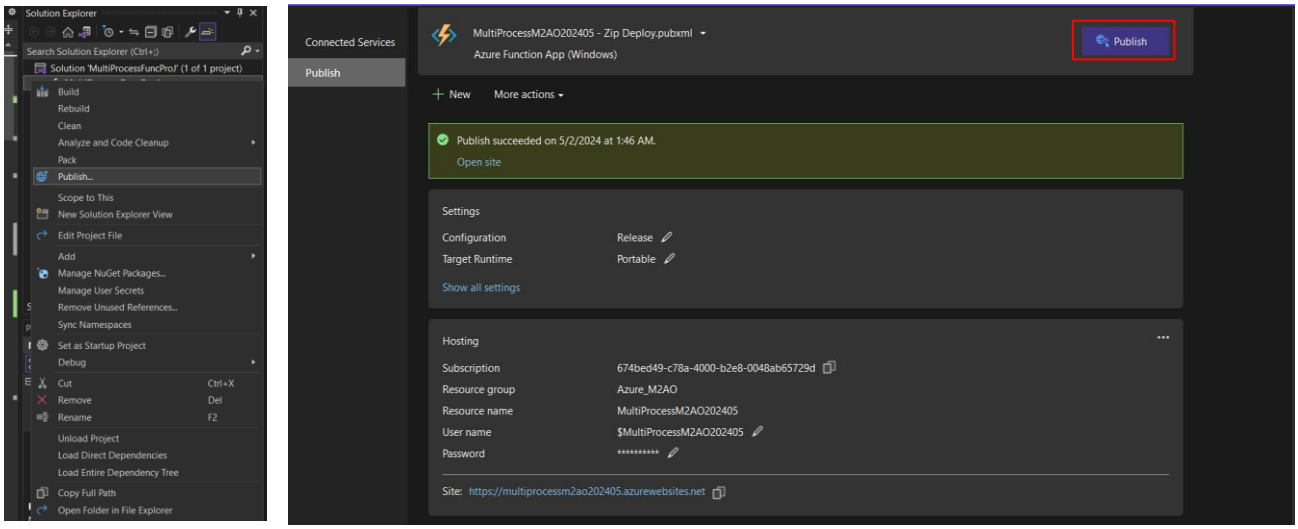


Figure 73:Publish Azure Function

8- Now you can check that the publishing process is done successfully from the Azure portal, open the Azure_M2AO Resource on your account, and you must find the function “IoTHubTwinTrig” there.

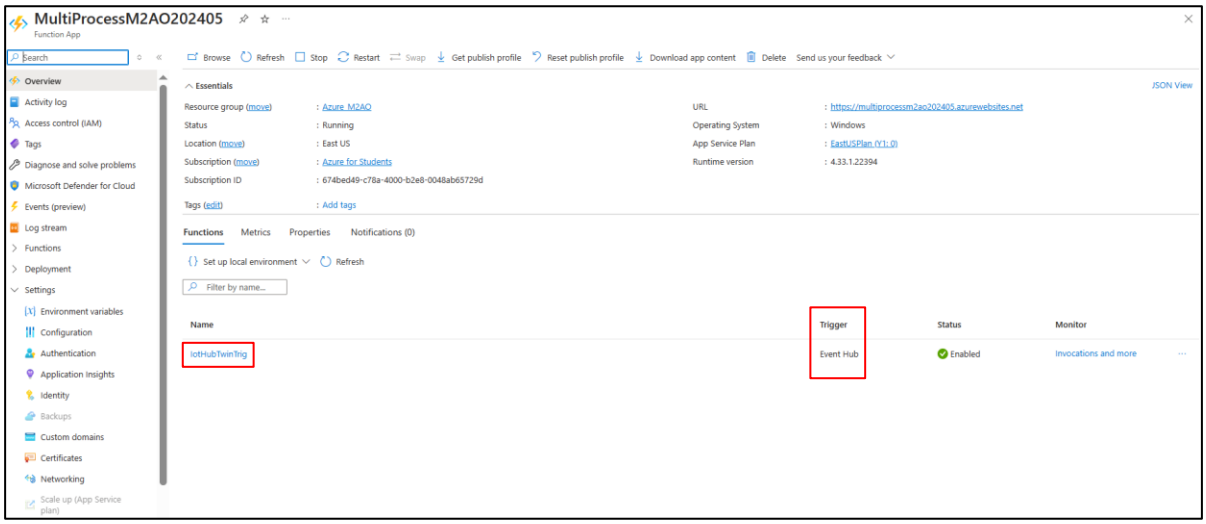


Figure 74:Publish Process Check

9-In Azure Function, Click on Environment Variables.

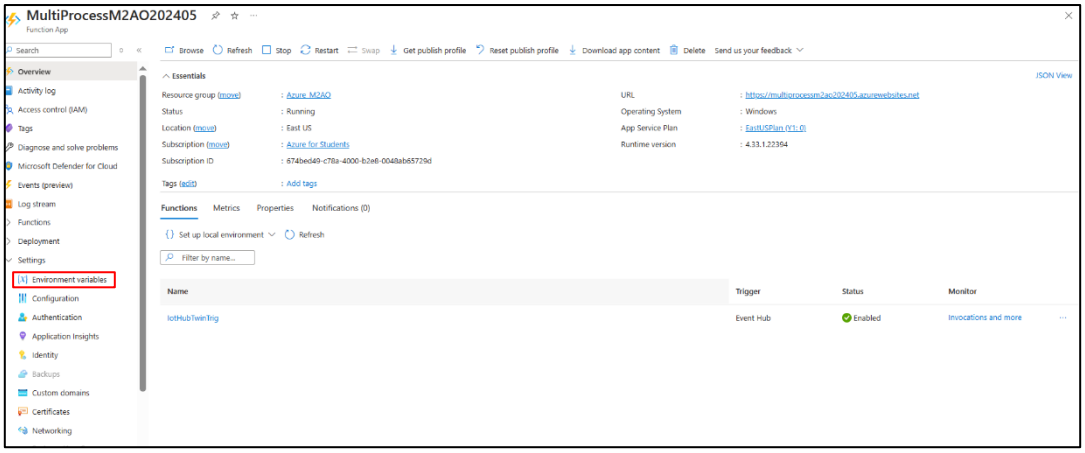


Figure 75: Click Environment Variables

10-Add "ADT_SERVICE_URL", and "EventHubConnectionString".

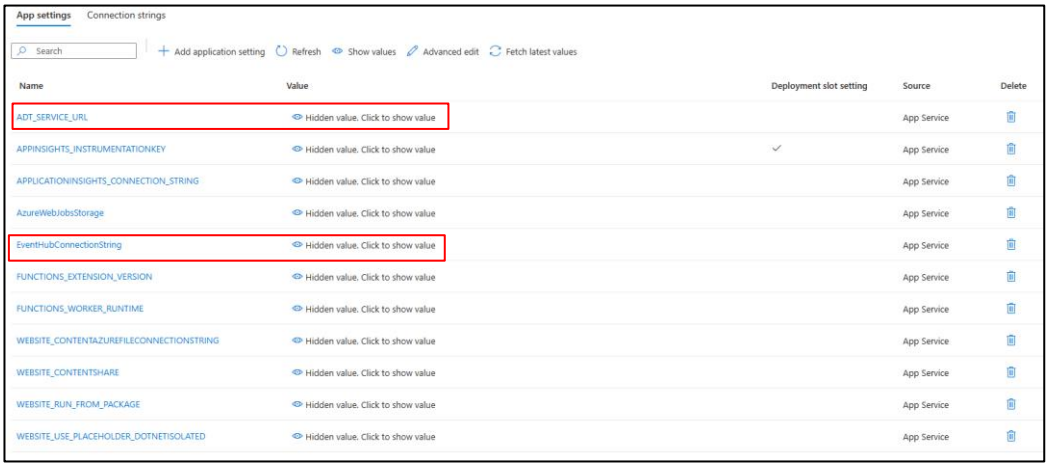


Figure 76: Add my Environment Variables

11-Assign an access role On Azure CLI:

The Azure function requires a bearer token to be passed to it. To make sure the bearer token is passed, grant the function app the Azure Digital Twins Data Owner role for your Azure Digital Twins instance, which will give the function app permission to perform data plane activities on the instance.

1. Use the following command to create a system-managed identity for your function (if the function already has one, this command will print its details). Take note of the principalId field in the output. You'll use this ID to refer to the function so that you can grant it permissions in the next step.

```
az functionapp identity assign --resource-group <your-resource-group> --name <your-function-app-
```

2.Use the principalId value in the following command to give the function the Azure Digital Twins Data Owner role for your Azure Digital Twins instance.

```
az dt role-assignment create --dt-name <your-Azure-Digital-Twins-instance> --assignee "<principal-ID>" --role "Azure Digital Twins Data Owner"
```

8.2.2.4 Azure Digital Twin

Azure Digital Twins is a platform as a service (PaaS) offering that enables the creation of twin graphs based on digital models of entire environments, which could be buildings, factories, farms, energy networks, railways, stadiums, and more—even entire cities. These digital models can be used to gain insights that drive better products, optimized operations, reduced costs, and breakthrough customer experiences.

Figure 77:Azure Digital Twin

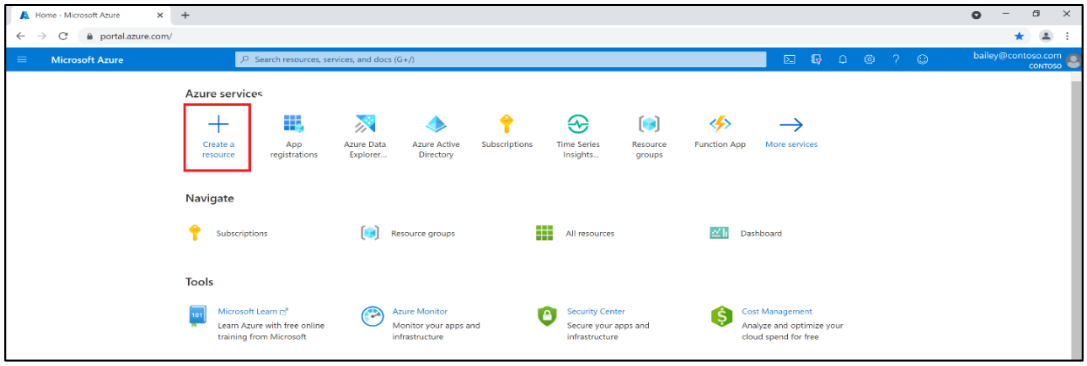
Azure Digital Twins can be used to design a digital twin architecture that represents actual IoT devices in a wider cloud solution and connects to IoT Hub device twins to send and receive live data.

Take advantage of your domain expertise on top of Azure Digital Twins to build customized, connected solutions that:

- **Model any environment and bring digital twins to life in a scalable and secure manner.**
- **Connect assets such as IoT devices and existing business systems, using a robust event system to build dynamic business logic and data processing.**
- **Query the live execution environment to extract real-time insights from your twin graph.**
- **Build connected 3D visualizations of your environment that display business logic and twin data in context.**
- **Query historized environment data and integrate with other Azure data, analytics, and AI services to better track the past and predict the future.**

1. Create an Azure Digital Twins instance:

1. **Once in the portal, start by selecting Create a resource in the Azure services home page menu.**



2. **Search for *Azure Digital Twins* in the search box and choose the Azure Digital Twins service from the results.**

Figure 78:Create a resource [Digital Twin]

Leave the Plan field set to Azure Digital Twins and select the Create button to start creating a new instance of the service.

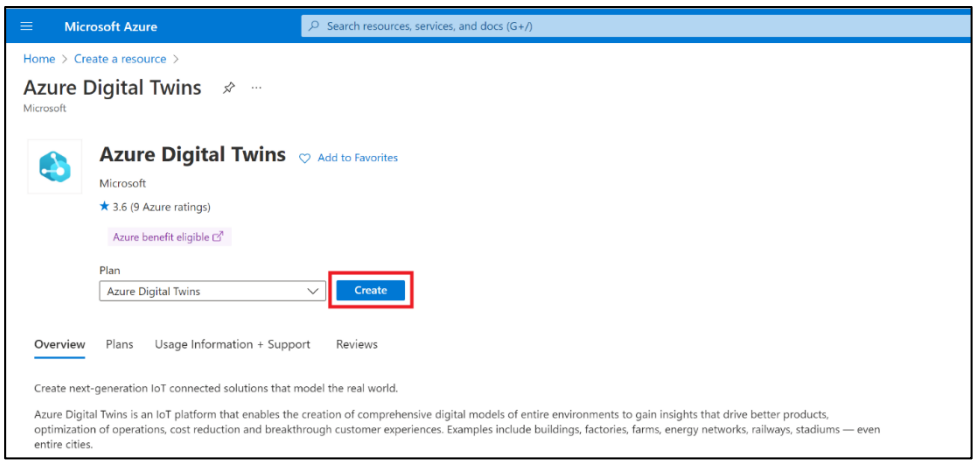


Figure 79:Click on Create [Digital twin]

3. **Fill in the fields on the Basics tab of setup, including your Subscription, Resource group, a Resource name for your new instance, and Region. Check the Assign Azure Digital Twins Data Owner Role box to permit yourself to manage data**

4. **in this instance. In my account, Subscription is “Azure For Students” , the Resource group is “Azure_M2AO” , the Resource name is “m2ao-twin”, and the Reg**

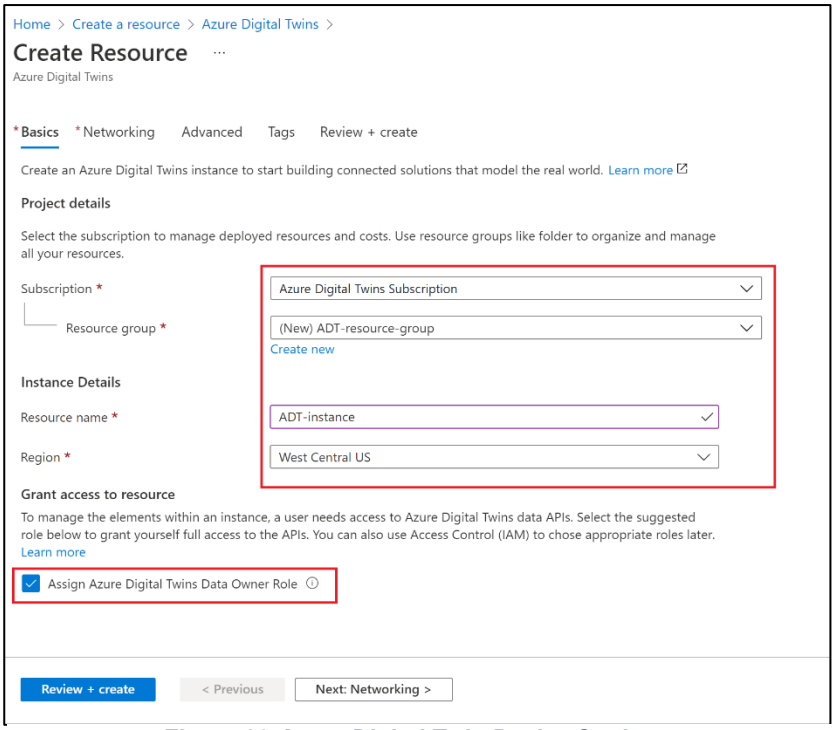


Figure 80:Azure Digital Twin Basics Settings

5. ion is EAST US.
6. Select Review + Create to finish creating your instance.
7. You will see a summary page showing the details you've entered. Confirm and create the instance by selecting Create. This will take you to an Overview page tracking the deployment status of the instance.

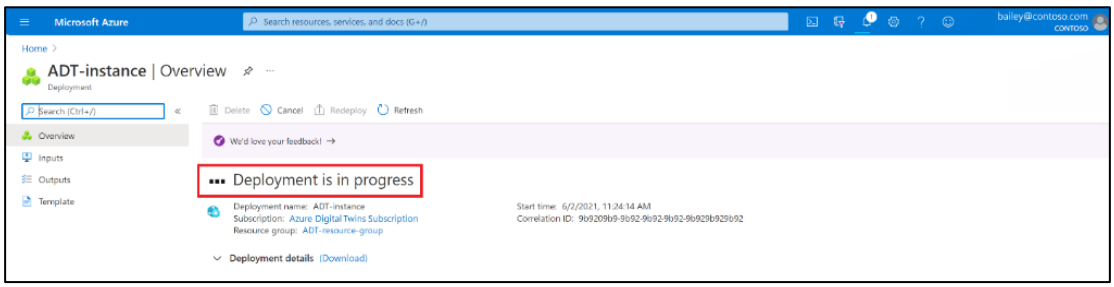


Figure 81:Azure Digital Twin Deployment status

Wait for the page to say that your deployment is complete.

2-Open instance in Azure Digital Twins Explorer:

After deployment completes, use the Go to resource button to navigate to the instance's Overview page in the portal.

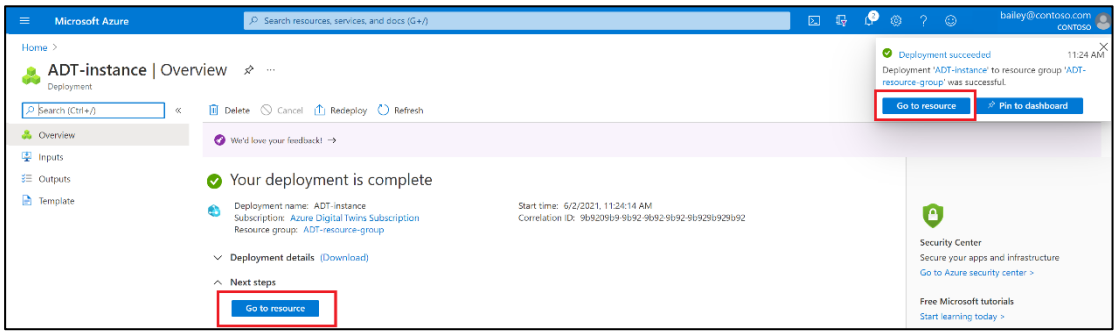


Figure 82:Go to resource [Azure Digital Twin]

Next, select the Open Azure Digital Twins Explorer (preview) button, I will show from my account.

Figure 83:Azure Digital Twin Explorer (Open)

This will open Azure Digital Twins Explorer in a new tab. If this is your first time using Explorer, you'll see a welcome modal summarizing its key features.

Next, you'll use Azure Digital Twins Explorer to set up the sample models and twin graphs of your system, I'll show you my System models, and twin by importing the model files and the twin graph file from my browser.

What are Models?

The first step in creating an Azure Digital Twins graph is to define the vocabulary for your environment. Models are generic definitions for each type of entity that exists in your environment.

Models for Azure Digital Twins are written in Digital Twin Definition Language (DTDL), a data object language like JSON-LD. Each model describes a single type of entity in terms of its properties, relationships, and components.

Upload the models (.json files)

In Azure Digital Twins Explorer, follow these steps to upload Model files.

1- In the Models panel, select the Upload a model icon that shows an arrow pointing upwards.

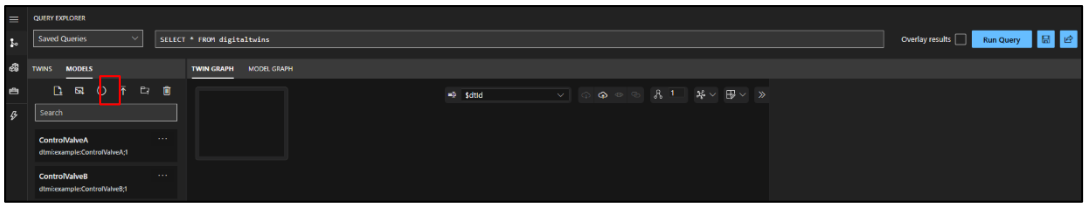


Figure 84:Model Icon

2-In the Open window that appears, navigate to the folder containing the [. Json] files on your browser.

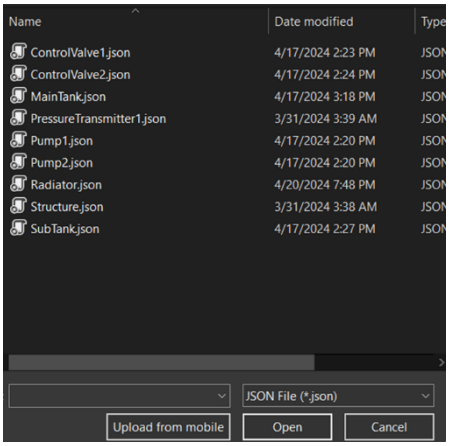


Figure 85:Models. Json

3-Select All Files. Json, and select Open to upload them all at once, Azure Digital Twins Explorer will upload these model files to your Azure Digital Twins instance. They should show up on the Models panel and display their friendly names and full model IDs.

Twins and the twin graph:

Now that our model definitions have been uploaded to your Azure Digital Twins instance, you can use these definitions to create digital twins for the elements in your environment.

Every digital twin in your solution represents an entity from the physical environment. You can create many twins based on the same model type. The twins will be connected with relationships into a twin graph that represents the full system.

In this section, I'll upload a pre-created graph containing All twins of my system.

- **Import the graph (.xlsx file)**

In Azure Digital Twins Explorer, follow these steps to import the graph.

1. In the Twin Graph panel, select the Import Graph icon that shows an arrow pointing into a cloud.

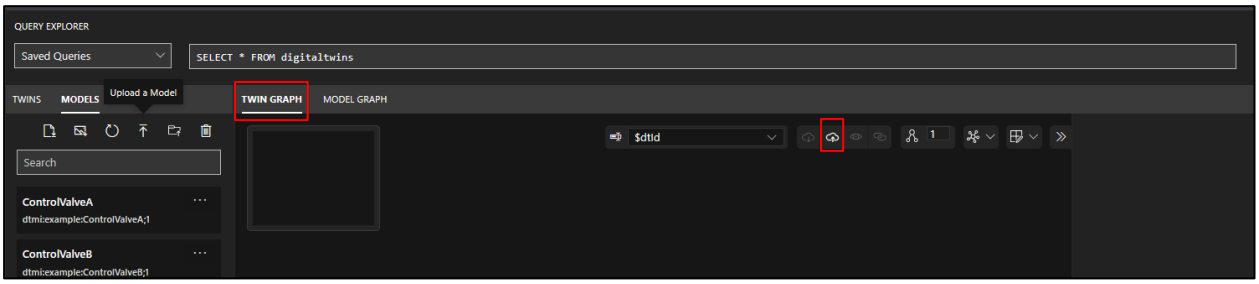


Figure 86:Import Graph file

2. In the Open window, navigate to the SystemScenario.xlsx file and select Open.

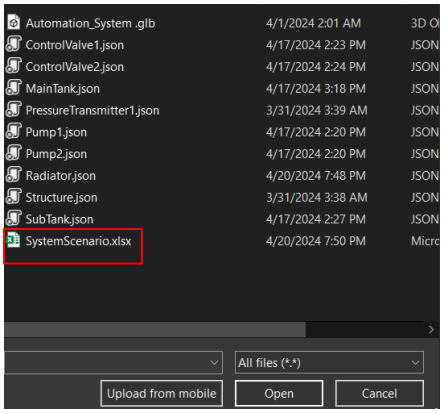


Figure 87:Graph.xlsx

StructureA	contains	{"TT1_PLC1": 0, "TT2_PLC3": 0, "Stirrer":0, "J1":0}
MainTank	contains	{"Pump1":0}
PumpA	contains	{"FT1_PLC2": 0 }
PressureTransmitterA	contains	{"FC1":0}
ControlValveA	contains	{"WT1_PLC2": 0 }
SubTank	contains	{"LC1":0}
ControlValveB	contains	{"Pump2":0}
PumpB	contains	{"Fan":0 }

Figure 88:Showing (Graph.xlsx)

After a few seconds, Azure Digital Twins Explorer opens an Import view that shows a preview of the graph to be loaded.

3. To finish importing the graph, select the Save icon in the upper-right corner of the graph preview panel.

4. Azure Digital Twins Explorer will use the uploaded file to create the requested twins and the relationships between them. Make sure you see the following dialog box indicating that the import was successful before moving on.

Select Close, the graph has now been uploaded to Azure Digital Twins Explorer, and the Twin Graph panel will reload. It will appear empty.

5. To see the graph, select the Run Query button in the Query Explorer panel, near the top of the Azure Digital Twins Explorer window.

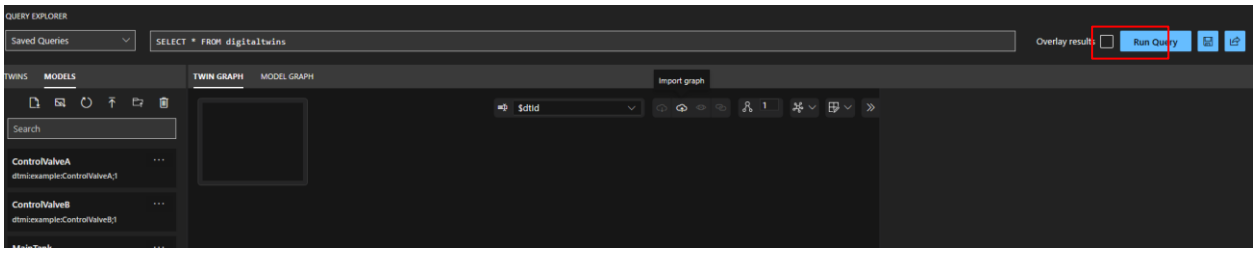


Figure 89:Run Query

This action runs the default query to select and display all digital twins. Azure Digital Twins Explorer retrieves all twins and relationships from the service. It draws the graph defined by them in the Twin Graph panel.

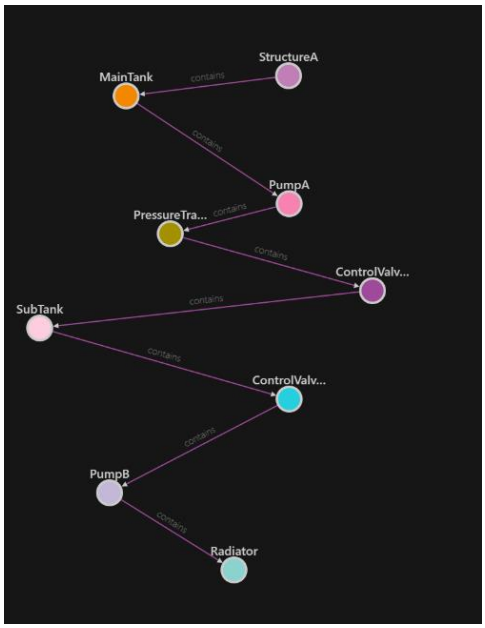


Figure 90:Digital Twins of System

- View twin properties:

MainTank Data:

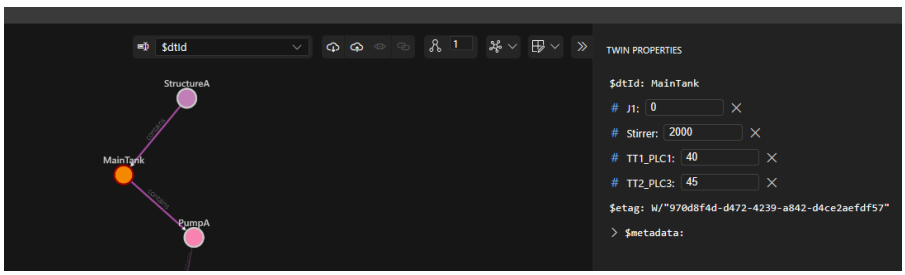


Figure 91:Main Tank Properties

Radiator Data:

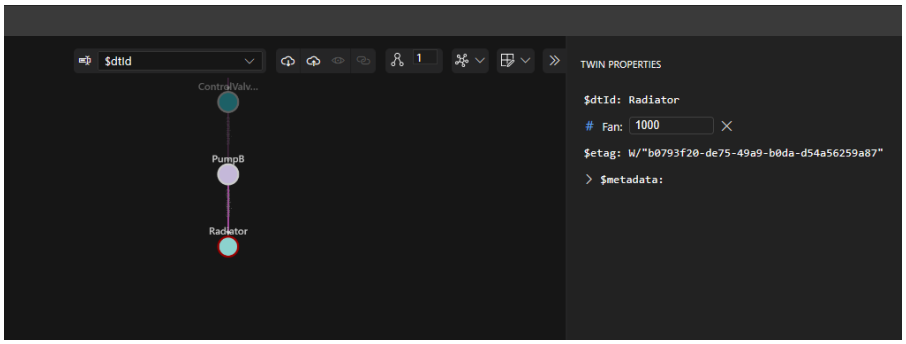


Figure 92:Radiator Property

Sub Tank Data:

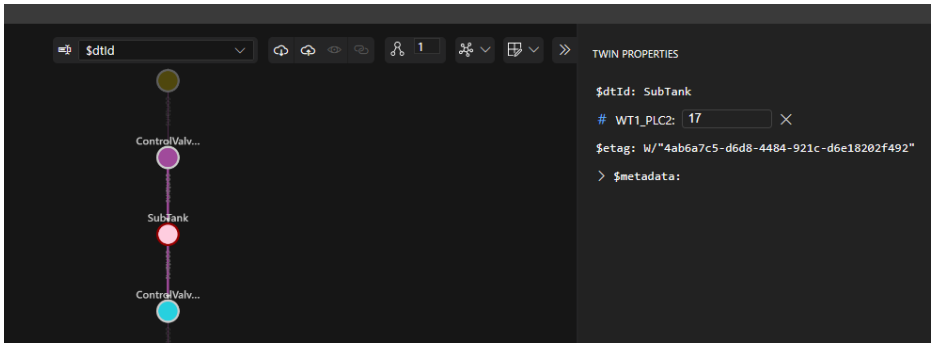


Figure 93:Sub Tank Property

3- 3D Scenes Studio for Azure Digital Twins:

To build a 3d visualization of our system for monitoring, analysis, and query on business insights, you must follow these steps.

1-Create storage resources

create a new storage account and a container in the storage account. 3D Scenes Studio will use this storage container to store your 3D file and configuration information, You'll also set up read and write permissions for the storage account. To set these backing resources up quickly, this section uses the Azure Cloud Shell.

1. **Navigate to the Cloud Shell in your browser, Run the following command to set the CLI context to your subscription for this session, my subscription is "Azure For Students."**

```
az account set --subscription "<your-Azure-subscription-ID>"
```

2. **Run the following command to create a storage account in your subscription. The command contains placeholders for you to enter a name and choose a region for your storage account and a placeholder for your resource group. My**

Storage account is “m2aostorage”, the resource group is “Azure_M2AO”, and my Location is “EAST US”.

```
az storage account create --resource-group <your-resource-group> --name <name-for-your-storage-account> --location <region> --sku Standard_RAGRS
```

When the command completes successfully, you'll see your new storage account details in the output. Look for the ID value in the output and copy it to use in the next command

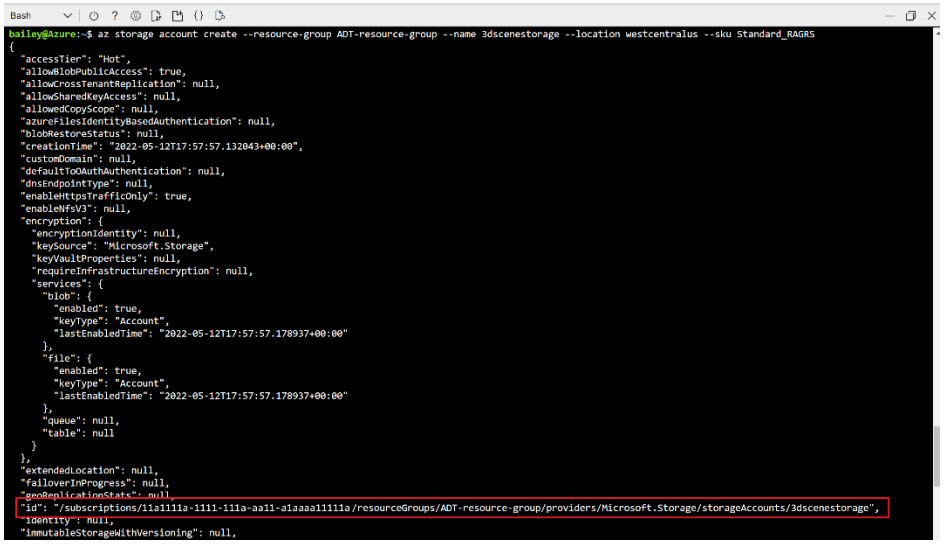


Figure 94:Storage Id

3. **Run the following command to grant yourself the *Storage Blob Data Owner* on the storage account. This level of access will allow you to perform both read and write operations in 3D Scenes Studio. The command contains placeholders for the email associated with your Azure account and the ID of your storage account that you copied in the previous step.**

```
az role assignment create --role "Storage Blob Data Owner" --assignee <your-Azure-email> --scope <ID-of-your-storage-account>
```

When the command completes successfully, you'll see details of the role assignment in the output.

4. **Run the following command to configure CORS for your storage account. This will be necessary for 3D Scenes Studio to access your storage container. The command contains a placeholder for the name of your storage account.**

```
az storage cors add --services b --methods GET OPTIONS POST PUT --origins https://explorer.digitaltwins.azure.net --allowed-headers Authorization x-ms-version x-ms-blob-type --account-name <your-storage-account>
```

This command doesn't have any output.

5. **Run the following command to create a private container in the storage account. Your 3D Scenes Studio files will be stored here. The command contains a placeholder for you to enter a name for your storage container and a placeholder for the name of your storage account. My container name is “m2aocontainer”.**

```
az storage container create --name <name-for-your-container> --public-access off --account-name <your-storage-account>
```

When the command completes successfully, the output will show "created": true.

2-Initialize your 3D Scenes Studio environment

Now that all your resources are set up, you can use them to create an environment in *3D Scenes Studio*. In this section, you'll create a scene and customize it for the System graph that's in your Azure Digital Twins instance.

1. **Navigate to the 3D Scenes Studio.**

2. **Select the Edit icon next to the instance name to configure the instance and storage container details.**

Figure 95:Edit to Suitable Container

- **For the Azure Digital Twins instance URL, fill in the *hostname* of your instance from the Collect hostname step into this URL: https://<your-instance-hostname>.**
- **For the Azure Storage account URL, fill in the name of your storage account from the Create storage resources step into this URL: https://<your-storage-account>. blob.core.windows.net.**
- **For the Azure Storage container name, enter the name of your storage container from the Create storage resources step.**

- **Select Save.**

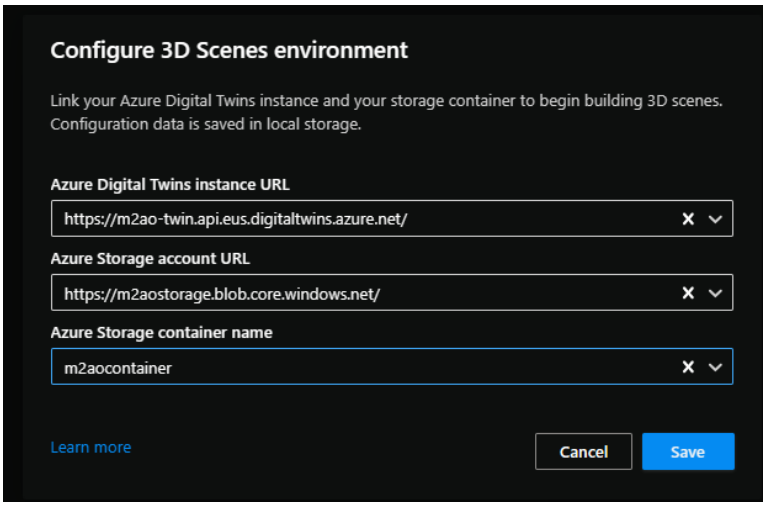


Figure 96:Configure 3d Scenes environment.

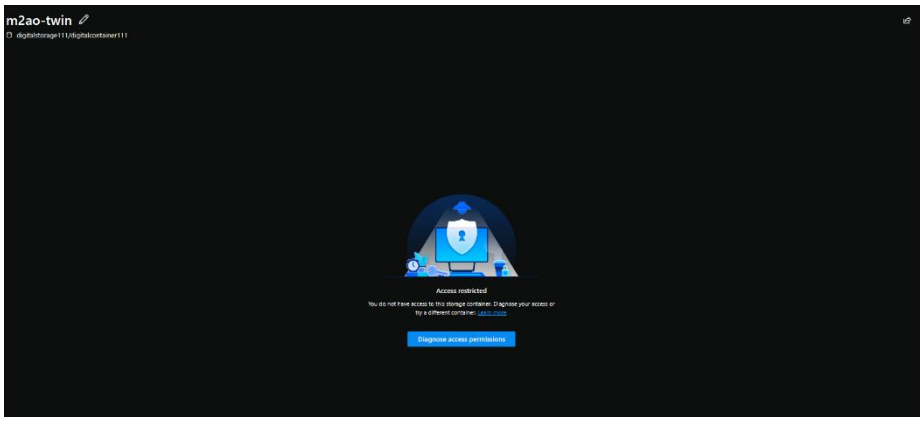


Figure 97:3D Scene View

3. Create a 3D scene.

In this section, I'll identify how to create 3D scene in extension. glb a new 3D scene,

<u>Here are the steps to prepare a mechanical model for use with Azure Digital Twin:</u>

1- Design the Mechanical Model: Utilize 3D modeling software like Solidworks to create the mechanical model of the system.

2- Ensure Compatibility: Ensure that the model is saved in a format compatible with Azure Digital Twin. STEP (.step or .stp) format is commonly used for interoperability between different CAD software.

3- Conversion to GLB Format: Convert the model to GLB (.glb) format, which is the preferred format for 3D models in Azure Digital Twin.

4- Utilize Conversion Software: Employ software like Blender to convert the model to GLB format because Solidworks doesn't directly support GLB conversion.

5- Verify Individual Part Selection: Ensure that after conversion, the model retains its integrity and that individual parts can still be selected and manipulated. This is important for interactive experiences within Azure Digital Twin.

6- Check for Compatibility Issues: Verify that the converted GLB model functions correctly within the Azure Digital Twin environment, ensuring that it accurately represents the physical system.

7- Upload to Azure Digital Twin: Finally, upload the GLB format of the mechanical model to Azure Digital Twin, where it can be integrated into the digital twin environment for simulation, monitoring, and analysis purposes.

Design the Mechanical Model

1. Description of the System:

The system comprises two interconnected tanks, each equipped with a pump, a valve, two temperature sensors, a pressure transmitter sensor, and a load cell sensor. The tanks serve as reservoirs for the fluid being transported or stored, while the pumps facilitate the transfer of fluid between the tanks or to external systems. The valves control the flow of fluid within the system, allowing for precise regulation and isolation of different sections. Temperature sensors monitor the temperature of the fluid within each tank, ensuring optimal operating conditions. Pressure transmitter sensors measure the pressure of the fluid, providing real-time data for monitoring and control purposes. Finally, load cell sensors measure the weight or mass of the fluid within the tanks, enabling accurate monitoring of fluid levels and quantities.

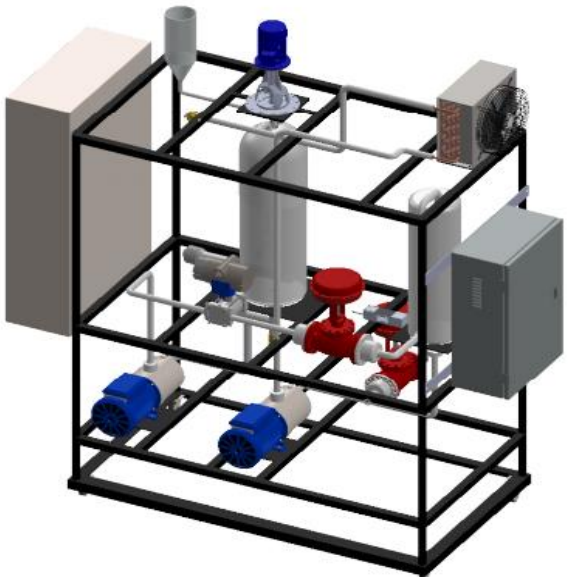


Figure 98:System components Overview

2. Ensure Compatibility

After drawing this system, it was converted to STEP format and saved, ready to be imported into Blender software

3. Conversion to GLB Format (using Blender):

what are the features of Blender software?

Blender is a powerful open-source 3D creation suite that offers a wide range of tools, including:

- **Modeling**
- **Animation**
- **Rendering**
- **Compositing**
- **Motion Tracking**
- **Simulation**
- **Scripting and Customization**
- **Cross-Platform and Open Source**

What are the steps performed in Blender?

1. **Open the Model in Blender:**

- **Launch Blender and import the mechanical assembly model file (. STEP) by navigating to File > import and select gltf 2.0 (. glb/. gltf).**

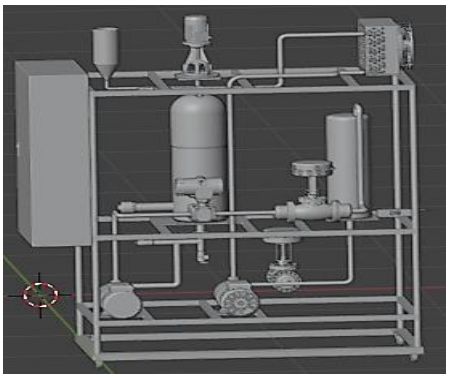


Figure 99:open model in the Blender

2. **Enter Edit Mode:**

- **Select the entire assembly by pressing A to select all parts.**

- **Enter Edit Mode by pressing Tab or selecting Edit Mode from the dropdown menu at the top-left corner of the viewport.**

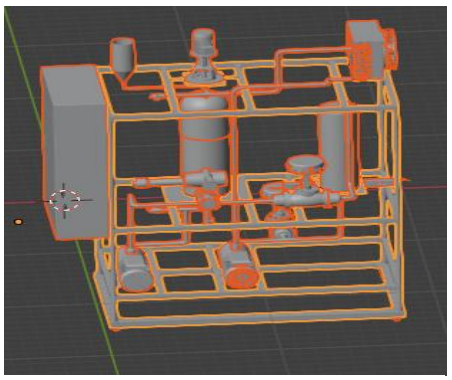


Figure 100:Enter Edit mode

3. **Separate Parts:**
   - **In Edit Mode, select the faces, vertices, or edges that define each individual part of the assembly.**
   - **Once selected, press P to bring up the Separate menu.**
   - **Choose "By Selection" from the Separate menu to separate the selected geometry into a new object. Repeat this process for each part of the assembly until all parts are separated.**

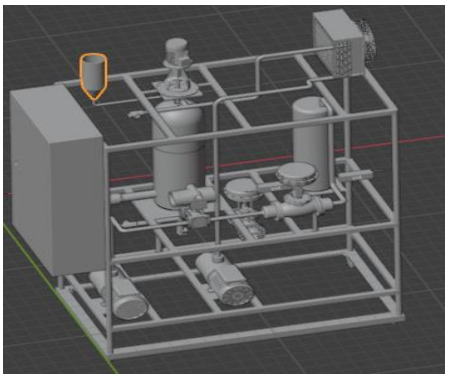


Figure 101:Feed_tank

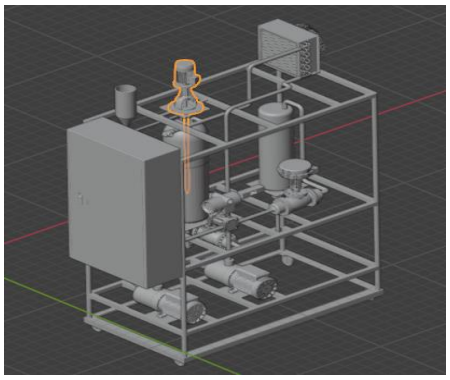


Figure 102:Stirrer

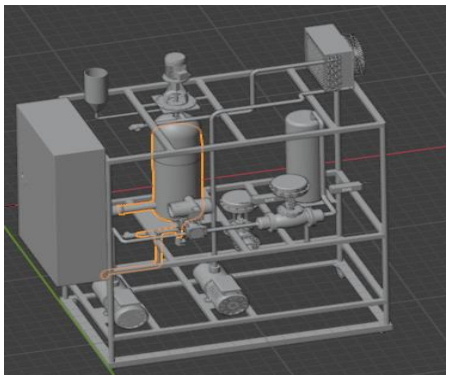


Figure 103:Main_tank

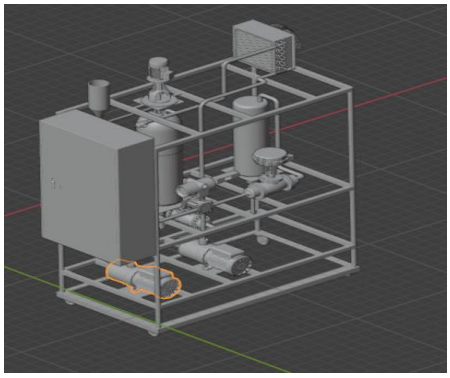


Figure 104:PumpA

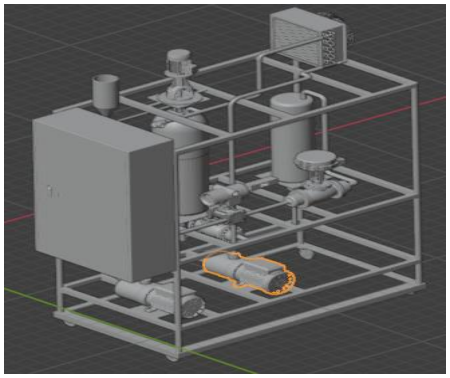


Figure 106:Pump_2

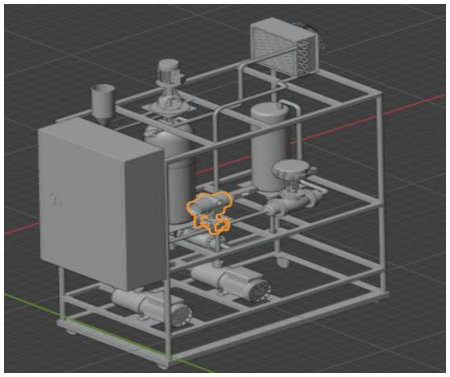


Figure 105:Pressure_Transmitter

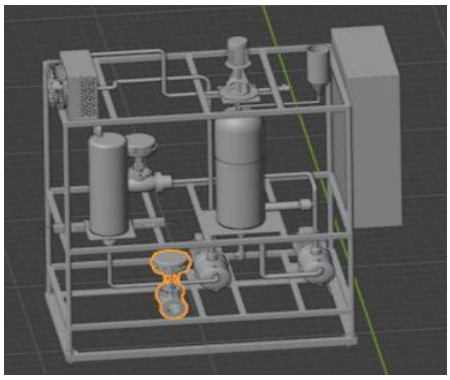


Figure 107: Vlave_2(Output)

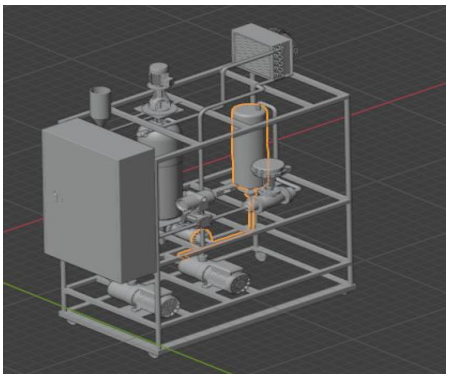


Figure 108: Small_tank

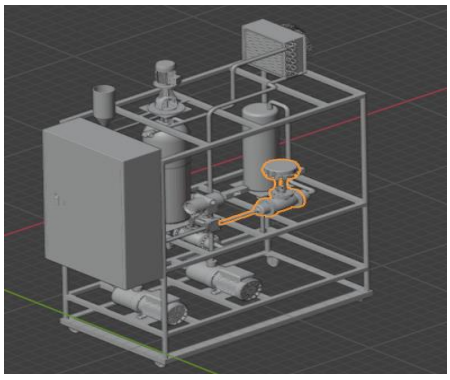


Figure 109: Valve_1

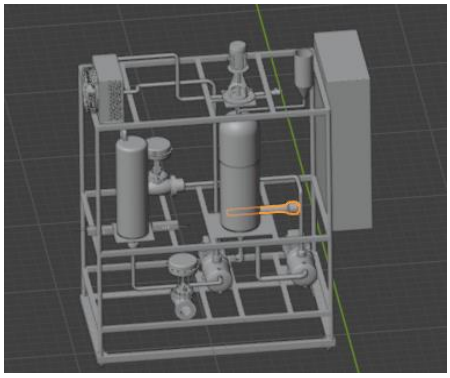


Figure 110: Electric_heater

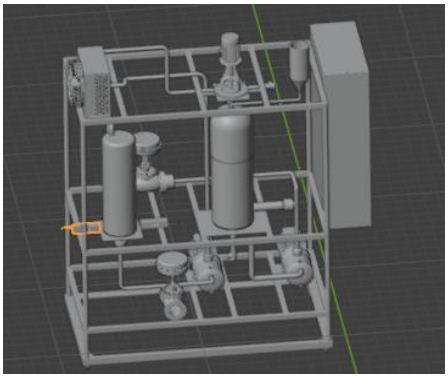


Figure 112:LoadCell_Sensor

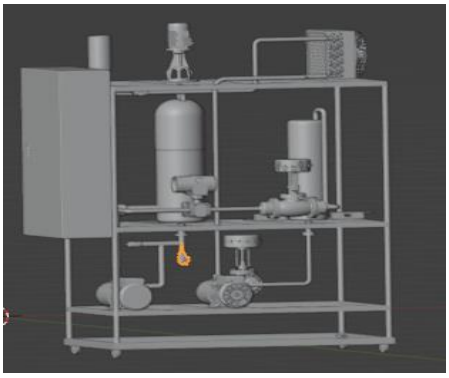


Figure 111:Temperature_Sensor2

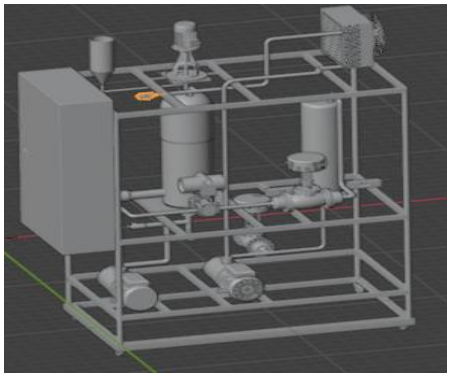


Figure 113:Temperature_Sensor1

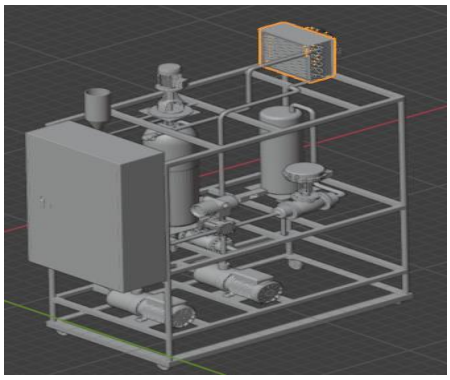


Figure 114:Radiatior

4. Rename Parts (Optional):

- **With each part selected, you can rename them in the Outliner panel on the right side of the screen. This will make it easier to identify and select each part later.**

5. Exit Edit Mode:

- **Once all parts are separated and named, exit Edit Mode by pressing Tab or selecting Object Mode from the dropdown menu.**

6. Select Individual Parts:

- **In Object Mode, you can now select each part individually by clicking on them in the viewport or the Outliner panel.**

7. Export to. glb Format:

- **With all parts selected or the entire assembly selected, navigate to File > Export > glTF (. glb/. gltf).**
- **In the Export dialog box, specify the export settings, such as the export location and whether to export selected objects or the entire scene.**
- **Click on the Export button to save the model in. glb format.**

8. Finalize Export:

- **In the Export glTF dialog, ensure that the desired export settings are selected, including whether to embed textures and materials.**
- **Click on the Export glTF button to finalize the export process.**

9. Verify Exported Model:

- **Once the export process is complete, verify that the exported .glb file contains all separated parts of the mechanical assembly.**
- **You can open the .glb file incompatible software or viewers to ensure that each part is correctly separated and preserved.**

So, We utilized Blender software to accomplish two tasks:

Firstly, to convert the model to .glb format.

Secondly, to separate each part, allowing us to select each component individually and label them accurately.

4. Add a 3D scene.

2. **Select the Add 3D Scene button to start creating a new scene. Enter a Name and Description for your scene and select Upload file.**
3. **Browse for the *Automation_System.glb* file and open it. Select Create.**

Once the file is uploaded, you'll see it listed back on the main screen of 3D Scenes Studio.

Figure 115:Automation_System.glb Added

4. **Select the scene to open and view it. The scene will open in Build mode.**

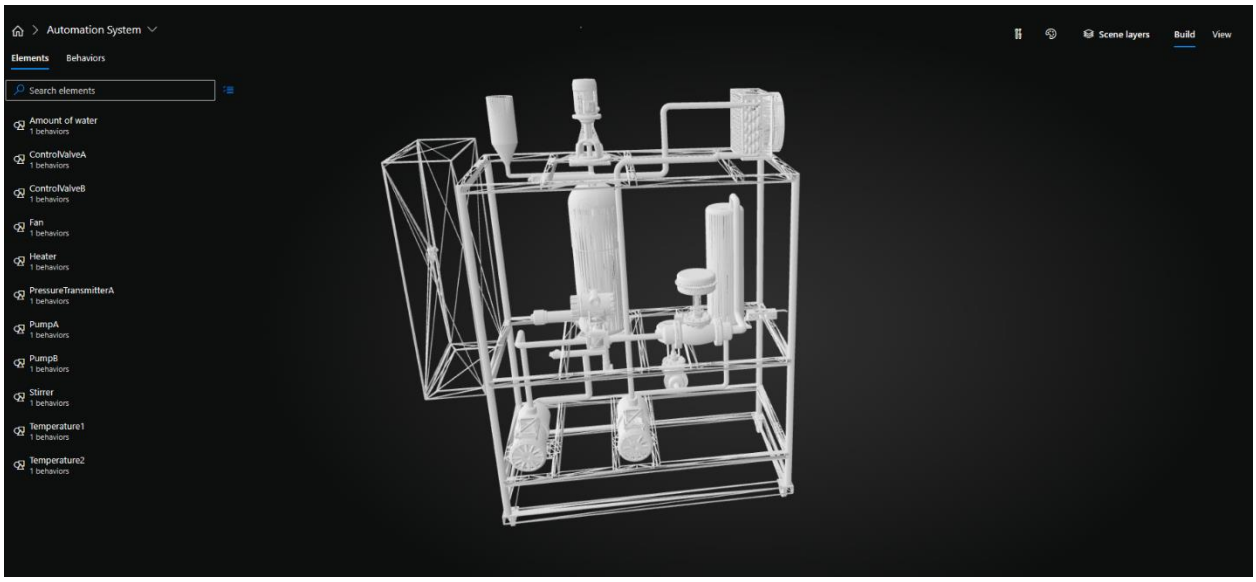


Figure 116:Build Mode of System

5. Create a scene element.

In this section, I will show you how to create one element such as “Temperature Sensor1”, the other elements are the same way.

1. **select temperature sensor1 and then click create a new element.**

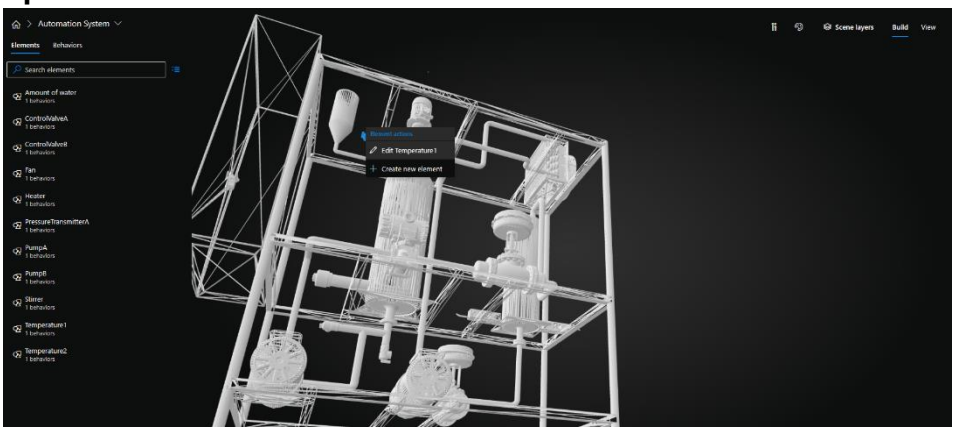


Figure 117:Create a new element.

2. Choose a Suitable twin, from the twins you uploaded previously.

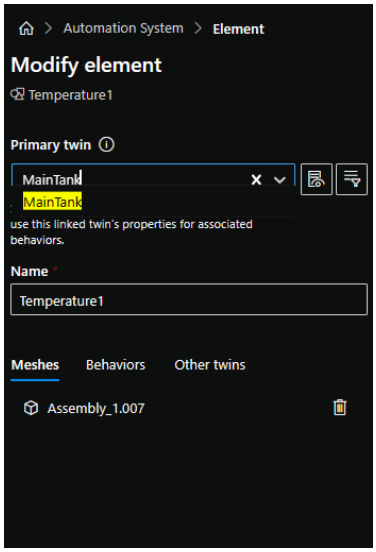


Figure 118:Element Settings

6. Create a behavior

1. click on new behavior.

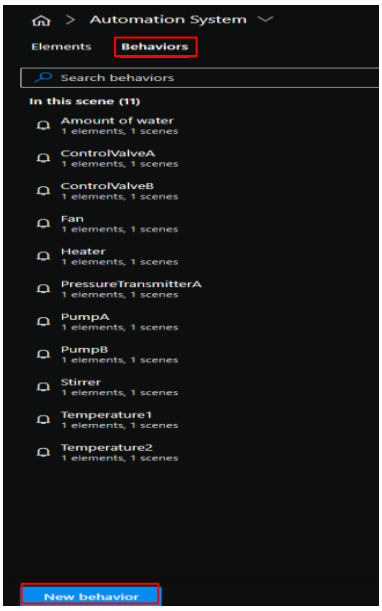


Figure 119:Create a new behavior

2.Write name of behavior, choose suitable element for this behavior.

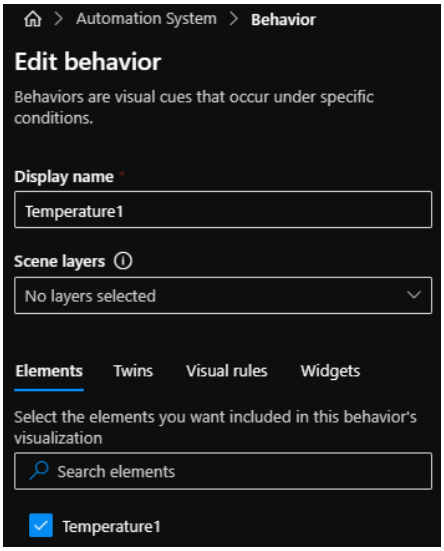


Figure 120: Behavior settings

3.click on Visual rule to create rules of this element.

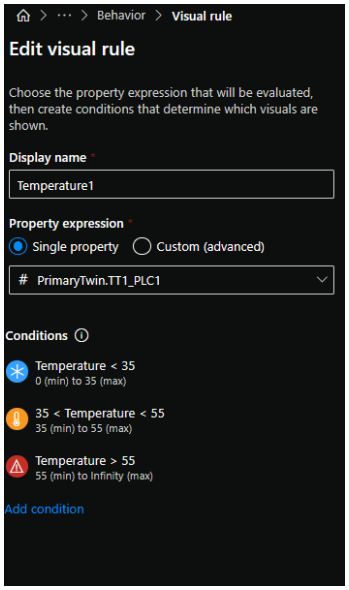


Figure 121: Add Visual rules

4.click on widgets, to show readings of sensor on it.

Figure 122:Add Widget

5. After this, you must see it like this.

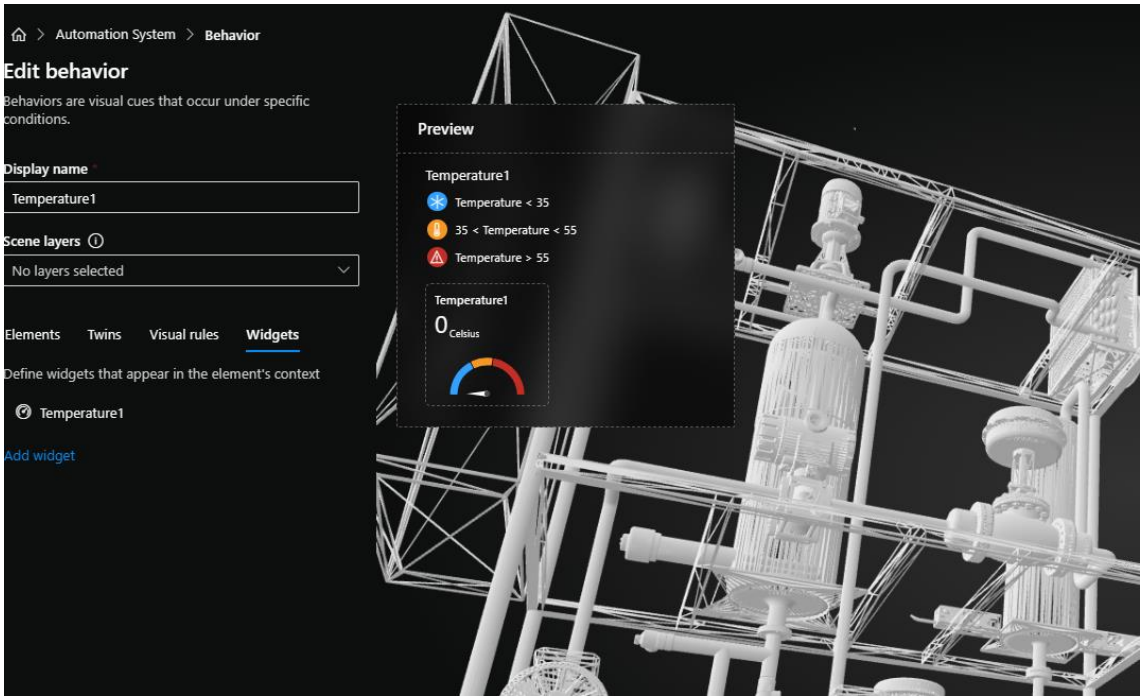


Figure 123:Behavior Overview

6. make these steps on each element in your system, and finally you will get this.

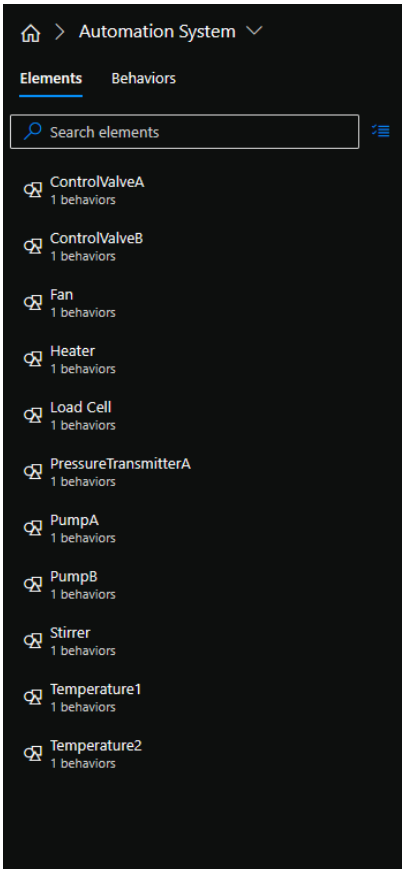


Figure 125:Elements Overview

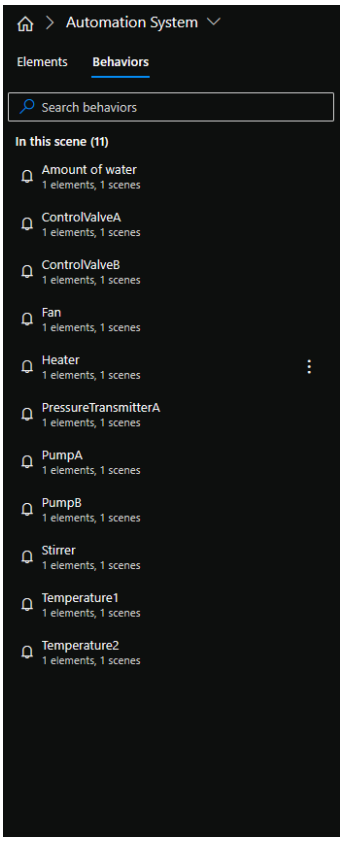


Figure 124:Behaviors Overview

7. View scene

Now, we built our system in 3d scene studio, and then to monitor, visualize, and analyze our system in parallel to the real environment, we will click on the view tab on the right of the build tab.

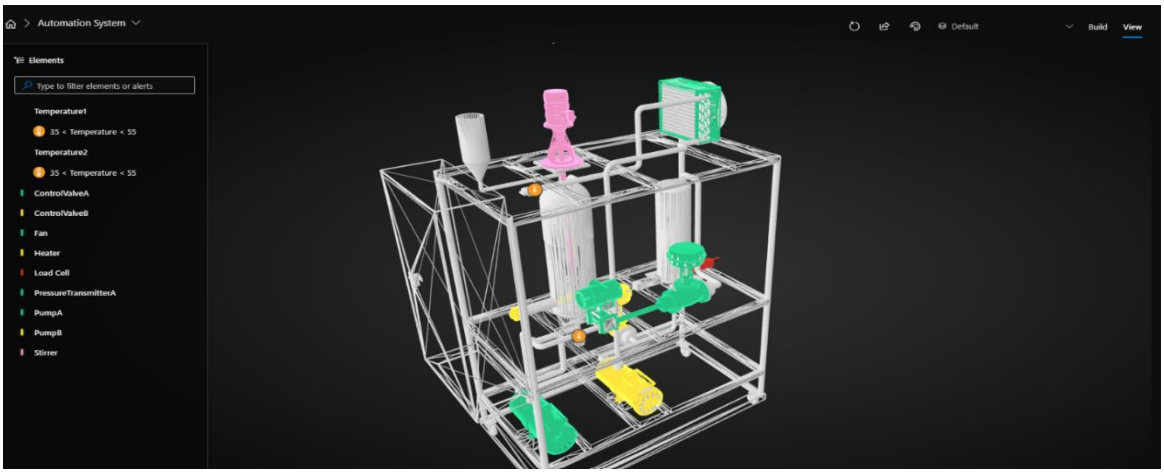


Figure 126:View Mode

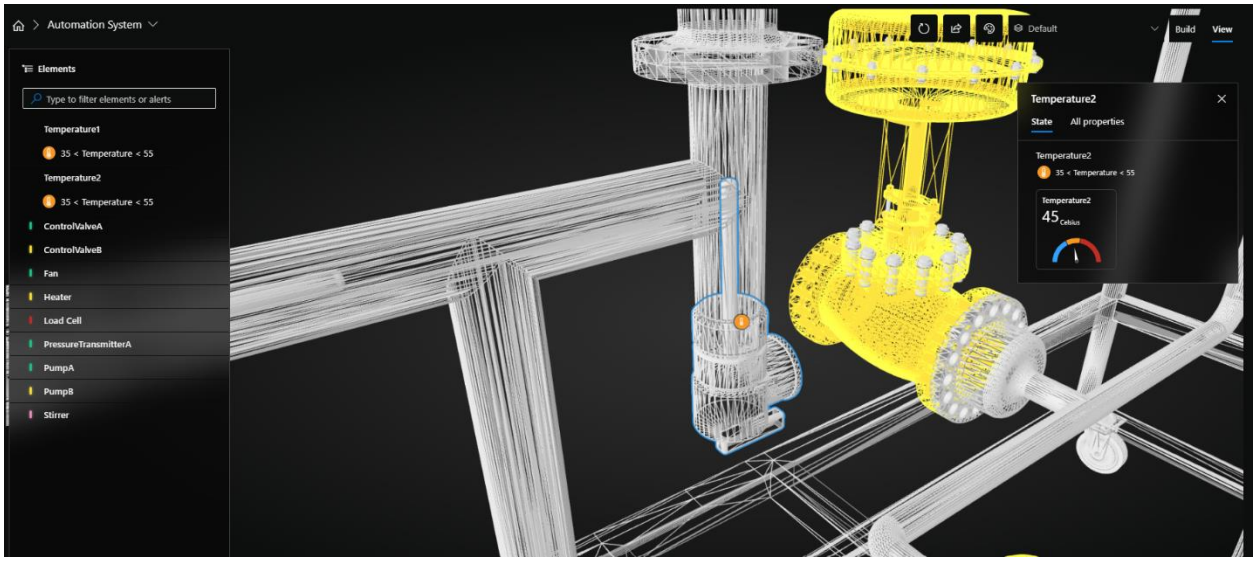


Figure 127:Temperature1 Sensor

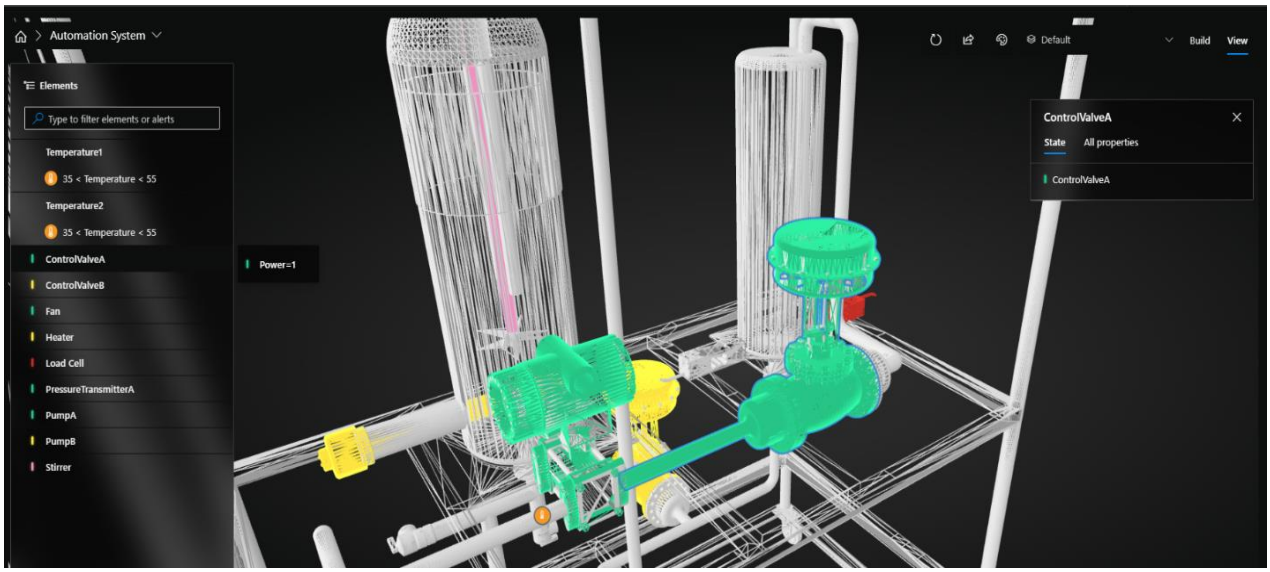


Figure 128:Control Valve A

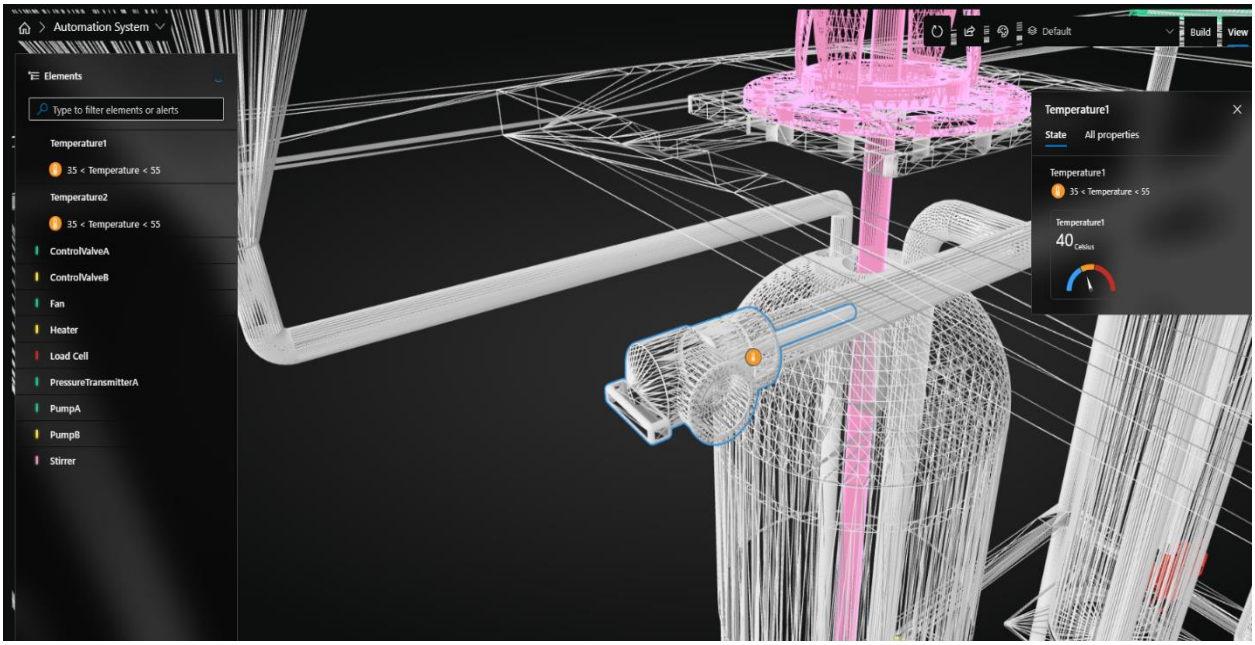


Figure 129:Temperature 2 Sensor

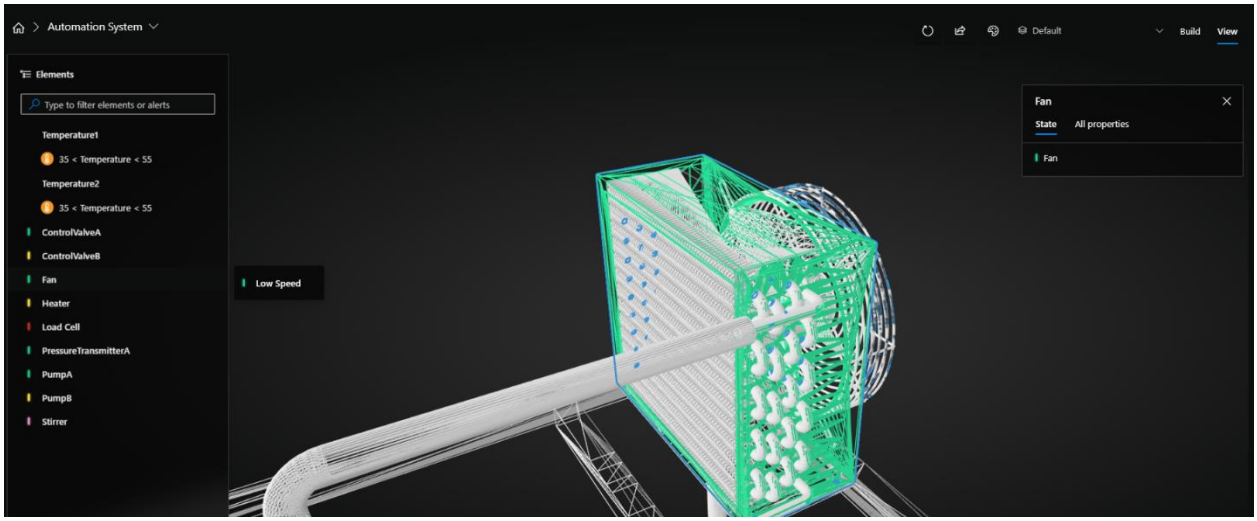


Figure 130:Radiator Device

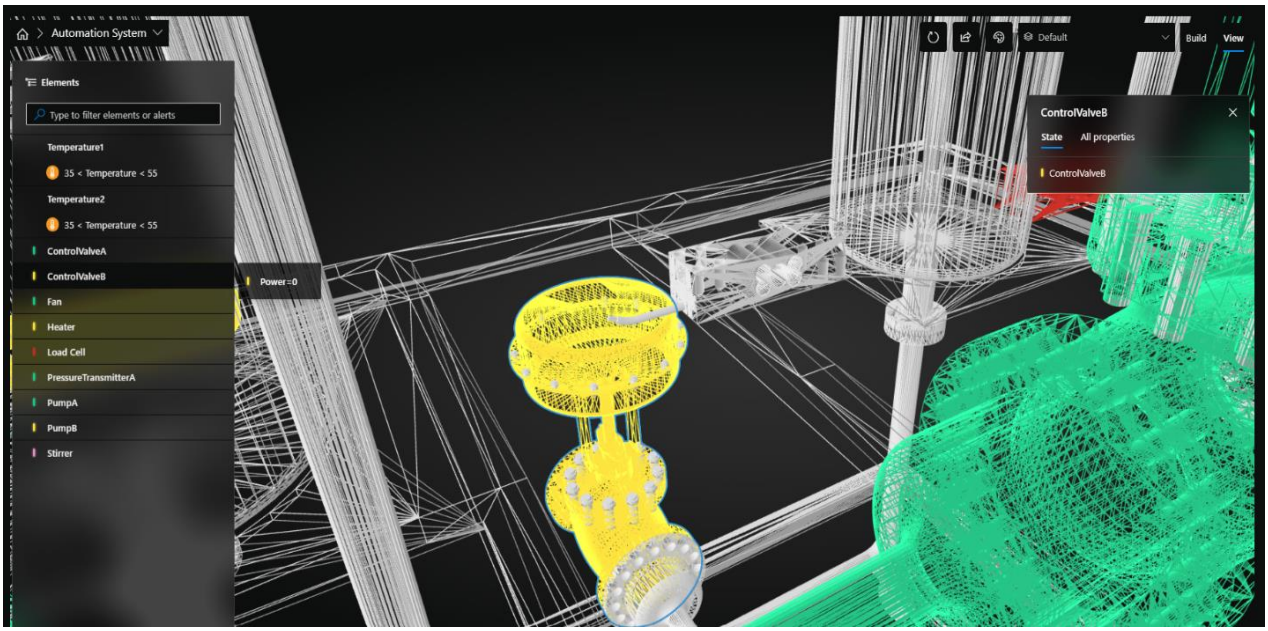


Figure 131:Control Valve B

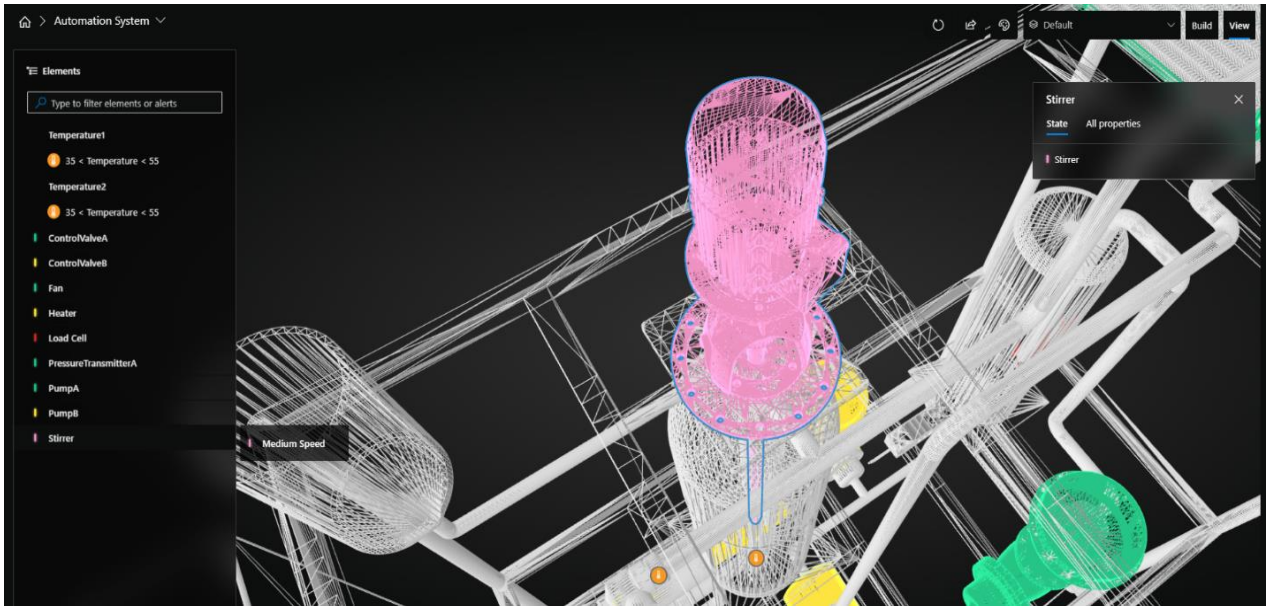


Figure 132:Stirrer

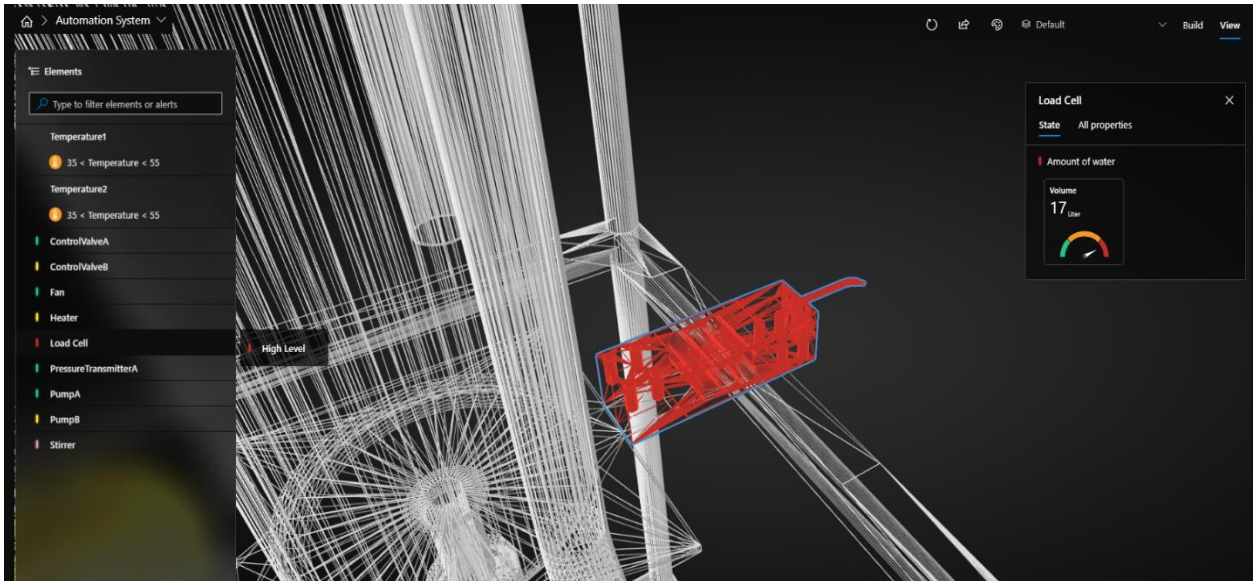


Figure 133:Load Cell

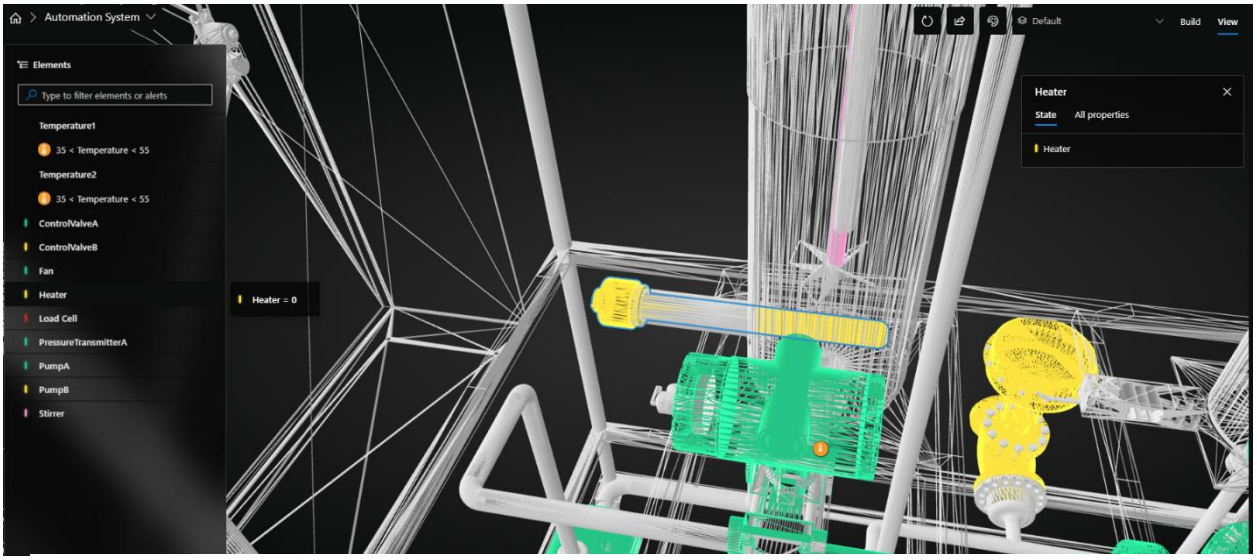


Figure 134:Heater.

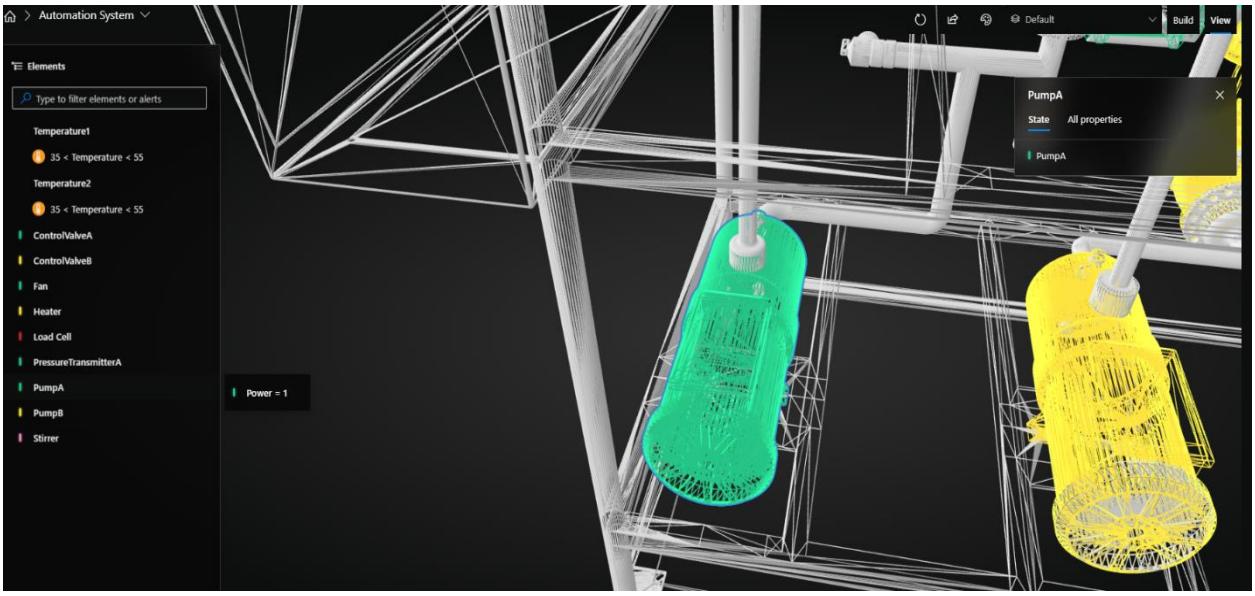


Figure 135:PumpA

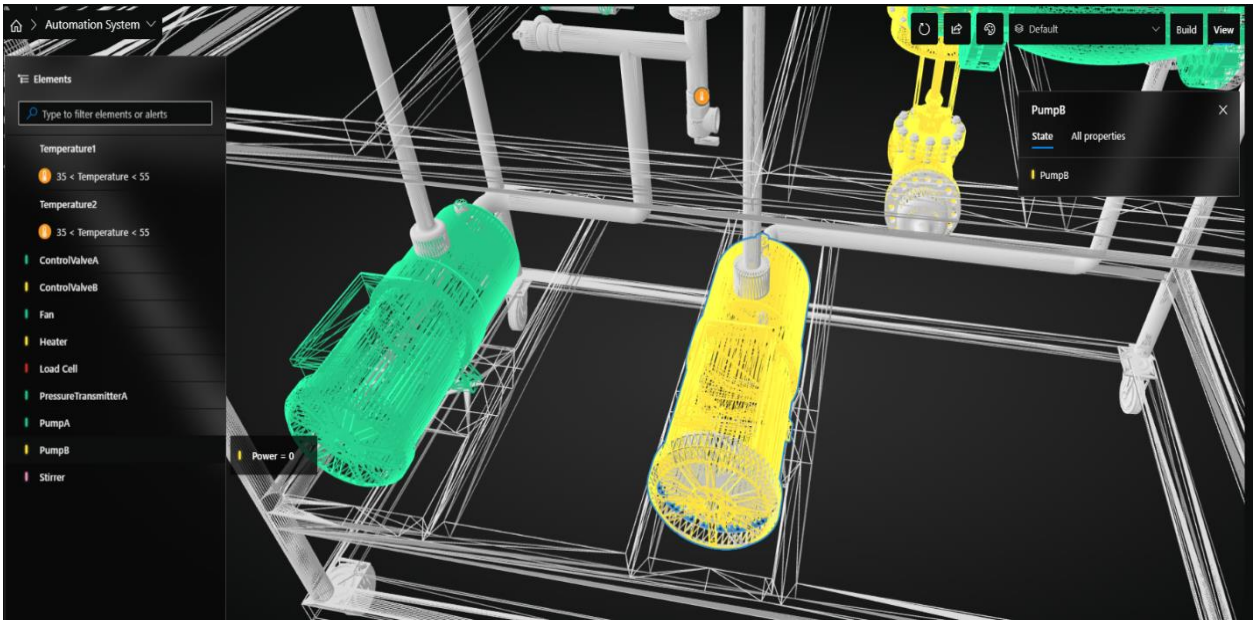


Figure 136: PumpB

8.2.2.5 Azure Data Explorer

Azure Data Explorer is a fully managed, high-performance, big data analytics platform that makes it easy to analyze high volumes of data in near real time. The Azure Data Explorer toolbox gives you an end-to-end solution for data ingestion, query, visualization, and management.

Figure 137:Azure Date Explorer

By analyzing structured, semi-structured, and unstructured data across time series, and by using Machine Learning, Azure Data Explorer makes it simple to extract key insights, spot patterns and trends, and create forecasting models. Azure Data Explorer uses a traditional relational model, organizing data into tables with strongly typed schemas. Tables are stored within databases, and a cluster can manage multiple databases. Azure Data Explorer is scalable, secure, robust, and enterprise-ready, and is useful for log analytics, time series analytics, IoT, and general-purpose exploratory analytics.

Azure Data Explorer capabilities are extended by other services built on its query language: Kusto Query Language (KQL). These services include Azure Monitor logs, Application Insights, Time Series Insights, and Microsoft Defender for Endpoint.

1. Create a data history connection for Azure Digital Twins:

Data history is an Azure Digital Twins feature for automatically historizing graph updates to Azure Data Explorer. This data can be queried using the Azure Digital Twins query plugin for Azure Data Explorer to gain insights about your environment over time.

1. Create a Kusto (Azure Data Explorer) cluster and database:

create a Kusto (Azure Data Explorer) cluster and database to receive the data from Azure Digital Twins.

Use the following CLI commands to create the required resources. The commands use several local variables ($location, $resource group, $cluster name, and $database name) that were created earlier in setting up local variables for the CLI session.

Start by adding the Kusto extension to your CLI session if you don't have it already.

```
az extension add --name kusto
```

Next, create the Kusto cluster. The command below requires 5-10 minutes to execute and will create an E2a v4 cluster in the developer tier. This type of cluster has a single node for the engine and data-management cluster and is applicable for development and test scenarios. For more information about the tiers in Azure Data Explorer and

how to select the right options for your production workload, see Select the correct compute SKU for your Azure Data Explorer cluster and Azure Data Explorer Pricing.

```
az kusto cluster create --cluster-name $clustername --sku name="Dev(No SLA)_Standard_E2a_v4" tier="Basic" --resource-group $resourcegroup --location $location --type SystemAssigned
```

Create a database in your new Kusto cluster (using the cluster name from above and in the same location). This database will be used to store contextualized Azure Digital Twins data. The command below creates a database with a soft delete period of 365 days, and a hot cache period of 31 days. For more information about the options available for this command, see az Kusto database create.

```
az kusto database create --cluster-name $clustername --database-name $databasename --resource-group $resourcegroup --read-write-database soft-delete-period=P365D hot-cache-period=P31D location=$location
```

2. Set up data history connection:

Now that you've created the required resources, use the command in this section to create a data history connection between the Azure Digital Twins instance, the event hub, and the Azure Data Explorer cluster.

This command will also create three tables in your Azure Data Explorer database to store twin property updates, twin lifecycle events, and relationship lifecycle events. For more information about these types of historized data and their corresponding Azure Data Explorer tables, see Data Types and Schemas.

Start by navigating to your Azure Digital Twins instance in the Azure portal (you can find the instance by entering its name into the portal search bar). Then complete the following steps.

1. Select Data history from the Connect Outputs section of the instance's menu.

Select Create a Connection. Doing so will begin the process of creating a data history connection

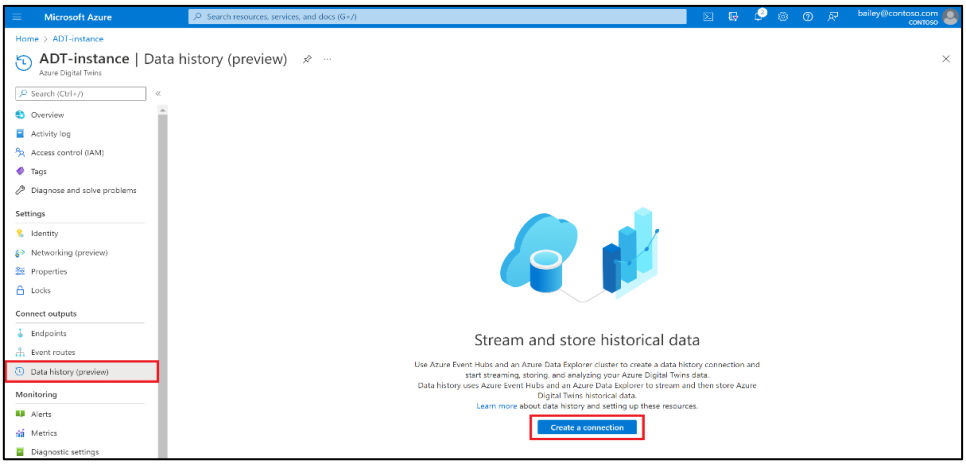


Figure 138:Create a Connection in ADT_Instance

2. **(SOME USERS) If you don't already have a managed identity enabled for your Azure Digital Twins instance, you'll see this page first, asking you to turn on Identity for the instance as the first step for the data history connection.**

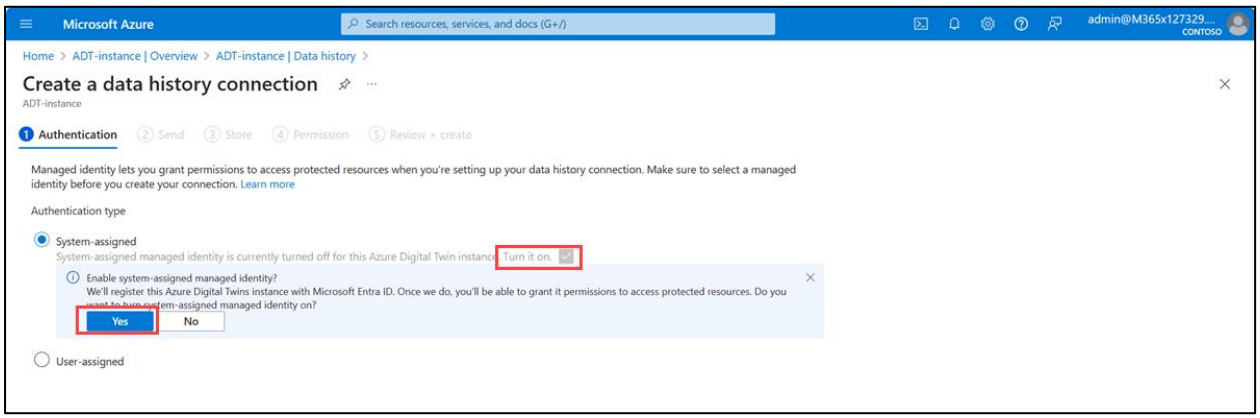


Figure 139:Authentication of Data History

If you do already have a managed identity enabled, your setup will skip this step and you'll see the next page immediately.

3. On the Send page, enter the details of the Event Hubs resources that you created earlier. In my account, the Subscription is “Azure for Students” , m2aonamespace, and m2ao_eventhub

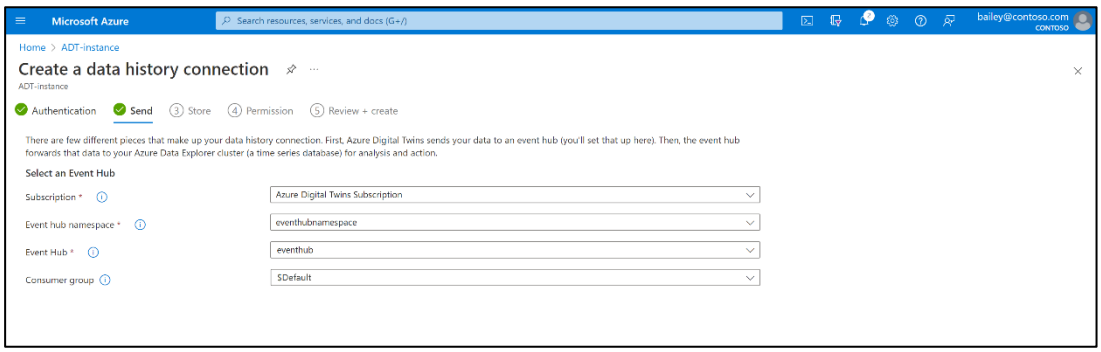


Figure 140:Connect Data History to Event Hub

Select Next.

5. **On the Store page, enter the details of the Azure Data Explorer resources that you created earlier. You can choose a custom name for the table that will store twin property updates or leave it blank to use the default table name of *AdtPropertyEvents*, and you can choose whether twin property deletions (events that remove properties entirely) should be included with the historized data. If you want to historize twin lifecycles and relationship lifecycle events, enter custom names for these tables.**

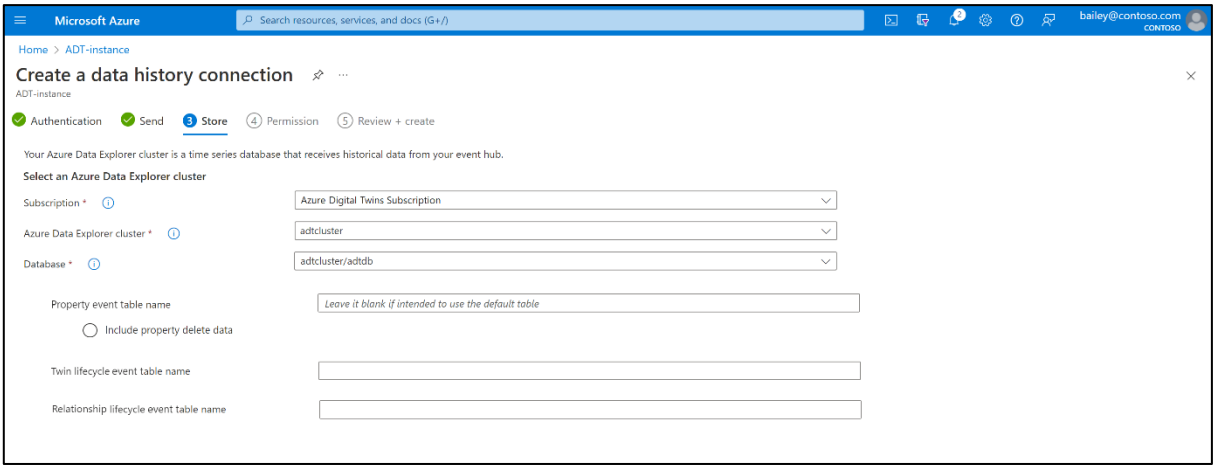


Figure 141:Store Data In DB

Select Next.

6. On the Permission page, select all the checkboxes to give your Azure Digital Twins instance permission to connect to the Event Hubs and Azure Data Explorer resources. If you already have equal or higher permissions in place, you can skip this step

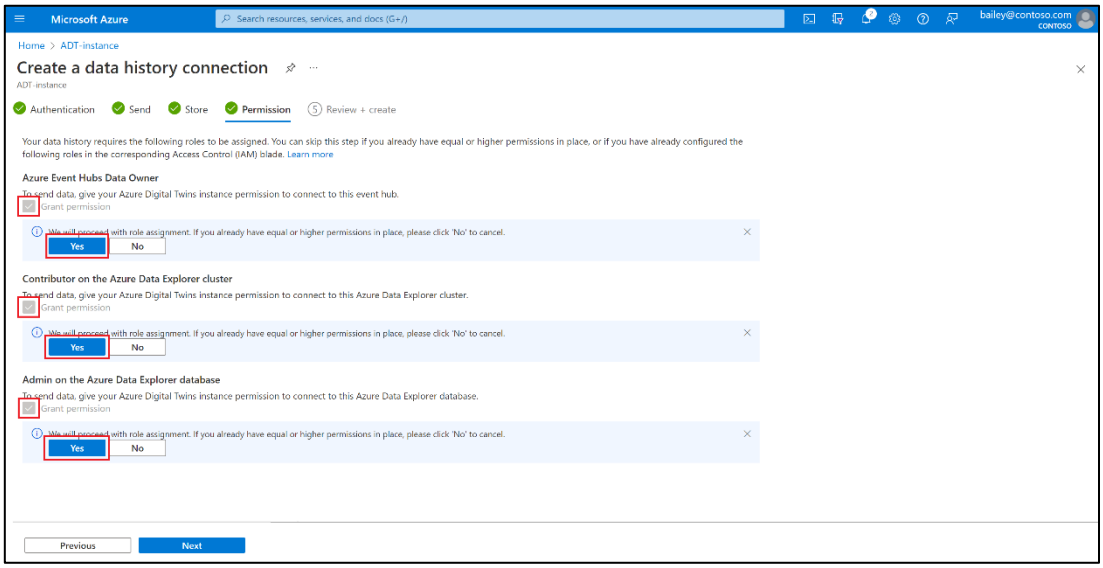


Figure 142:Permission of Each Service

Select Next.

7. On the Review + create page, review the details of your resources and select Create Connection.

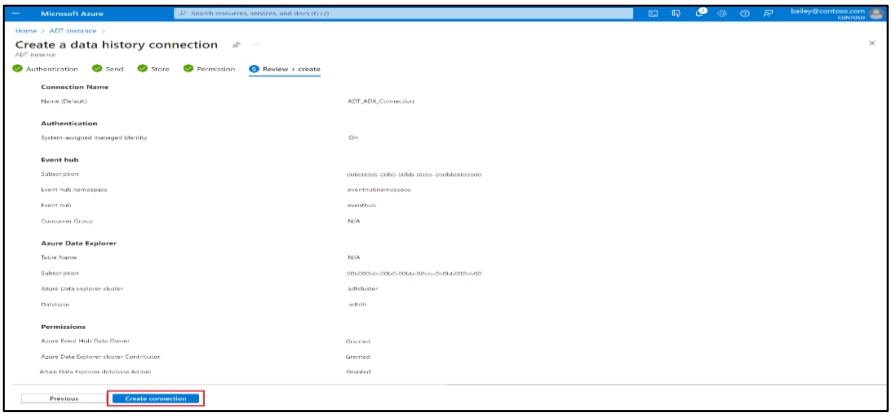


Figure 143:Review + Create Page

When the connection is finished creating, you'll be taken back to the Data history page for the Azure Digital Twins instance, which now shows details of the data history connection you've created, I will show the following image from my account.

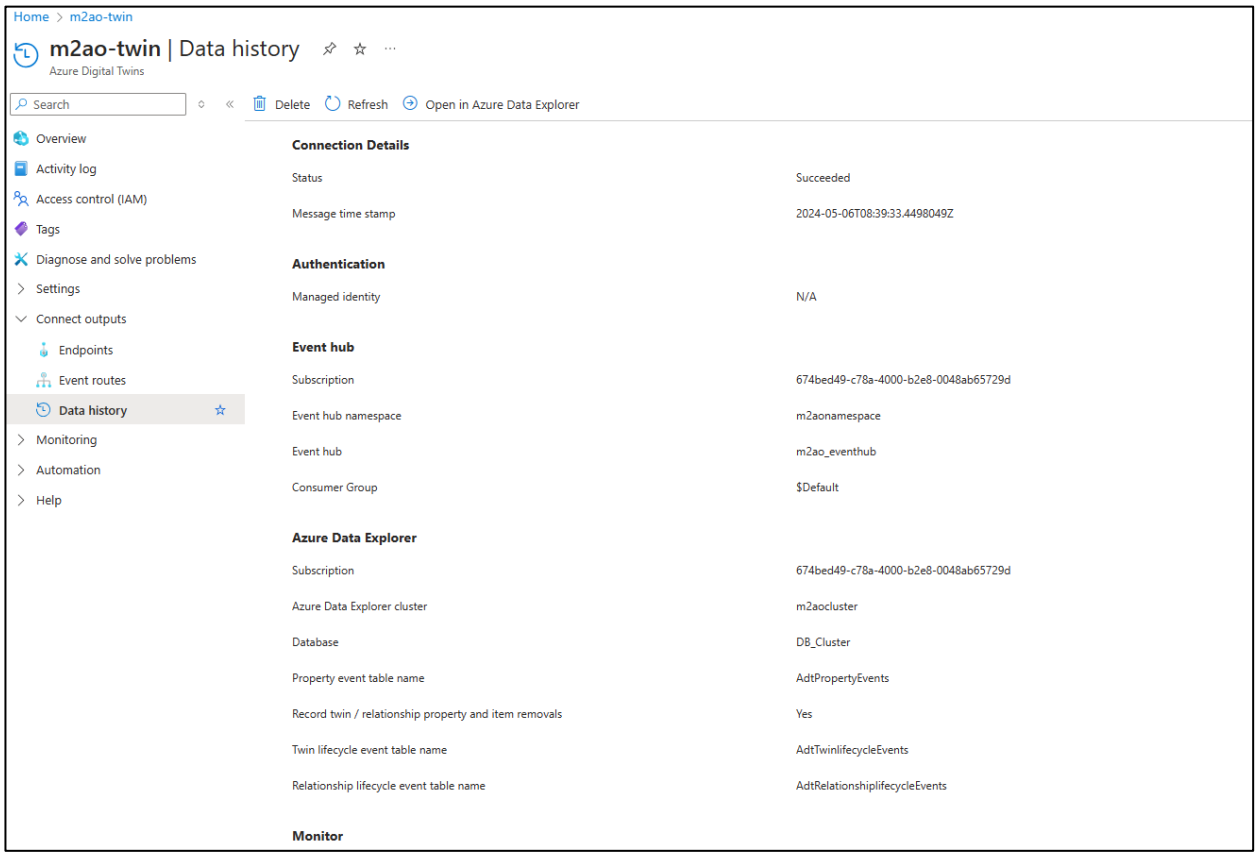


Figure 144:Data History Overview

3. View the historized updates in Azure Data Explorer:

This section will show you how to view all three types of historized updates that were generated by the simulator and stored in Azure Data Explorer tables. Start in the Azure portal and navigate to the Azure Data Explorer cluster you created earlier. Choose the Databases pane from the left menu to open the database view. Find the database you created for this article and select the checkbox next to it, then select Query.

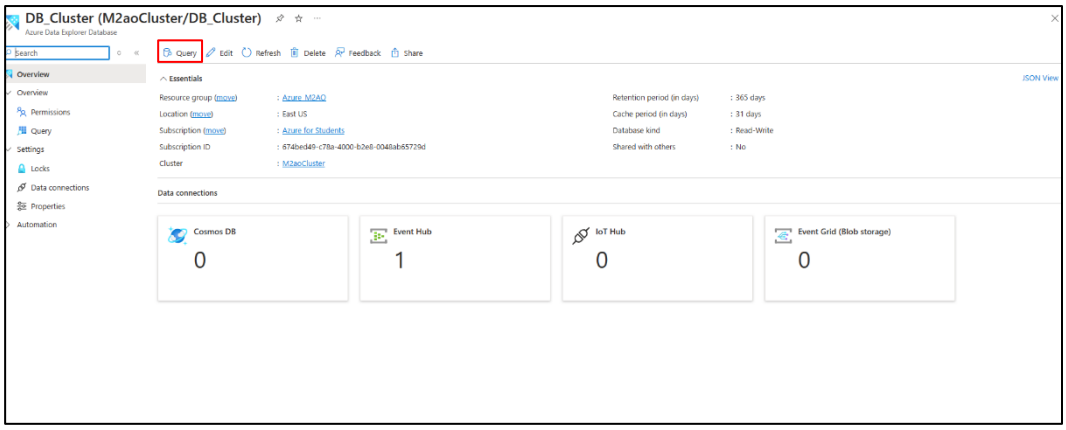


Figure 145:Query Button

Next, expand the cluster and database in the left pane to see the name of the data history tables. There should be three: one for relationship lifecycle events, one for twin lifecycle events, and one for twin property update events. You'll use these table names to run queries on the tables to verify and view the historized data.

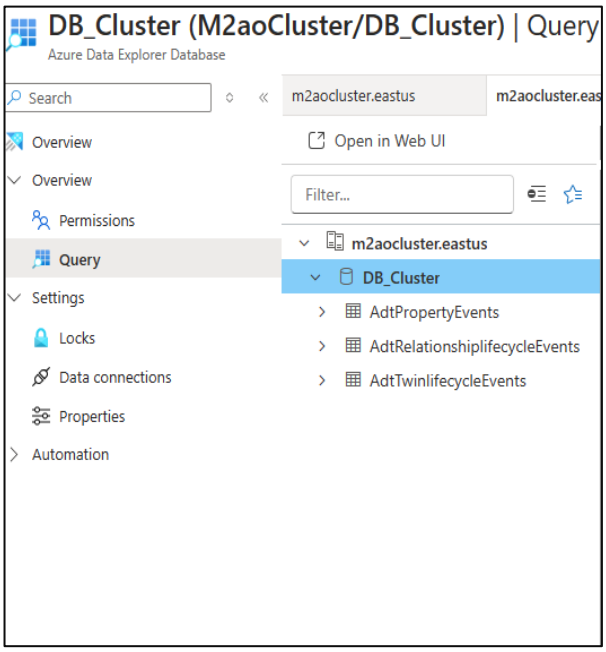


Figure 146:Tables in DB Cluster

Now Azure Data Explorer Cluster can enable us to conduct queries for business insights. It's possible to investigate any table within our cluster's database to examine sensor readings from a designated digital twin across various historical points. Moreover, we can execute calculations on the system's data for analysis. Beyond this, I'll present several scenarios that can be performed with real-time data.

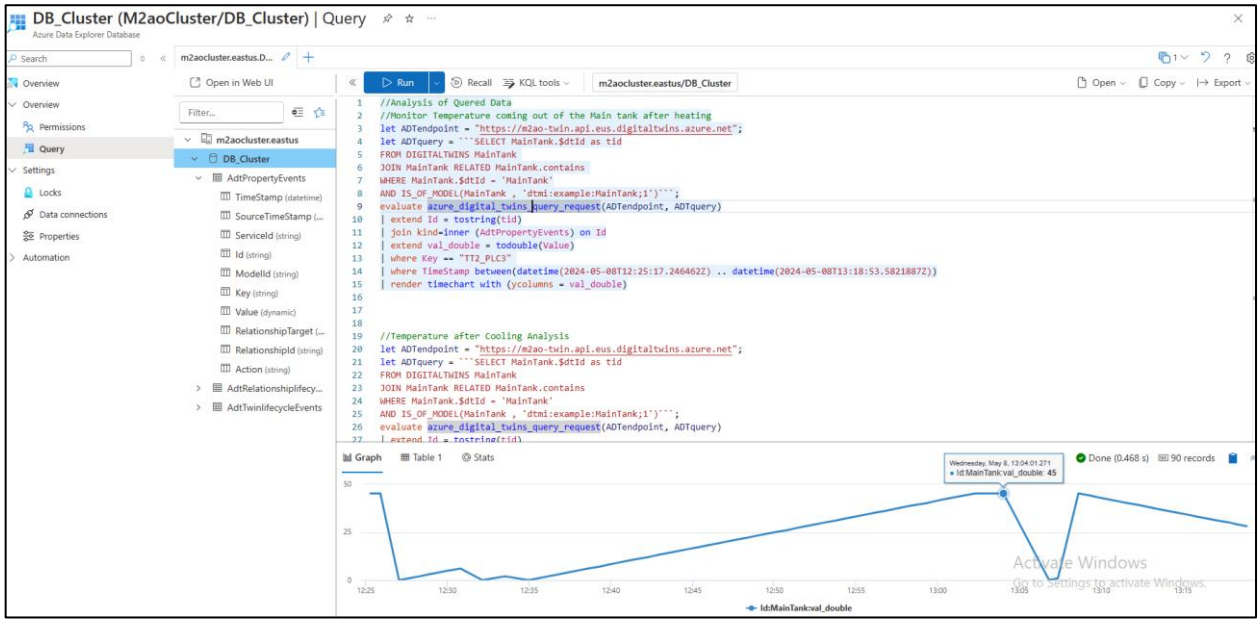
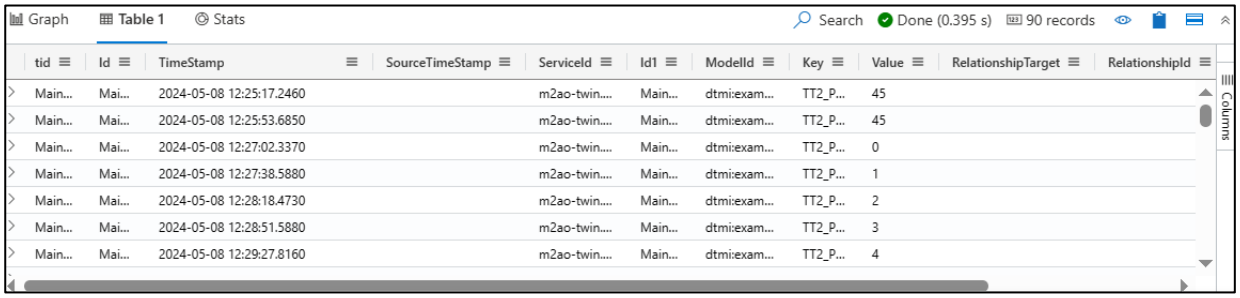


Figure 147:Query on Temperature2 Sensor



tid	Id	TimeStamp	SourceTimeStamp	ServiceId	Id1	ModelId	Key	Value	RelationshipTarget	RelationshipId
Main...	Mai...	2024-05-08 12:25:17.2460		m2ao-twin....	Main...	dtmi:exam...	TT2_P...	45		
Main...	Mai...	2024-05-08 12:25:53.6850		m2ao-twin....	Main...	dtmi:exam...	TT2_P...	45		
Main...	Mai...	2024-05-08 12:27:02.3370		m2ao-twin....	Main...	dtmi:exam...	TT2_P...	0		
Main...	Mai...	2024-05-08 12:27:38.5880		m2ao-twin....	Main...	dtmi:exam...	TT2_P...	1		
Main...	Mai...	2024-05-08 12:28:18.4730		m2ao-twin....	Main...	dtmi:exam...	TT2_P...	2		
Main...	Mai...	2024-05-08 12:28:51.5880		m2ao-twin....	Main...	dtmi:exam...	TT2_P...	3		
Main...	Mai...	2024-05-08 12:29:27.8160		m2ao-twin....	Main...	dtmi:exam...	TT2_P...	4		

Figure 148:Historized Data of Temperature1 Sensor

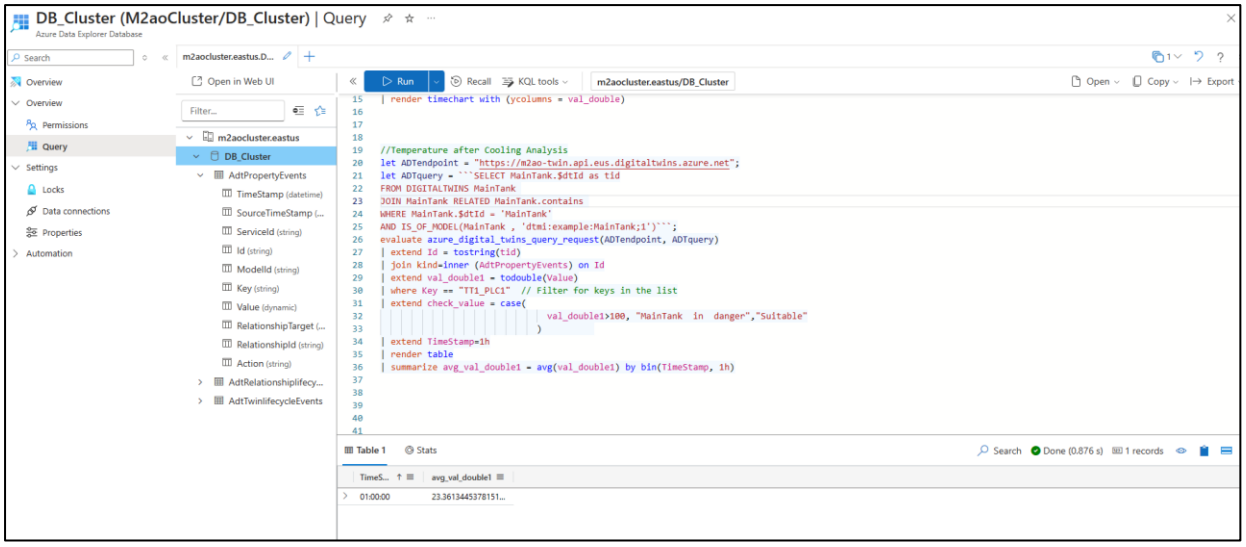


Figure 150:Query on Temperature1 Sensor

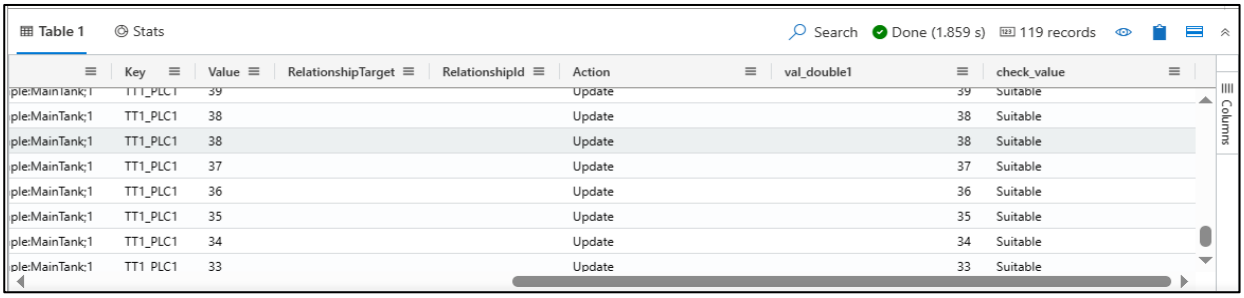


Figure 149:Query Status in Table of Temperature1 Sensor

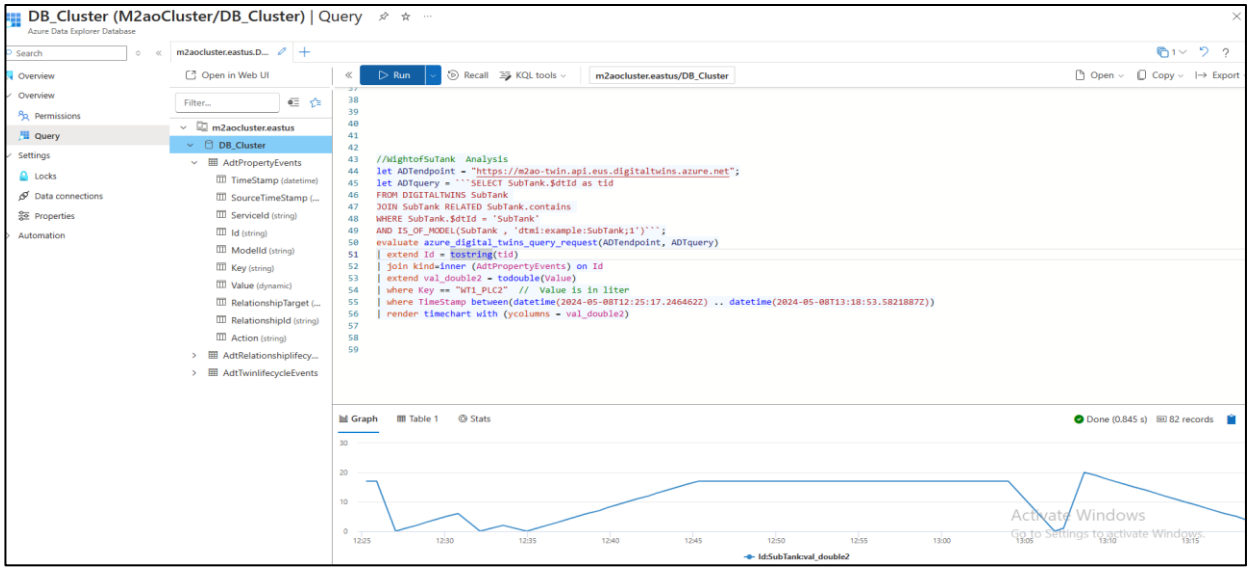


Figure 151:Query on Amount of Water

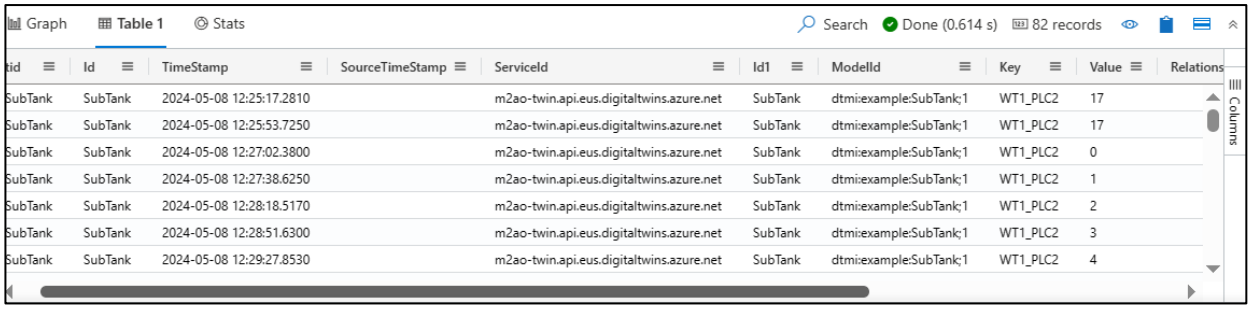


Figure 152:Table of Wights of Sub tank

Having completed these stages, it's evident that we've made significant progress towards embodying the principles of the Fourth Industrial Revolution, encompassing both the Industrial Internet of Things (IIoT) and Digital Twin technology. However, it's important to recognize that to fully realize our ambitious project, additional components are necessary. One such critical element is the implementation of predictive maintenance within our system. This can be achieved by a seamless integration of Azure Data Explorer, Azure Machine Learning, and Azure Digital Twin, among others.

9. Conclusion

This graduation project has successfully demonstrated the integration and implementation of several advanced technologies and methodologies that align with the principles of Industry 4.0. Through the design and deployment of a comprehensive IIoT system, we have created a robust platform that encompasses real-time data acquisition, processing, and visualization.

Key Achievements:

1. PLC and Protocols Integration: We developed a highly efficient PLC system integrated with various communication protocols, ensuring reliable data exchange between different components of the system.

2. IoT Platform Utilization: By leveraging Ubidots and Azure platforms, we established a seamless interface for monitoring and controlling the system remotely. This integration included the use of APIs and SDKs to facilitate device communication.

3. Synthetic Variables and Time-Series Data: The project utilized synthetic variables to enhance the data analysis capabilities, coupled with a two-year time-series backend for extensive data storage and historical analysis.

4. Digital Twin Implementation: We explored the concept of Digital Twins, using the Azure Cloud Platform to create a digital replica of the physical system. This involved the construction of 3D scenes and the implementation of behaviors to simulate real-world conditions.

5. Data Historization and Analysis: Azure Data Explorer was employed to create data history connections, allowing for comprehensive analysis and insights derived from historical data.

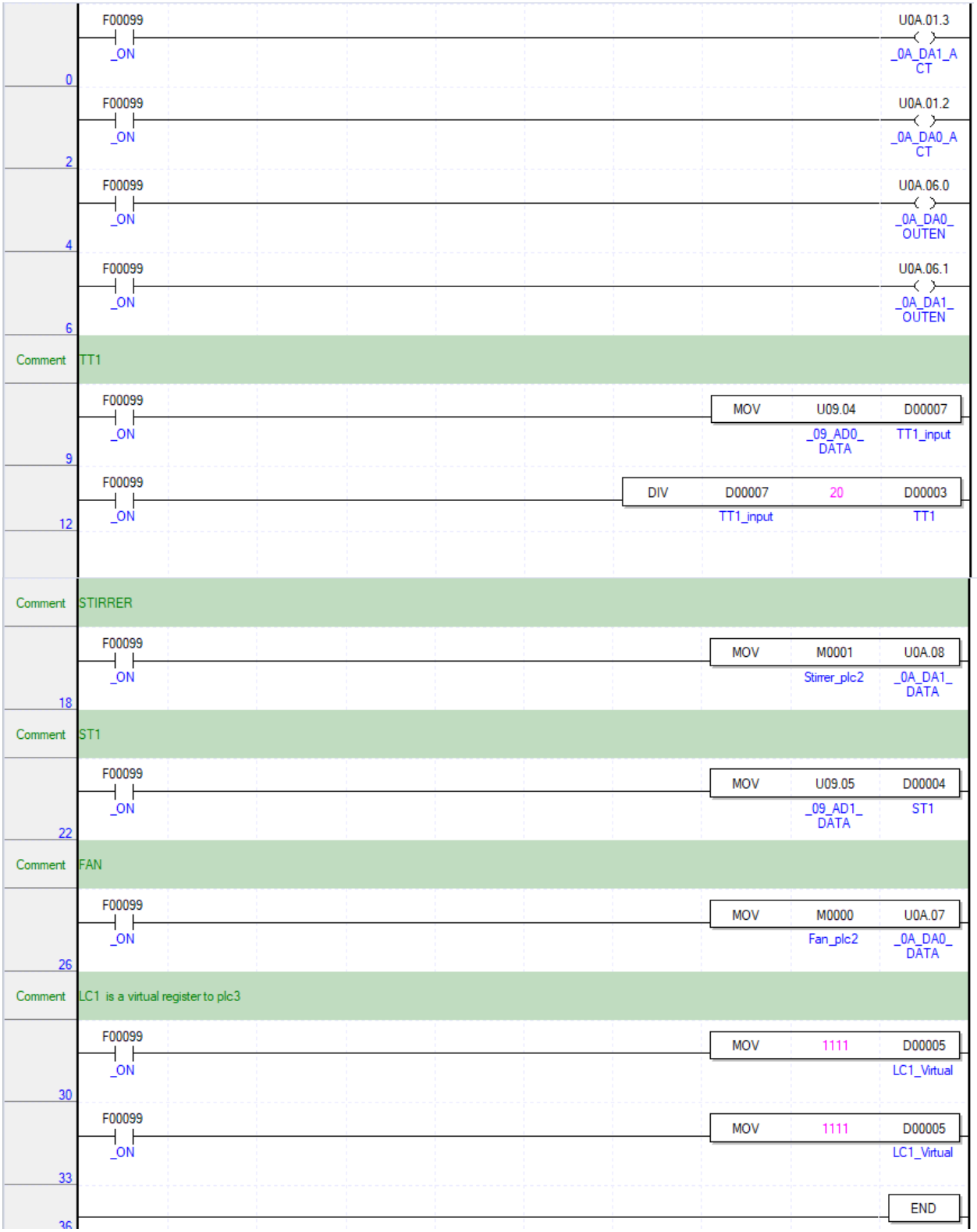
Future Directions:

To fully harness the potential of this system, further enhancements could include the implementation of predictive maintenance algorithms. Integrating machine learning models with Azure Machine Learning can enable the prediction of system failures and optimize maintenance schedules, thus reducing downtime and operational costs.

Overall, this project not only illustrates the technical feasibility of integrating IIoT and Digital Twin technologies but also sets a foundation for future innovations and applications in the realm of smart manufacturing and industrial automation.

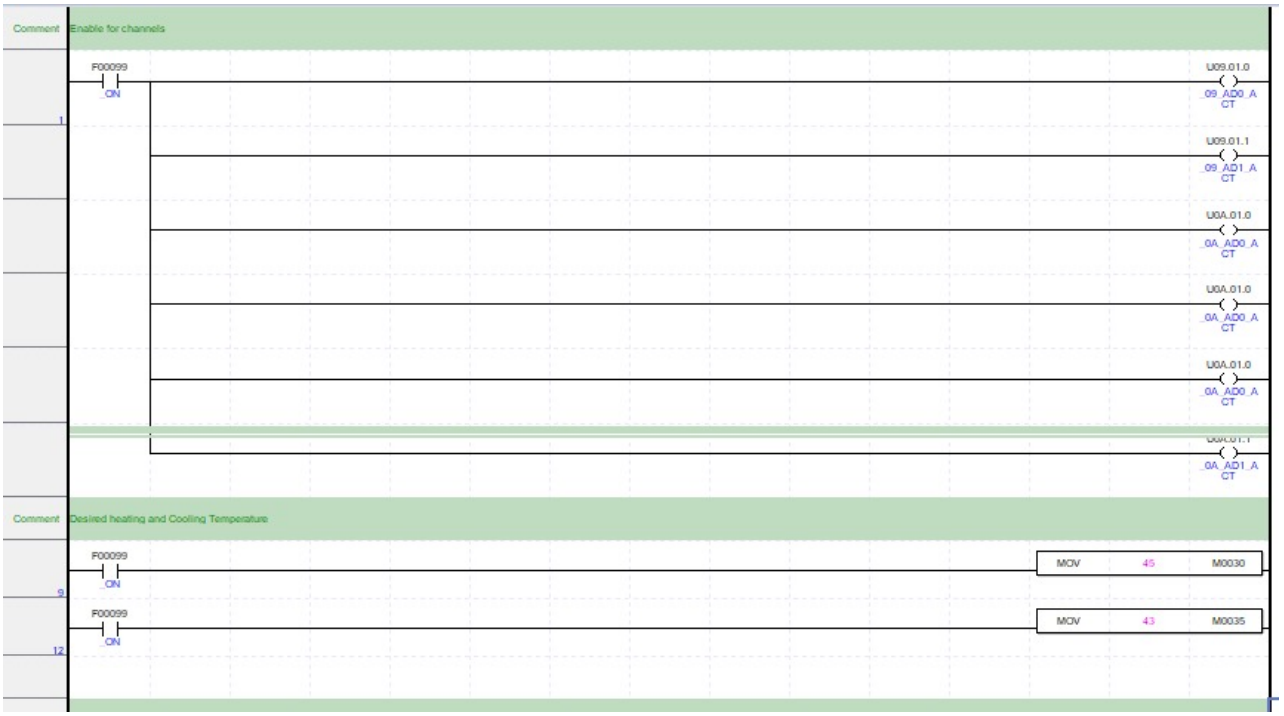
Appendix A

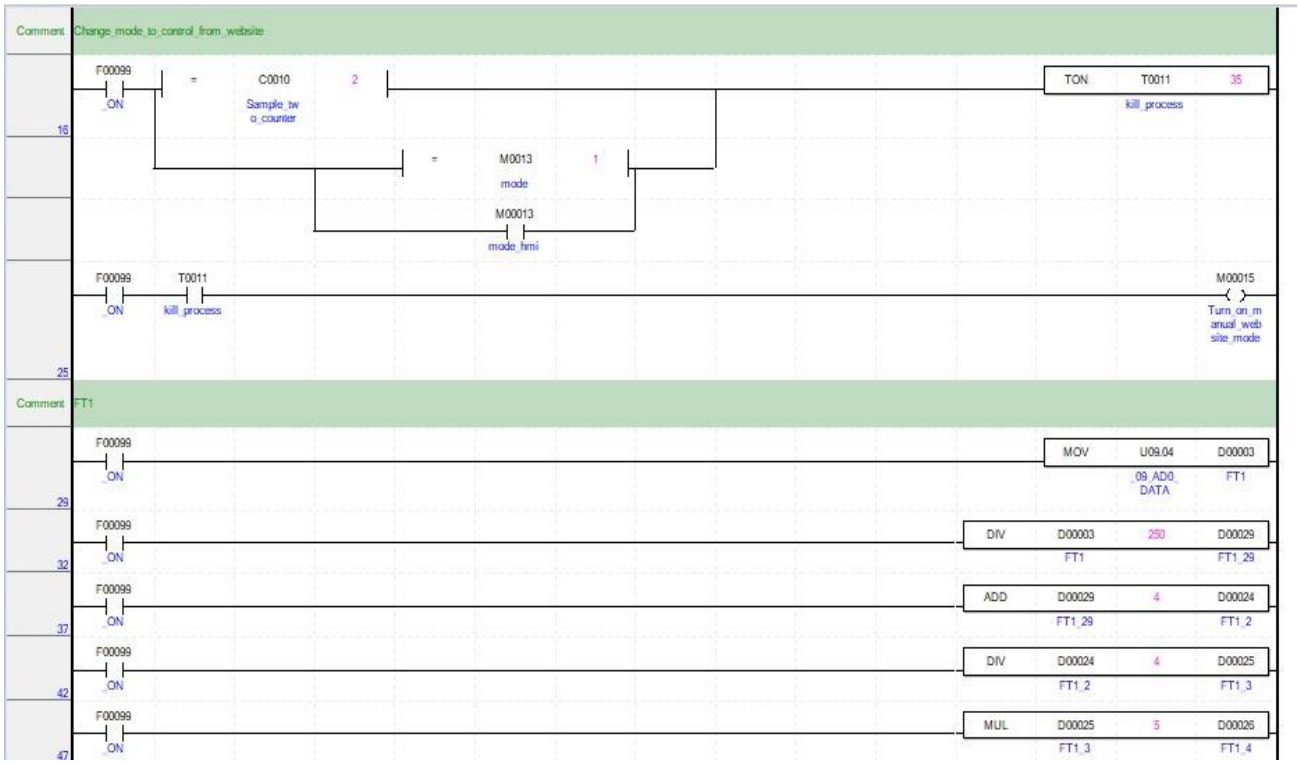
- **PIC1**

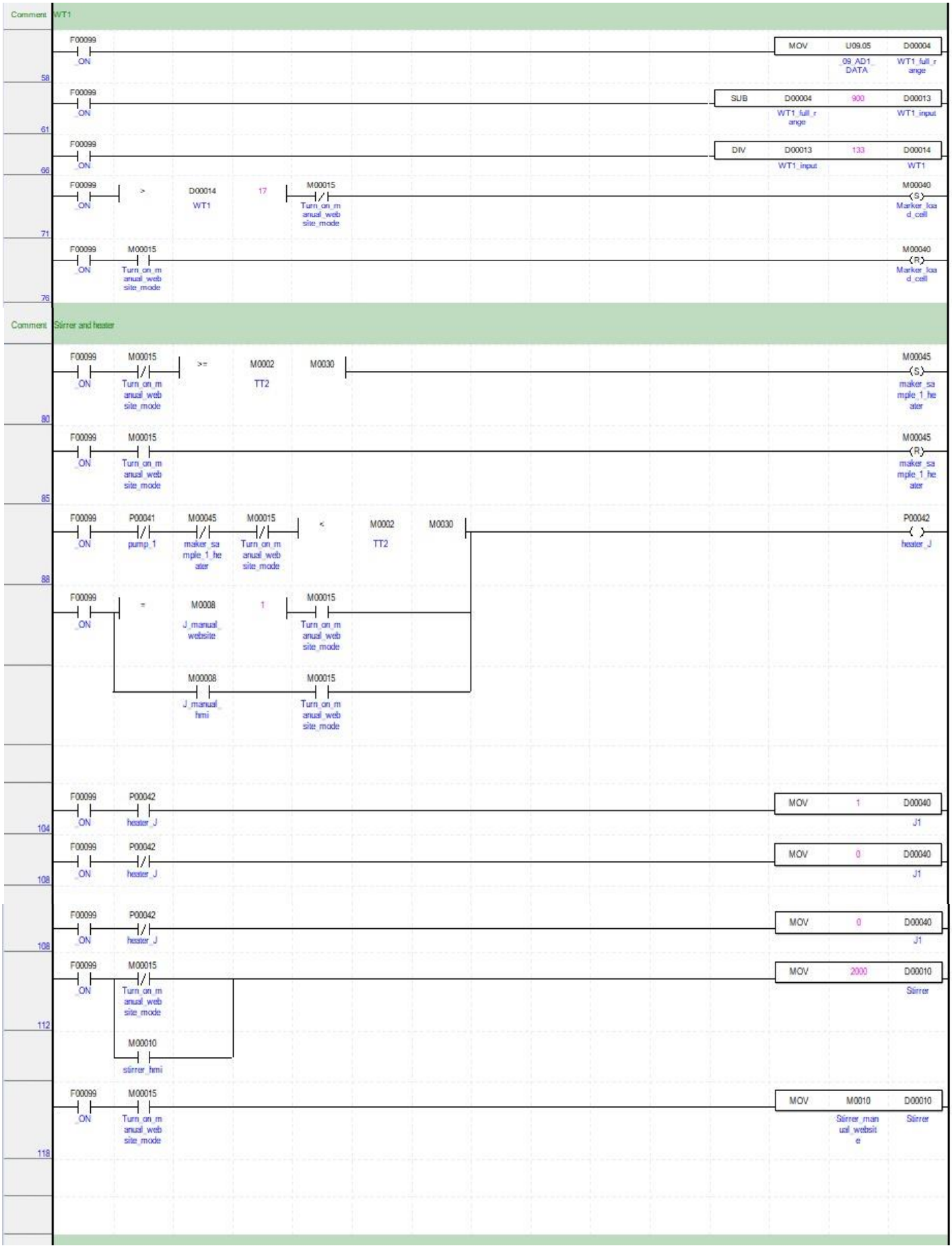


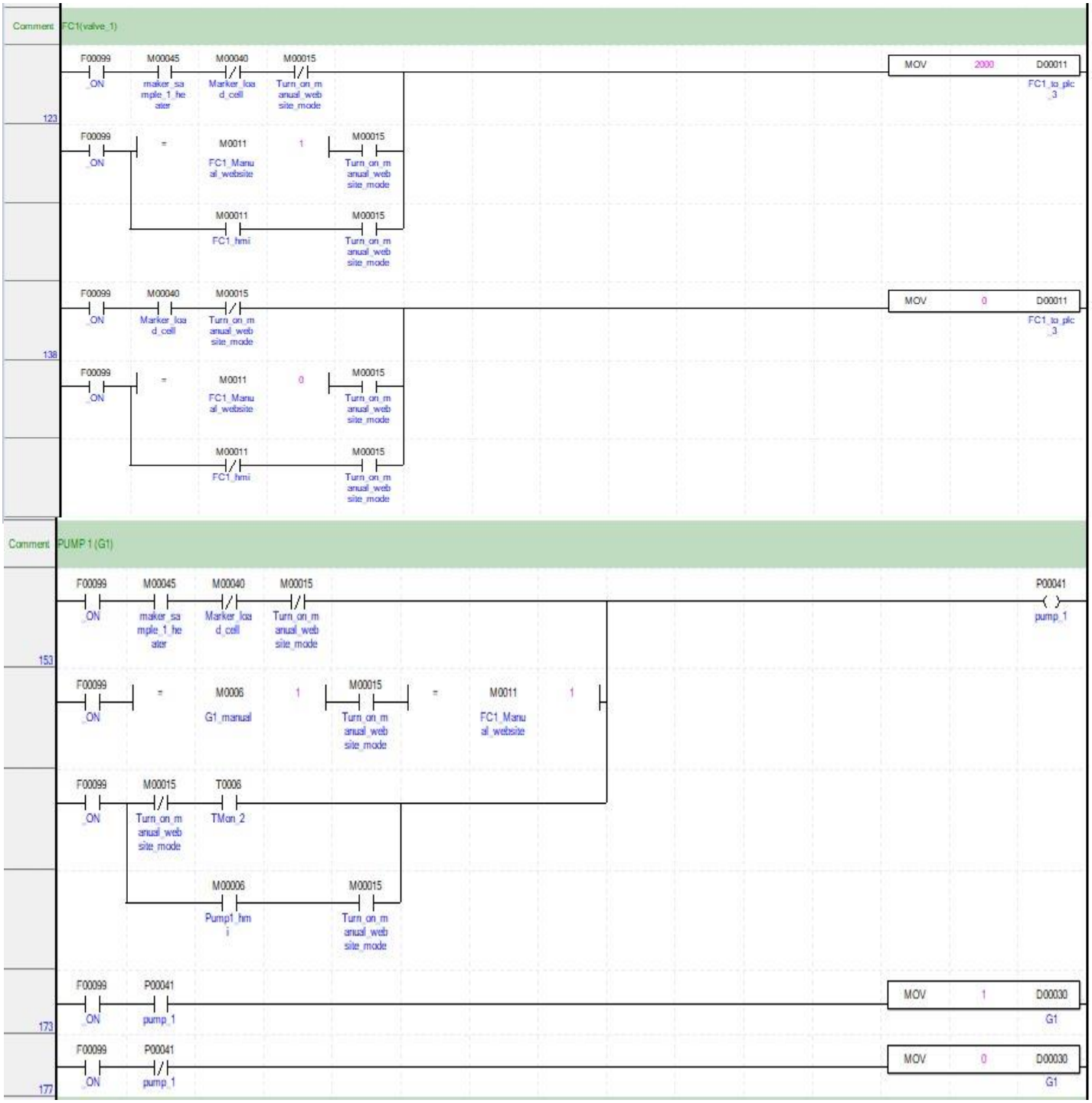
	Variable	Type	Device	Used	HMI	Comment
1	fan	WORD	D00002	☐	☐	fan
2	Fan_plc2	WORD	M0000	☑	☐	Fan is received from plc 2
3	J1	BIT	K00003	☐	☐	
4	LC1_Virtual	WORD	D00005	☑	☐	LC1 is written to PIC3
5	ST1	WORD	D00004	☑	☐	analog input ST1
6	ST1_mapping	WORD	D00010	☐	☐	
7	stirrer	WORD	D00001	☐	☐	stirrer
8	Stirrer_plc2	WORD	M0001	☑	☐	Stirrer is received from plc 2
9	TT1	WORD	D00003	☑	☐	TT1
10	TT1_input	WORD	D00007	☑	☐	TT2
11	_09_AD0_ACT	BIT	U09.01.0	☐	☐	Analog Input Option Board: AD0 Activation Status
12	_09_AD0_DATA	WORD	U09.04	☑	☐	Analog Input Option Board: AD0 Digital Output Data
13	_09_AD0_ERR	BIT	U09.01.8	☐	☐	Analog Input Option Board: AD0 Error Code
14	_09_AD0_IDD	BIT	U09.01.4	☐	☐	Analog Input Option Board: CH0 Input Disconnection F
15	_09_AD1_ACT	BIT	U09.01.1	☐	☐	Analog Input Option Board: AD1 Activation Status
16	_09_AD1_DATA	WORD	U09.05	☑	☐	Analog Input Option Board: AD1 Digital Output Data
17	_09_AD1_ERR	BIT	U09.01.9	☐	☐	Analog Input Option Board: AD1 Error Code
18	_09_AD1_IDD	BIT	U09.01.5	☐	☐	Analog Input Option Board: CH1 Input Disconnection F
19	_09_ERR	BIT	U09.00.0	☐	☐	Analog Input Option Board: Error
20	_09_RDY	BIT	U09.00.F	☐	☐	Analog Input Option Board: Ready
21	_0A_DA0_ACT	BIT	U0A.01.2	☑	☐	Analog Output Option Board: DA0 Activation Status
22	_0A_DA0_DATA	WORD	U0A.07	☑	☐	Analog Output Option Board: DA0 Digital Input Data
23	_0A_DA0_ERR	BIT	U0A.01.A	☐	☐	Analog Output Option Board: DA0 Error Code
24	_0A_DA0_OUTEN	BIT	U0A.06.0	☑	☐	Analog Output Option Board: DA0 Output Enable
25	_0A_DA1_ACT	BIT	U0A.01.3	☑	☐	Analog Output Option Board: Output CH1 Activation St
26	_0A_DA1_DATA	WORD	U0A.08	☑	☐	Analog Output Option Board: DA1 Digital Input Data
27	_0A_DA1_ERR	BIT	U0A.01.B	☐	☐	Analog Output Option Board: DA1 Error Code
28	_0A_DA1_OUTEN	BIT	U0A.06.1	☑	☐	Analog Output Option Board: DA1 Output Enable
29	_0A_ERR	BIT	U0A.00.0	☐	☐	Analog Output Option Board: Error
30	_0A_OUTEN	WORD	U0A.06	☐	☐	Analog Output Option Board: Output Enable
31	_0A_RDY	BIT	U0A.00.F	☐	☐	Analog Output Option Board: Ready

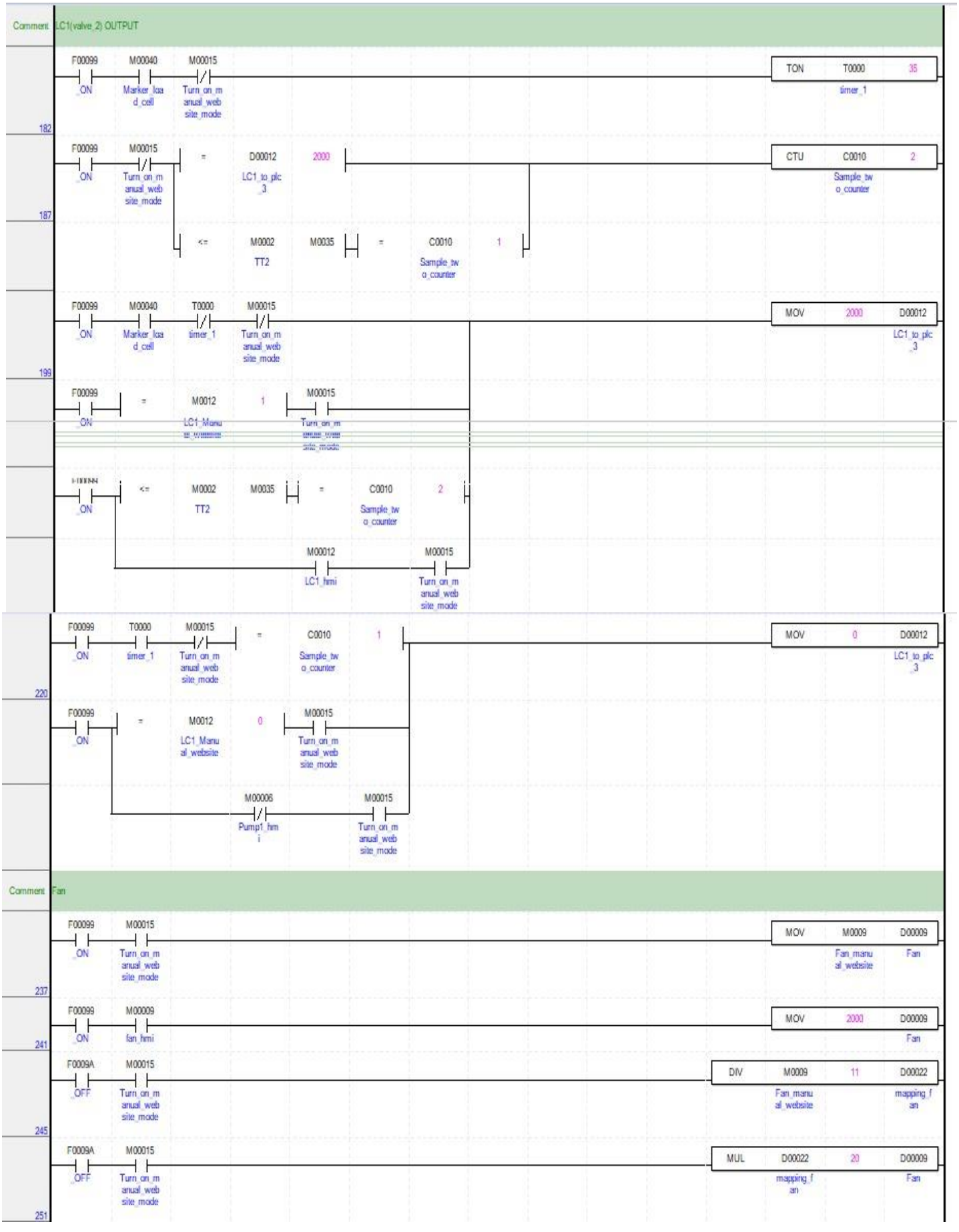
- **PIC2**

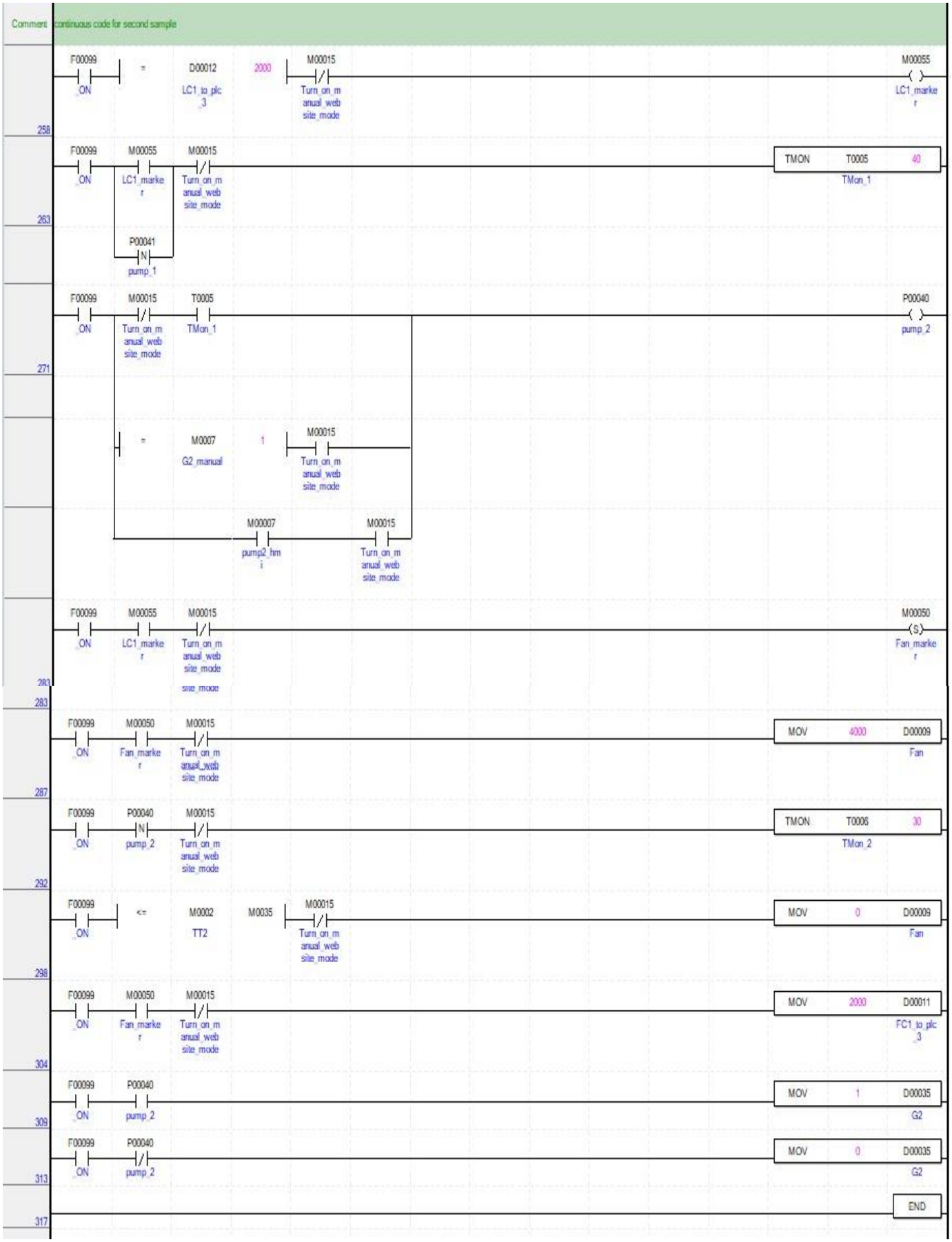

Comment
Enable for channels
F00099
_ON
U09.01.0
_09_AD0_ACT
U09.01.1
_09_AD1_ACT
U0A.01.0
_0A_AD0_ACT
U0A.01.0
_0A_AD0_ACT
U0A.01.0
_0A_AD0_ACT
_0A_AD1_ACT
Comment
Desired heating and Cooling Temperature
F00099
_ON
MOV
45
M0030
F00099
_ON
MOV
43
M0035


Comment
Change_mode_to_control_from_website
F00099
_ON
=
C0010
2
Sample_tw
o_counter
TON
T0011
35
kill_process
=
M0013
1
mode
M00013
mode_hmi
F00099
_ON
T0011
kill_process
M00015
Turn_on_m
anual_web
site_mode
Comment
FT1
F00099
_ON
MOV
U09.04
_09_AD0_
DATA
D00003
FT1
F00099
_ON
DIV
D00003
FT1
250
D00029
FT1_29
F00099
_ON
ADD
D00029
FT1_29
4
D00024
FT1_2
F00099
_ON
DIV
D00024
FT1_2
4
D00025
FT1_3
F00099
_ON
MUL
D00025
FT1_3
5
D00026
FT1_4

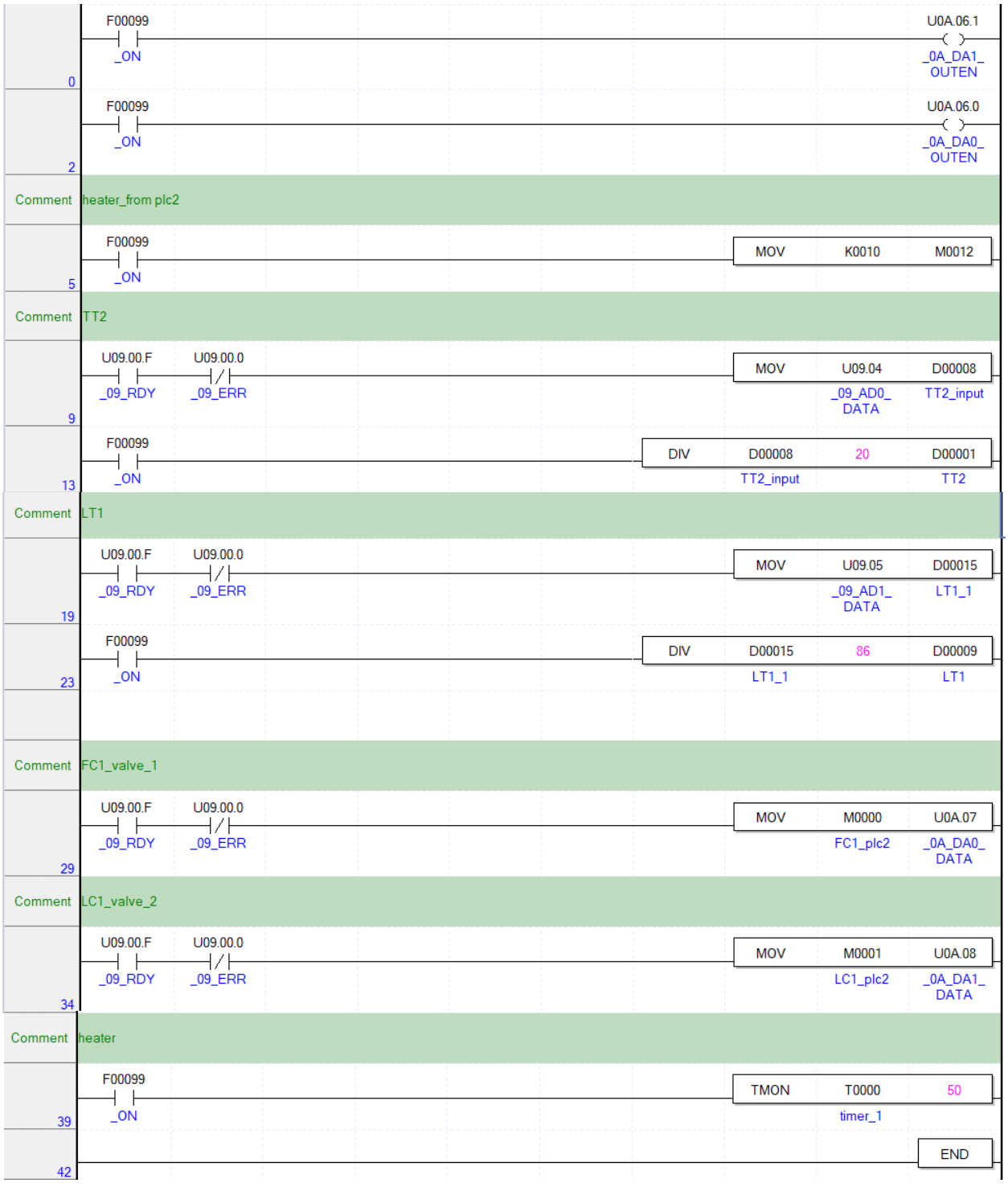

Comment WT1
F00099 _ON
MOV U09.05 _09_AD1_DATA D00004 WT1_full_range
SUB D00004 WT1_full_range 900 D00013 WT1_input
DIV D00013 WT1_input 133 D00014 WT1
> D00014 WT1 17 M00015 Turn_on_manual_web site_mode
M00040 (S) Marker_load_cell
M00040 (R) Marker_load_cell
Comment Stirrer and heater
>= M0002 TT2 M0030
M00045 (S) maker_sample_1_heater
M00045 (R) maker_sample_1_heater
P00041 pump_1
M00045 maker_sample_1_heater
< M0002 TT2 M0030
P00042 ( ) heater_J
= M0008 J_manual_website 1
M00008 J_manual_hmi
MOV 1 D00040 J1
MOV 0 D00040 J1
MOV 2000 D00010 Stirrer
M00010 stirrer_hmi
MOV M0010 Stirrer_manual_website D00010 Stirrer


Comment FC1(valve_1)
F00099 _ON
M00045 maker_sample_1_heater
M00040 Marker_load_cell
M00015 Turn_on_manual_website_mode
MOV 2000 D00011 FC1_to_plc_3
123
F00099 _ON
=
M0011 FC1_Manual_website
1
M00015 Turn_on_manual_website_mode
M00011 FC1_hmi
M00015 Turn_on_manual_website_mode
F00099 _ON
M00040 Marker_load_cell
M00015 Turn_on_manual_website_mode
MOV 0 D00011 FC1_to_plc_3
138
F00099 _ON
=
M0011 FC1_Manual_website
0
M00015 Turn_on_manual_website_mode
M00011 FC1_hmi
M00015 Turn_on_manual_website_mode
Comment PUMP 1 (G1)
F00099 _ON
M00045 maker_sample_1_heater
M00040 Marker_load_cell
M00015 Turn_on_manual_website_mode
P00041 pump_1
153
F00099 _ON
=
M0006 G1_manual
1
M00015 Turn_on_manual_website_mode
=
M0011 FC1_Manual_website
1
F00099 _ON
M00015 Turn_on_manual_website_mode
T0006 TMon_2
M00006 Pump1_hmi
M00015 Turn_on_manual_website_mode
F00099 _ON
P00041 pump_1
MOV 1 D00030 G1
173
F00099 _ON
P00041 pump_1
MOV 0 D00030 G1
177


Comment LC1(valve_2) OUTPUT
182
F00099 _ON
M00040 Marker_load_cell
M00015 Turn_on_manual_website_mode
TON T0000 35 timer_1
187
F00099 _ON
M00015 Turn_on_manual_website_mode
= D00012 LC1_to_plc_3 2000
CTU C0010 2 Sample_two_counter
<= M0002 TT2 M0035 = C0010 Sample_two_counter 1
199
F00099 _ON
M00040 Marker_load_cell
T0000 timer_1
M00015 Turn_on_manual_website_mode
MOV 2000 D00012 LC1_to_plc_3
F00099 _ON
= M0012 LC1_Manual_website
1 M00015 Turn_on_manual_website_mode
<= M0002 TT2 M0035 = C0010 Sample_two_counter 2
M00012 LC1_hmi
M00015 Turn_on_manual_website_mode
220
F00099 _ON
T0000 timer_1
M00015 Turn_on_manual_website_mode
= C0010 Sample_two_counter 1
MOV 0 D00012 LC1_to_plc_3
F00099 _ON
= M0012 LC1_Manual_website
0 M00015 Turn_on_manual_website_mode
M00006 Pump1_hmi
M00015 Turn_on_manual_website_mode
Comment Fan
237
F00099 _ON
M00015 Turn_on_manual_website_mode
MOV M0009 Fan_manual_website D00009 Fan
241
F00099 _ON
M00009 fan_hmi
MOV 2000 D00009 Fan
245
F0009A _OFF
M00015 Turn_on_manual_website_mode
DIV M0009 Fan_manual_website 11 D00022 mapping_fan
251
F0009A _OFF
M00015 Turn_on_manual_website_mode
MUL D00022 mapping_fan 20 D00009 Fan


Comment
continuous code for second sample
258
F00099 _ON
=
D00012 LC1_to_plc_3
2000
M00015 Turn_on_manual_web site_mode
M00055 LC1_marker
263
F00099 _ON
M00055 LC1_marker
M00015 Turn_on_manual_web site_mode
P00041 pump_1
TMON T0005 40 TMon_1
271
F00099 _ON
M00015 Turn_on_manual_web site_mode
T0005 TMon_1
P00040 pump_2
=
M0007 G2_manual
1
M00015 Turn_on_manual_web site_mode
M00007 pump2_fmi
M00015 Turn_on_manual_web site_mode
283
F00099 _ON
M00055 LC1_marker
M00015 Turn_on_manual_web site_mode
M00050 Fan_marker
287
F00099 _ON
M00050 Fan_marker
M00015 Turn_on_manual_web site_mode
MOV 4000 D00009 Fan
292
F00099 _ON
P00040 pump_2
M00015 Turn_on_manual_web site_mode
TMON T0006 30 TMon_2
298
F00099 _ON
<=
M0002 TT2
M0035
M00015 Turn_on_manual_web site_mode
MOV 0 D00009 Fan
304
F00099 _ON
M00050 Fan_marker
M00015 Turn_on_manual_web site_mode
MOV 2000 D00011 FC1_to_plc_3
309
F00099 _ON
P00040 pump_2
MOV 1 D00035 G2
313
F00099 _ON
P00040 pump_2
MOV 0 D00035 G2
317
END

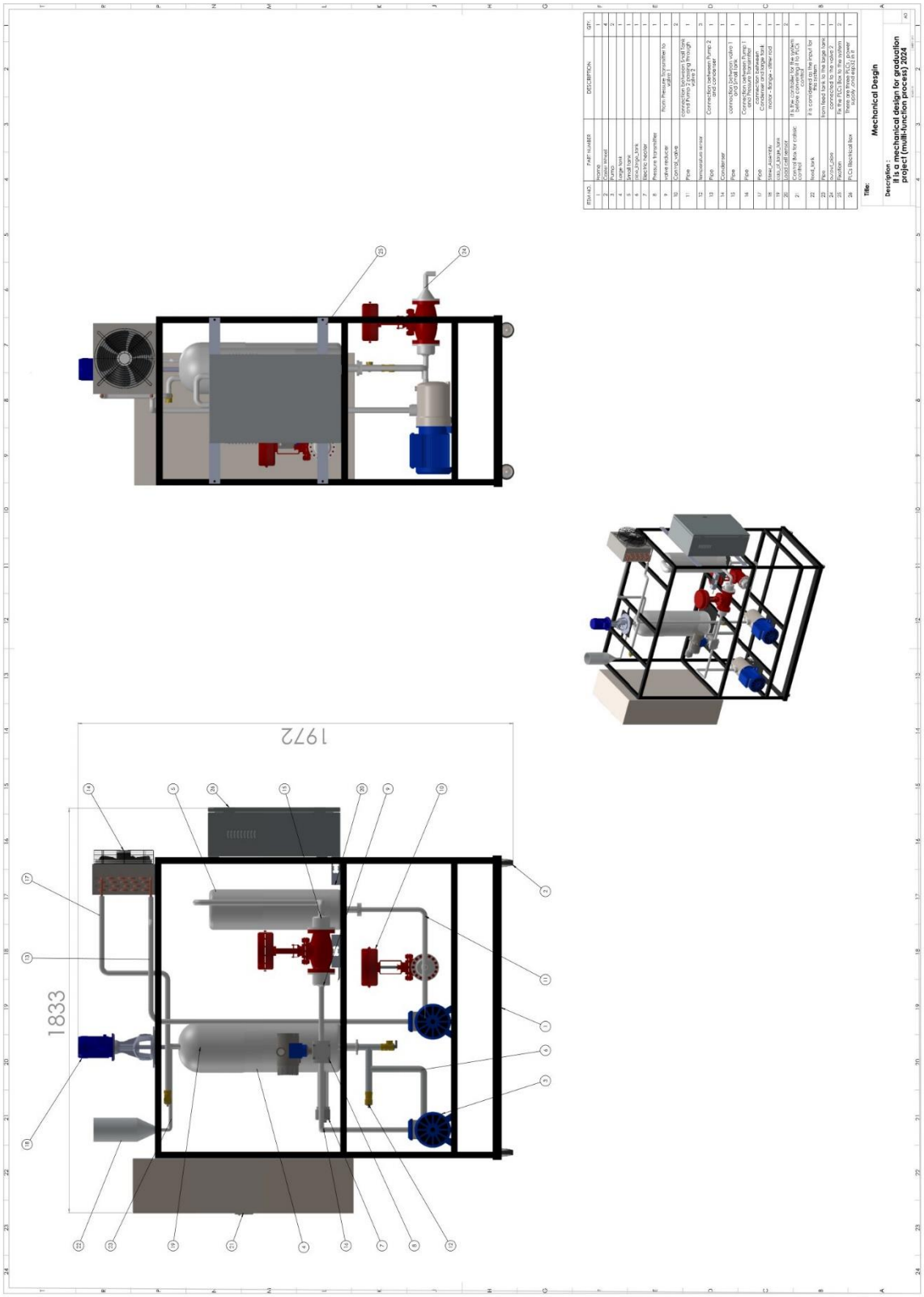
- **PIC3**

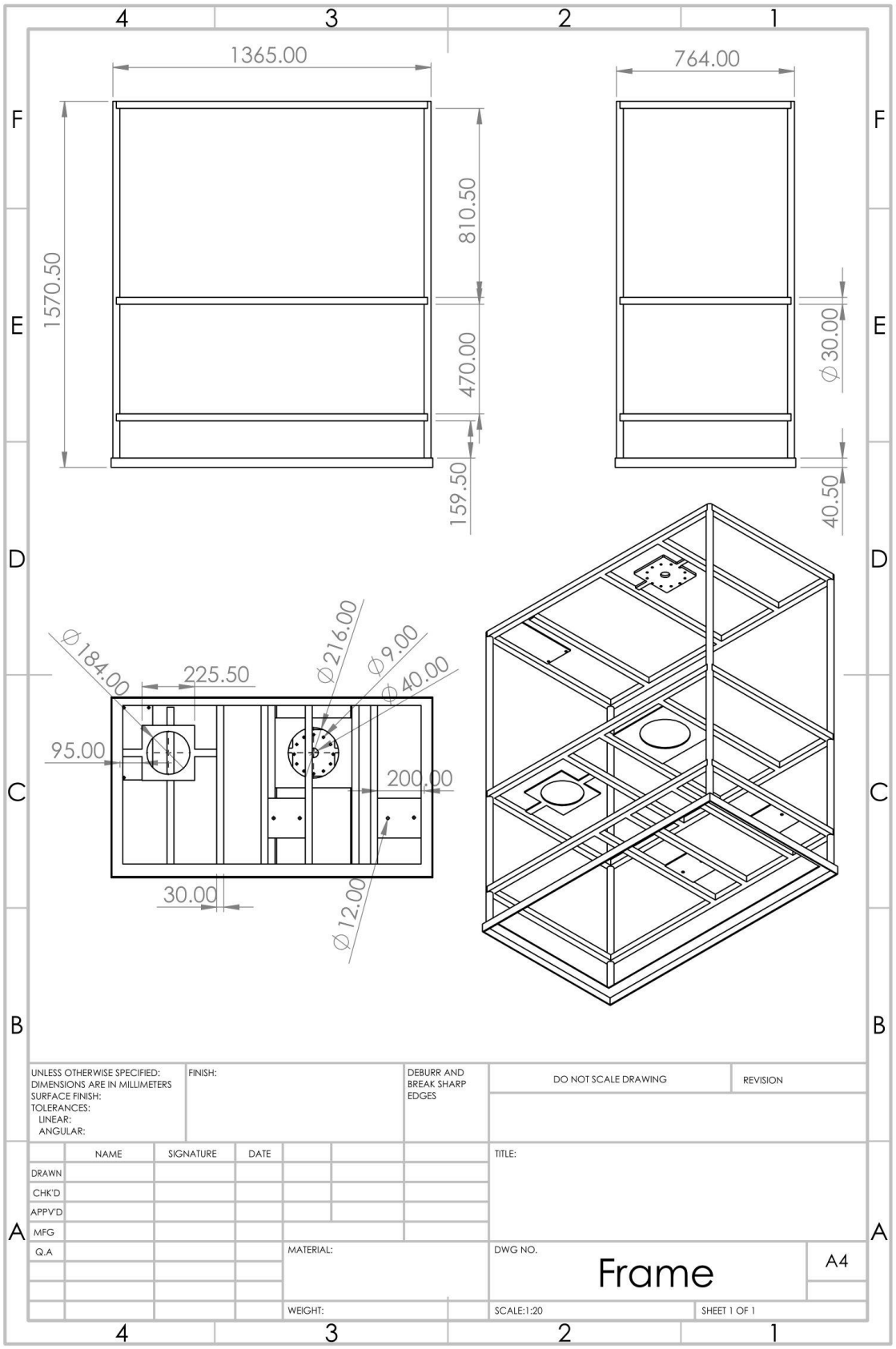

F00099
_ON
U0A.06.1
_0A_DA1_OUTEN
0
F00099
_ON
U0A.06.0
_0A_DA0_OUTEN
2
Comment heater_from plc2
F00099
_ON
MOV K0010 M0012
5
Comment TT2
U09.00.F
_09_RDY
U09.00.0
_09_ERR
MOV U09.04 D00008
_09_AD0_DATA
TT2_input
9
F00099
_ON
DIV D00008 20 D00001
TT2_input
TT2
13
Comment LT1
U09.00.F
_09_RDY
U09.00.0
_09_ERR
MOV U09.05 D00015
_09_AD1_DATA
LT1_1
19
F00099
_ON
DIV D00015 86 D00009
LT1_1
LT1
23
Comment FC1_valve_1
U09.00.F
_09_RDY
U09.00.0
_09_ERR
MOV M0000 U0A.07
FC1_plc2
_0A_DA0_DATA
29
Comment LC1_valve_2
U09.00.F
_09_RDY
U09.00.0
_09_ERR
MOV M0001 U0A.08
LC1_plc2
_0A_DA1_DATA
34
Comment heater
F00099
_ON
TMON T0000 50
timer_1
39
END
42

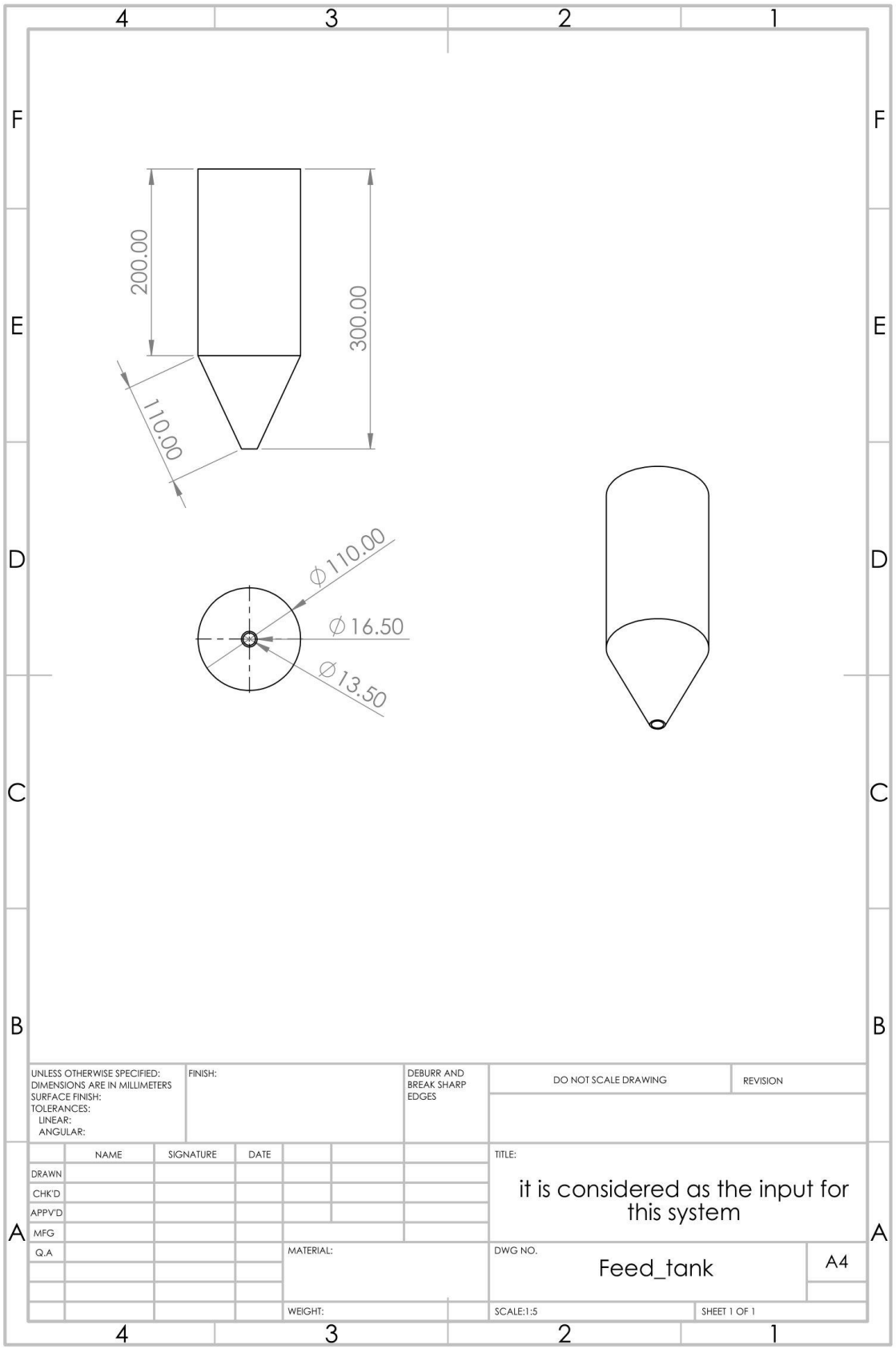
	Variable	Type	Device	Used	HMI	Comment
1	FC1_plc2	WORD	M0000	☑	☐	FC1 is received from PLC2
2	heater	BIT	P00040	☐	☐	
3	heater_lamp	BIT	P00043	☐	☐	
4	J1	BIT	K00000	☐	☐	
5	LC1_plc2	WORD	M0001	☑	☐	LC1 is received from PLC2
6	LC1_Virtual	WORD	M0002	☐	☐	LC1 is received from PLC 1
7	LT1	WORD	D00009	☑	☐	LT1
8	LT1_1	WORD	D00015	☑	☐	
9	LT1_2	WORD	D00016	☐	☐	
10	LT1_3	WORD	D00017	☐	☐	
11	LT1_4	WORD	D00018	☐	☐	
12	LT1_5	WORD	D00019	☐	☐	
13	LT1_6	WORD	D00020	☐	☐	
14	timer_1	BIT/WORD	T0000	☑	☐	
15	TT2	WORD	D00001	☑	☐	TT2
16	TT2_input	WORD	D00008	☑	☐	TT2_input
17	_09_AD0_ACT	BIT	U09.01.0	☐	☐	Analog Input Option Board: AD0 Activation Status
18	_09_AD0_DATA	WORD	U09.04	☑	☐	Analog Input Option Board: AD0 Digital Output Data
19	_09_AD0_ERR	BIT	U09.01.8	☐	☐	Analog Input Option Board: AD0 Error Code
20	_09_AD0_IDD	BIT	U09.01.4	☐	☐	Analog Input Option Board: CH0 Input Disconnection F
21	_09_AD1_ACT	BIT	U09.01.1	☐	☐	Analog Input Option Board: AD1 Activation Status
22	_09_AD1_DATA	WORD	U09.05	☑	☐	Analog Input Option Board: AD1 Digital Output Data
23	_09_AD1_ERR	BIT	U09.01.9	☐	☐	Analog Input Option Board: AD1 Error Code
24	_09_AD1_IDD	BIT	U09.01.5	☐	☐	Analog Input Option Board: CH1 Input Disconnection F
25	_09_ERR	BIT	U09.00.0	☑	☐	Analog Input Option Board: Error
26	_09_RDY	BIT	U09.00.F	☑	☐	Analog Input Option Board: Ready
27	_0A_DA0_ACT	BIT	U0A.01.2	☐	☐	Analog Output Option Board: DA0 Activation Status
28	_0A_DA0_DATA	WORD	U0A.07	☑	☐	Analog Output Option Board: DA0 Digital Input Data
29	_0A_DA0_ERR	BIT	U0A.01.A	☐	☐	Analog Output Option Board: DA0 Error Code
30	_0A_DA0_OUTEN	BIT	U0A.06.0	☑	☐	Analog Output Option Board: DA0 Output Enable
31	_0A_DA1_ACT	BIT	U0A.01.3	☐	☐	Analog Output Option Board: Output CH1 Activation St
32	_0A_DA1_DATA	WORD	U0A.08	☑	☐	Analog Output Option Board: DA1 Digital Input Data
33	_0A_DA1_ERR	BIT	U0A.01.B	☐	☐	Analog Output Option Board: DA1 Error Code
34	_0A_DA1_OUTEN	BIT	U0A.06.1	☑	☐	Analog Output Option Board: DA1 Output Enable
35	_0A_ERR	BIT	U0A.00.0	☐	☐	Analog Output Option Board: Error
36	_0A_OUTEN	WORD	U0A.06	☐	☐	Analog Output Option Board: Output Enable
37	_0A_RDY	BIT	U0A.00.F	☐	☐	Analog Output Option Board: Ready

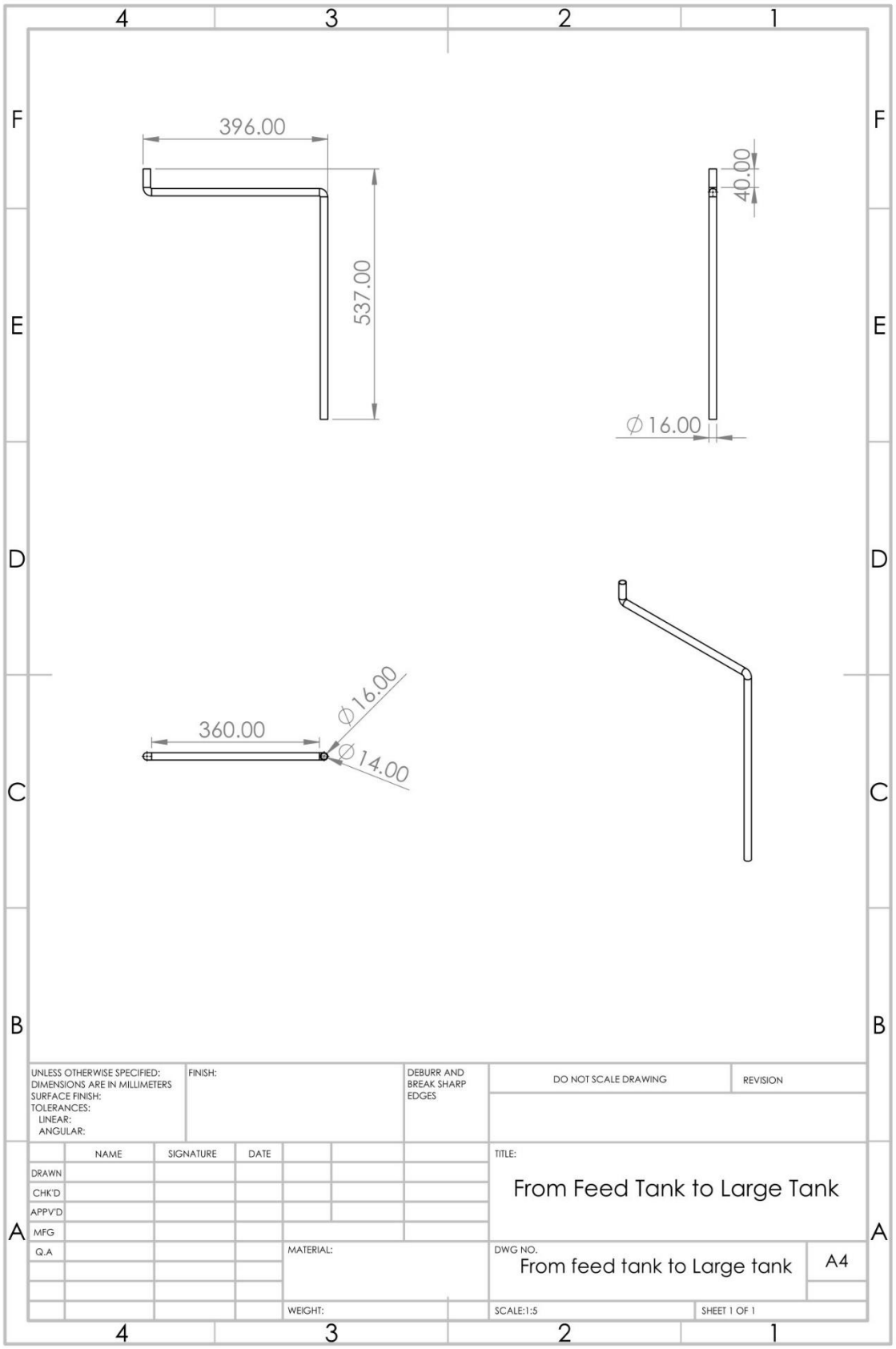
Appendix B

- **Mechanical Drawing**

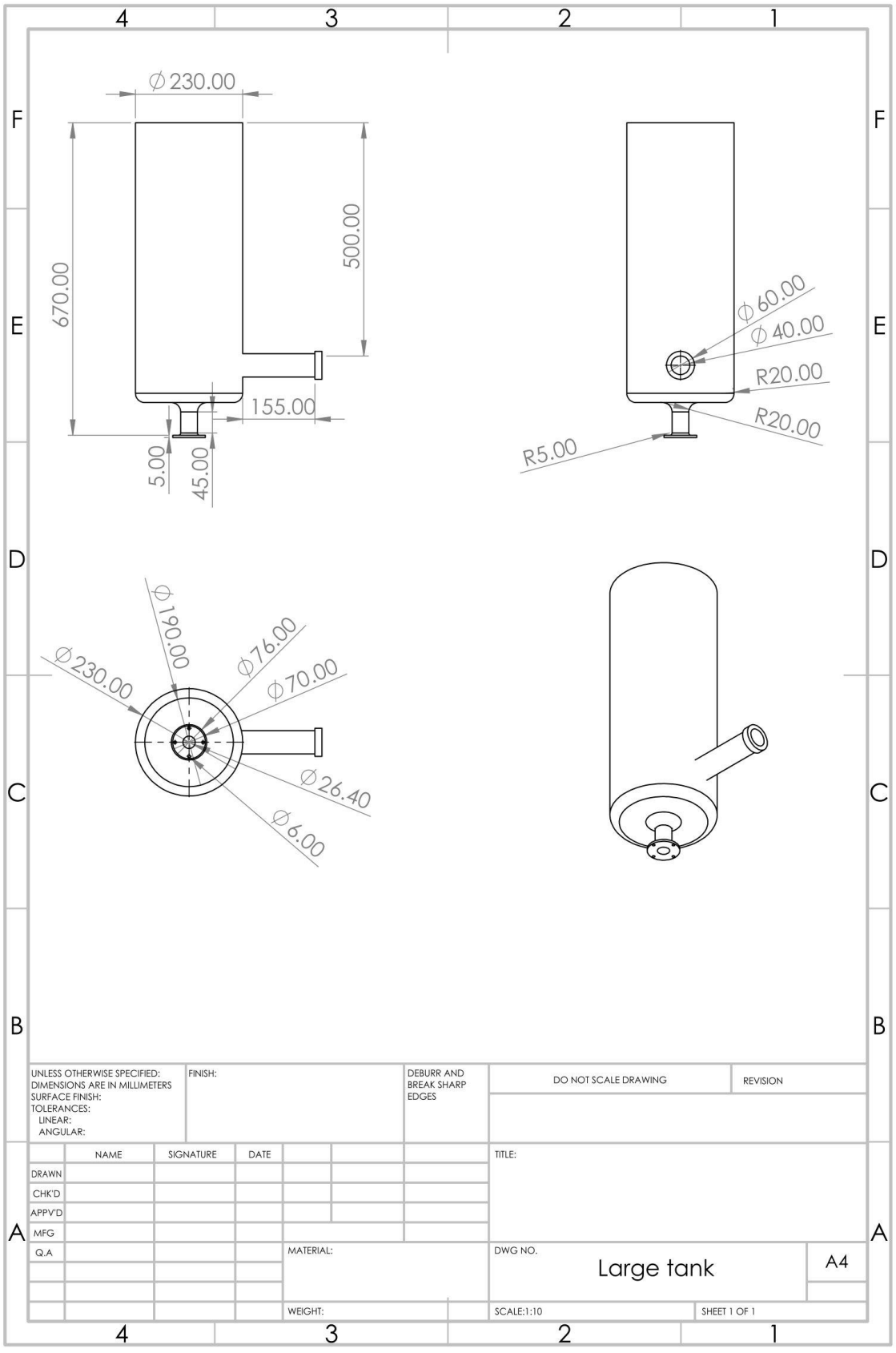


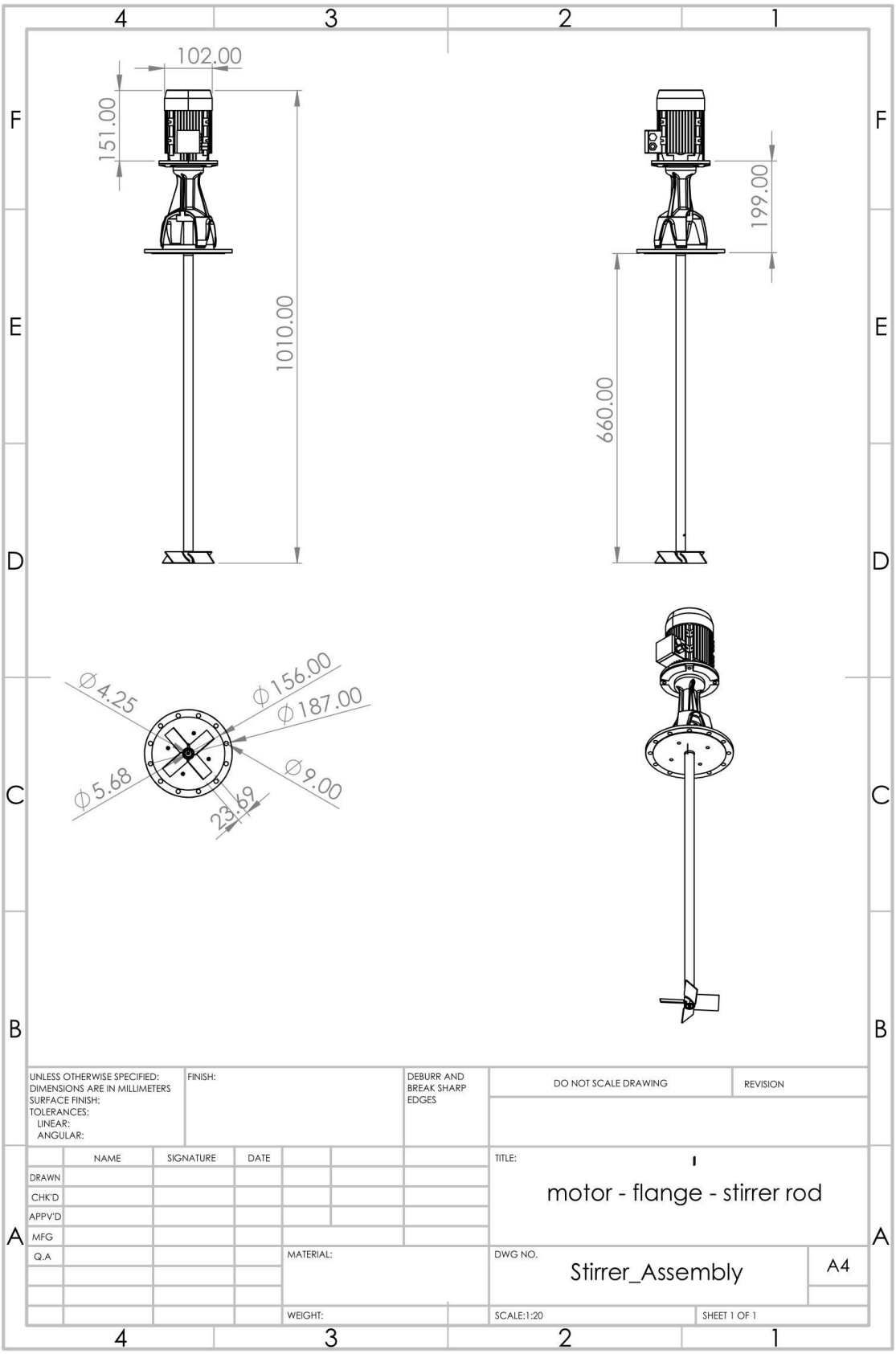

1365.00
764.00
1570.50
810.50
470.00
159.50
30.00
40.50
Ø184.00
225.50
Ø216.00
Ø9.00
Ø40.00
95.00
200.00
30.00
Ø12.00
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MILLIMETERS
SURFACE FINISH:
TOLERANCES:
LINEAR:
ANGULAR:
FINISH:
DEBURR AND BREAK SHARP EDGES
DO NOT SCALE DRAWING
REVISION
NAME
SIGNATURE
DATE
DRAWN
CHK'D
APPV'D
MFG
Q.A
TITLE:
MATERIAL:
DWG NO.
Frame
A4
WEIGHT:
SCALE:1:20
SHEET 1 OF 1

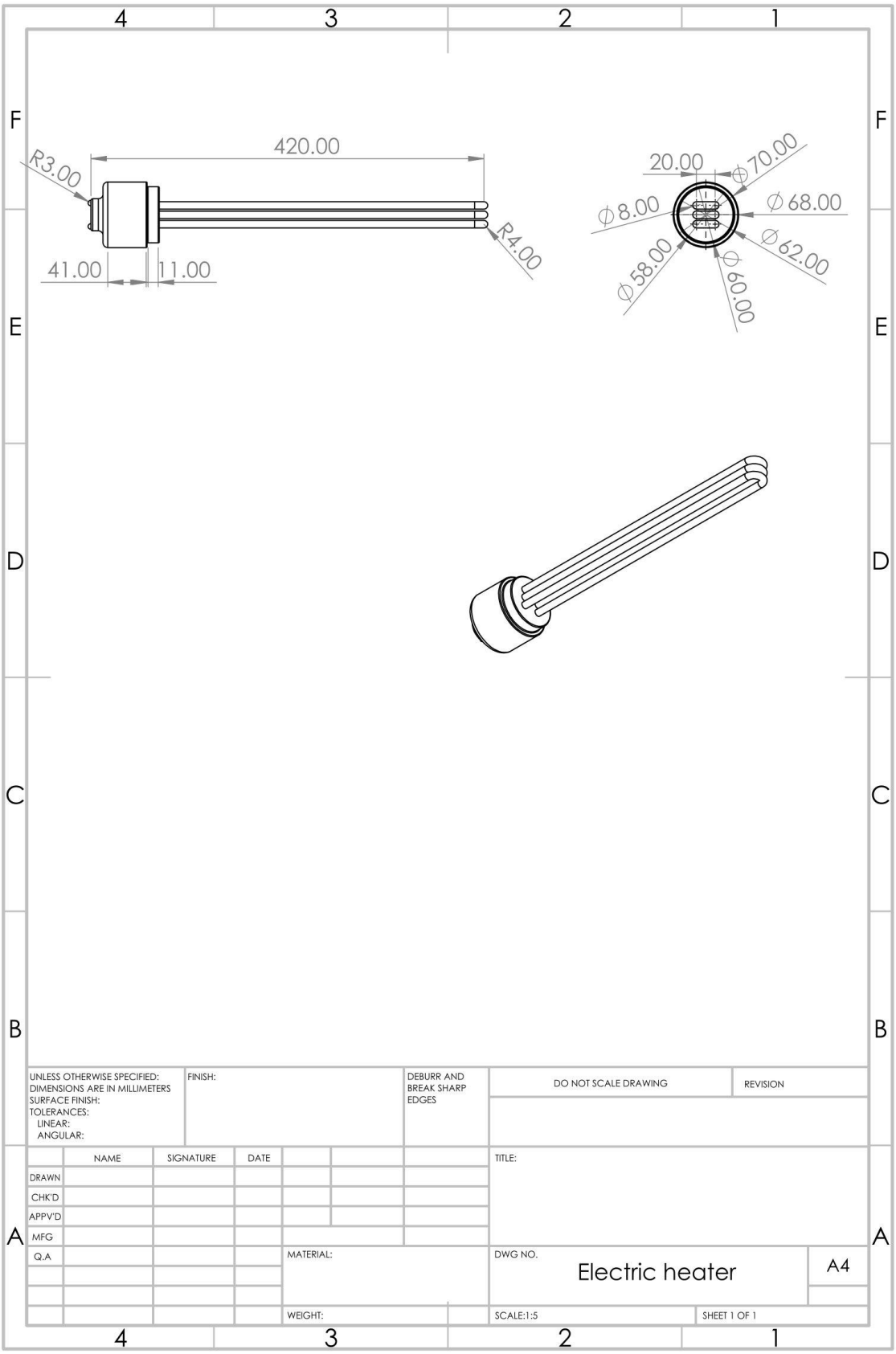

4
3
2
1
F
E
D
C
B
A
200.00
300.00
110.00
∅110.00
∅16.50
∅13.50
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MILLIMETERS
SURFACE FINISH:
TOLERANCES:
LINEAR:
ANGULAR:
FINISH:
DEBURR AND BREAK SHARP EDGES
DO NOT SCALE DRAWING
REVISION
NAME
SIGNATURE
DATE
DRAWN
CHK'D
APPV'D
MFG
Q.A
MATERIAL:
WEIGHT:
TITLE:
it is considered as the input for this system
DWG NO.
Feed_tank
A4
SCALE:1:5
SHEET 1 OF 1

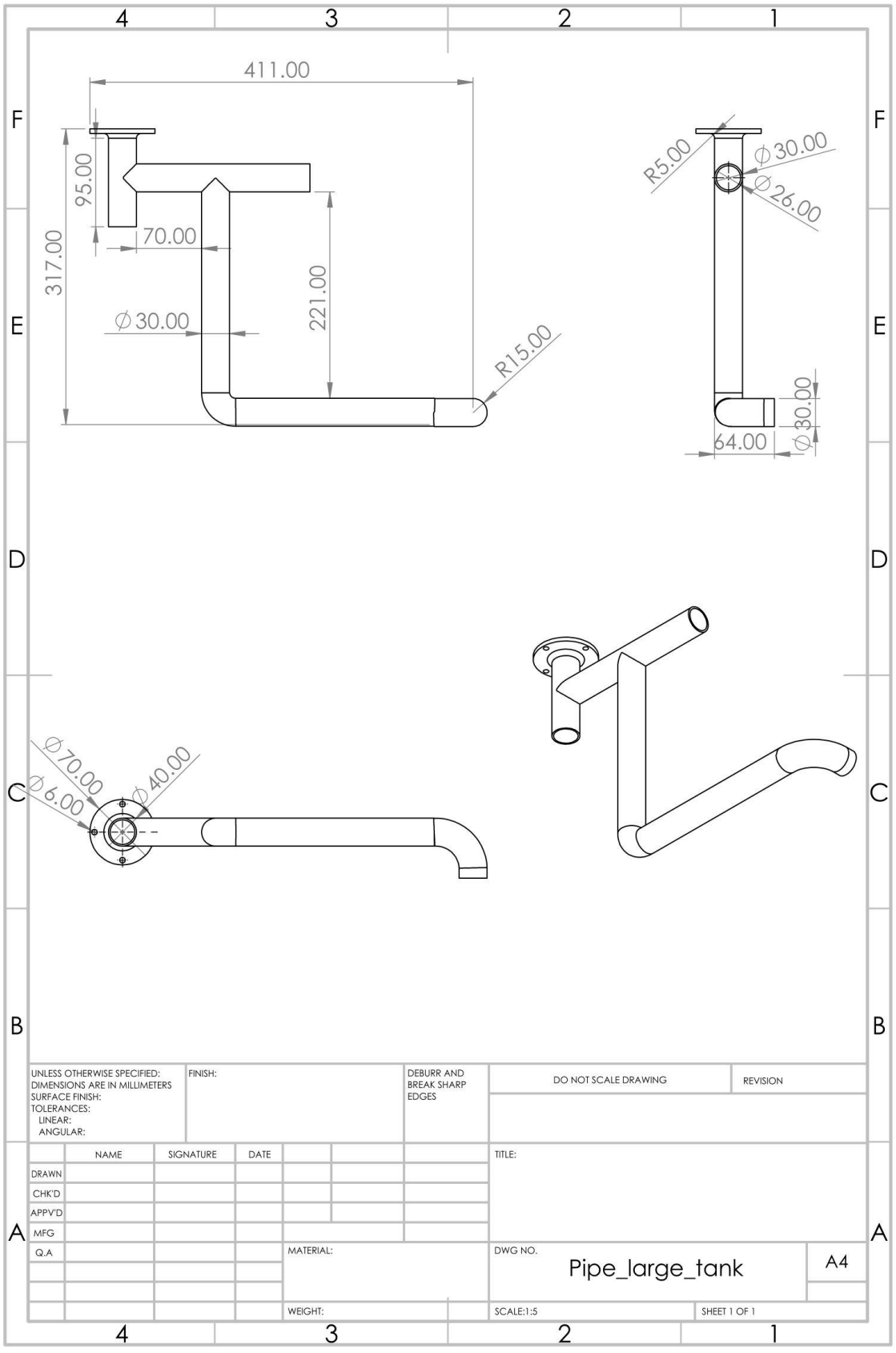


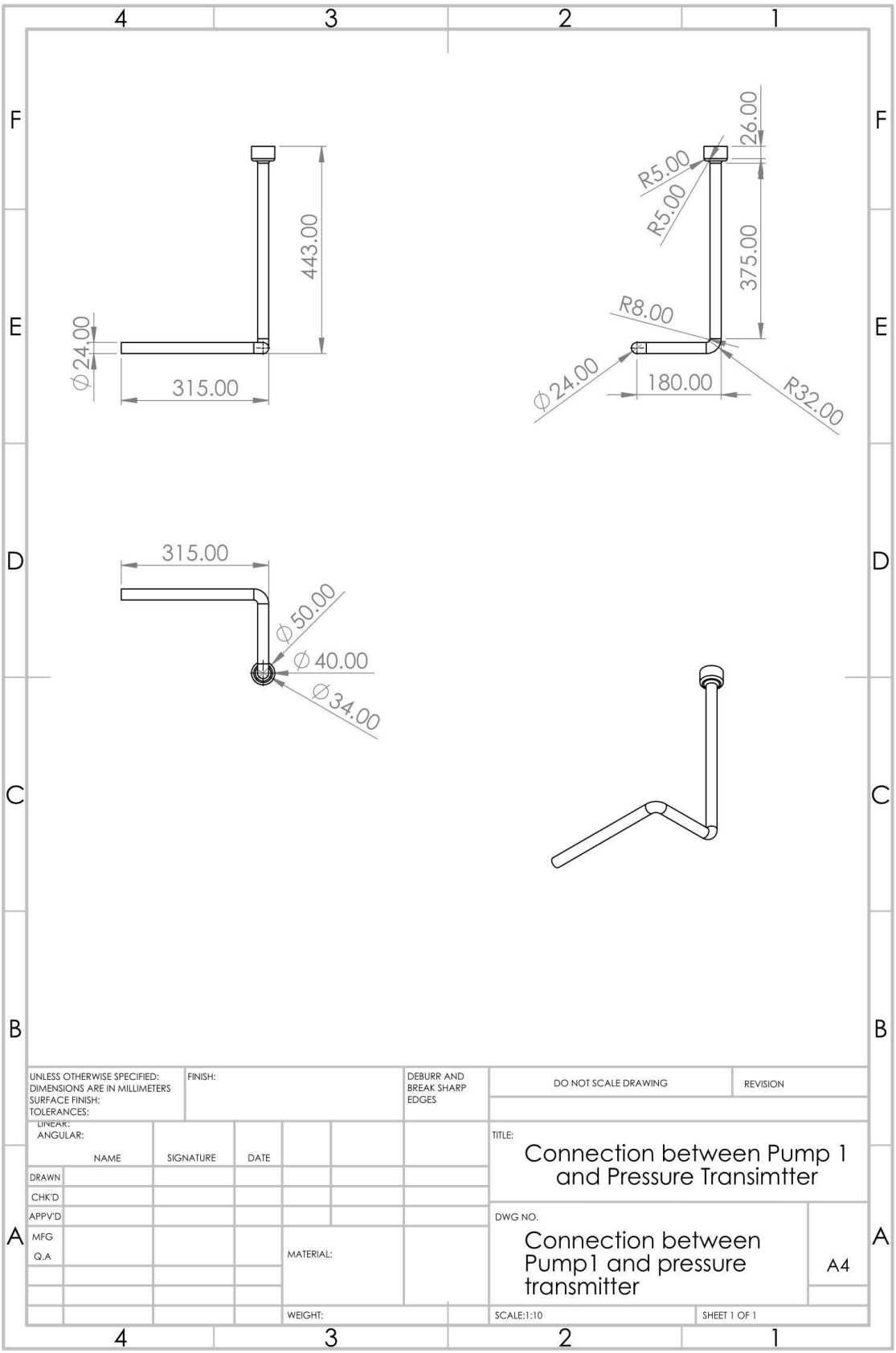
UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:		FINISH:		DEBURR AND BREAK SHARP EDGES	DO NOT SCALE DRAWING	REVISION	
	NAME	SIGNATURE	DATE		TITLE: From Feed Tank to Large Tank		
DRAWN							
CHK'D							
APPV'D							
MFG							
Q.A				MATERIAL:	DWG NO. From feed tank to Large tank	A4	
				WEIGHT:	SCALE:1:5	SHEET 1 OF 1	


4
3
2
1
F
E
D
C
B
A
∅230.00
670.00
500.00
155.00
5.00
45.00
∅60.00
∅40.00
R20.00
R20.00
R5.00
∅190.00
∅230.00
∅76.00
∅70.00
∅26.40
∅6.00
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MILLIMETERS
SURFACE FINISH:
TOLERANCES:
LINEAR:
ANGULAR:
FINISH:
DEBURR AND BREAK SHARP EDGES
DO NOT SCALE DRAWING
REVISION
NAME
SIGNATURE
DATE
DRAWN
CHK'D
APPV'D
MFG
Q.A
TITLE:
MATERIAL:
DWG NO.
Large tank
A4
WEIGHT:
SCALE:1:10
SHEET 1 OF 1

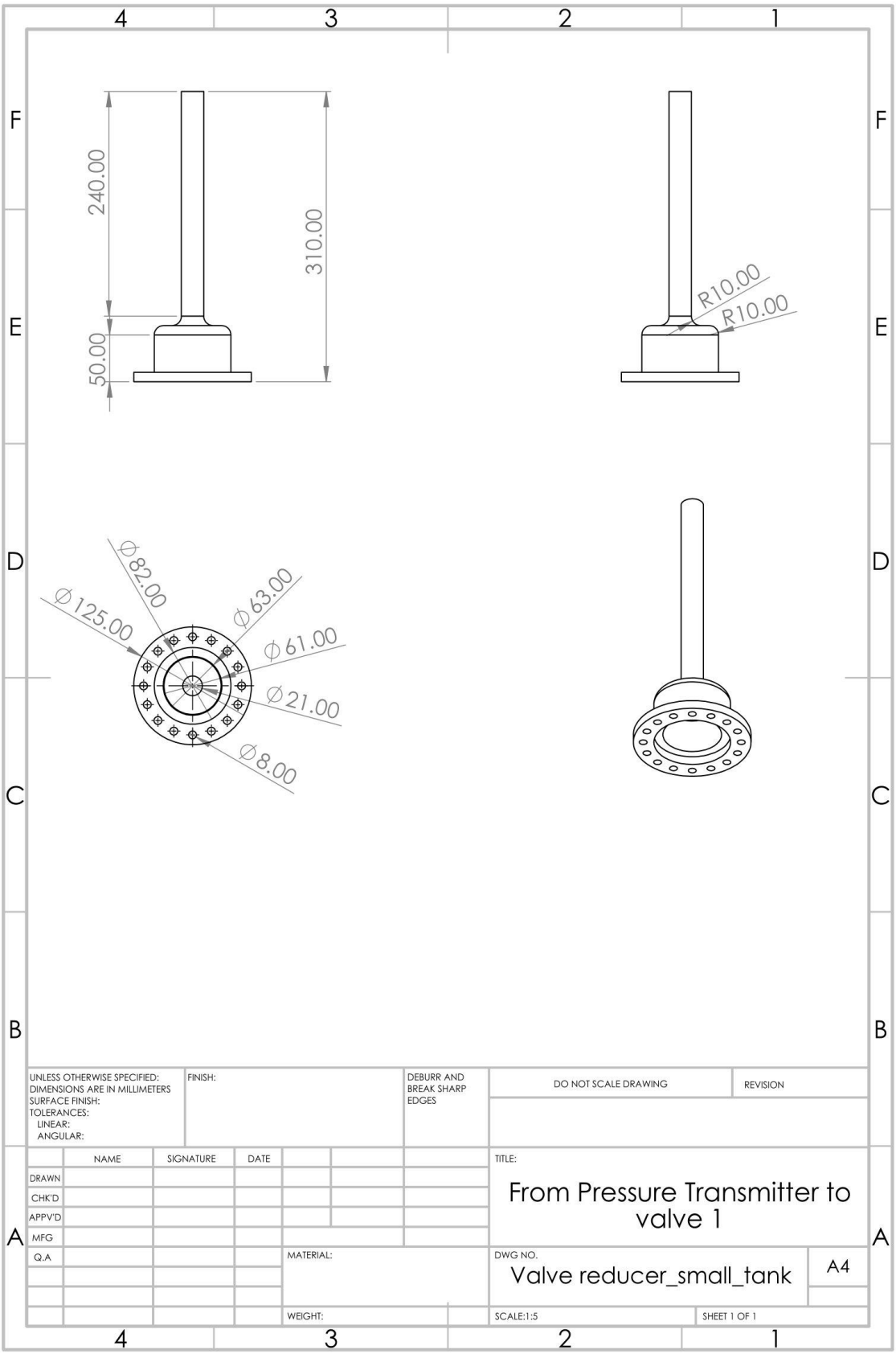

4
3
2
1
F
E
D
C
B
A
102.00
151.00
1010.00
199.00
660.00
Ø4.25
Ø156.00
Ø187.00
Ø5.68
Ø9.00
23.69
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MILLIMETERS
SURFACE FINISH:
TOLERANCES:
LINEAR:
ANGULAR:
FINISH:
DEBURR AND BREAK SHARP EDGES
DO NOT SCALE DRAWING
REVISION
NAME
SIGNATURE
DATE
DRAWN
CHK'D
APPV'D
MFG
Q.A
TITLE:
motor - flange - stirrer rod
MATERIAL:
DWG NO.
Stirrer_Assembly
A4
WEIGHT:
SCALE:1:20
SHEET 1 OF 1

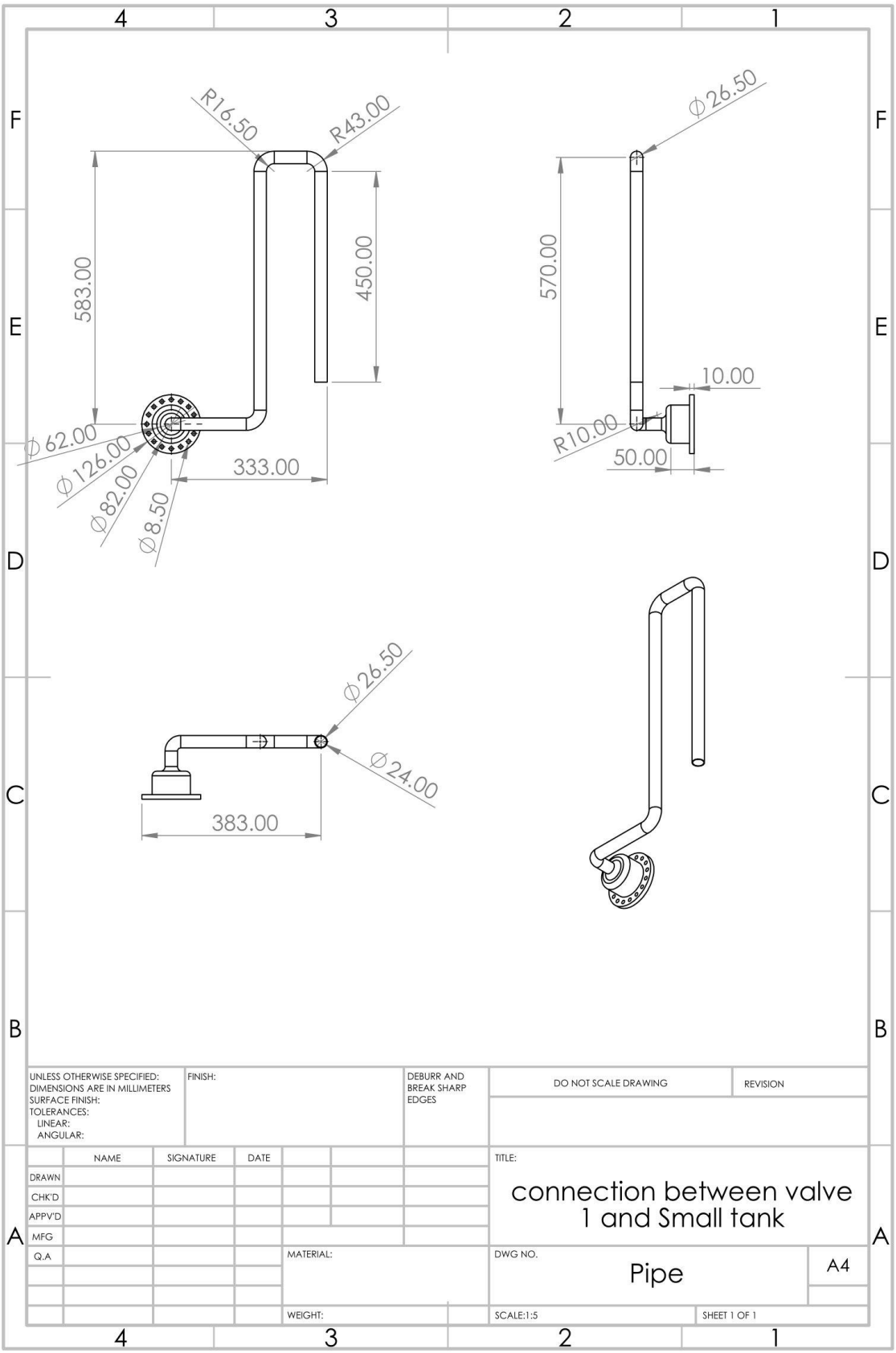

4
3
2
1
F
E
D
C
B
A
420.00
R3.00
R4.00
41.00
11.00
20.00
Ø70.00
Ø8.00
Ø68.00
Ø62.00
Ø58.00
Ø60.00
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MILLIMETERS
SURFACE FINISH:
TOLERANCES:
LINEAR:
ANGULAR:
FINISH:
DEBURR AND BREAK SHARP EDGES
DO NOT SCALE DRAWING
REVISION
NAME
SIGNATURE
DATE
TITLE:
DRAWN
CHK'D
APPV'D
MFG
Q.A
MATERIAL:
DWG NO.
Electric heater
A4
WEIGHT:
SCALE:1:5
SHEET 1 OF 1

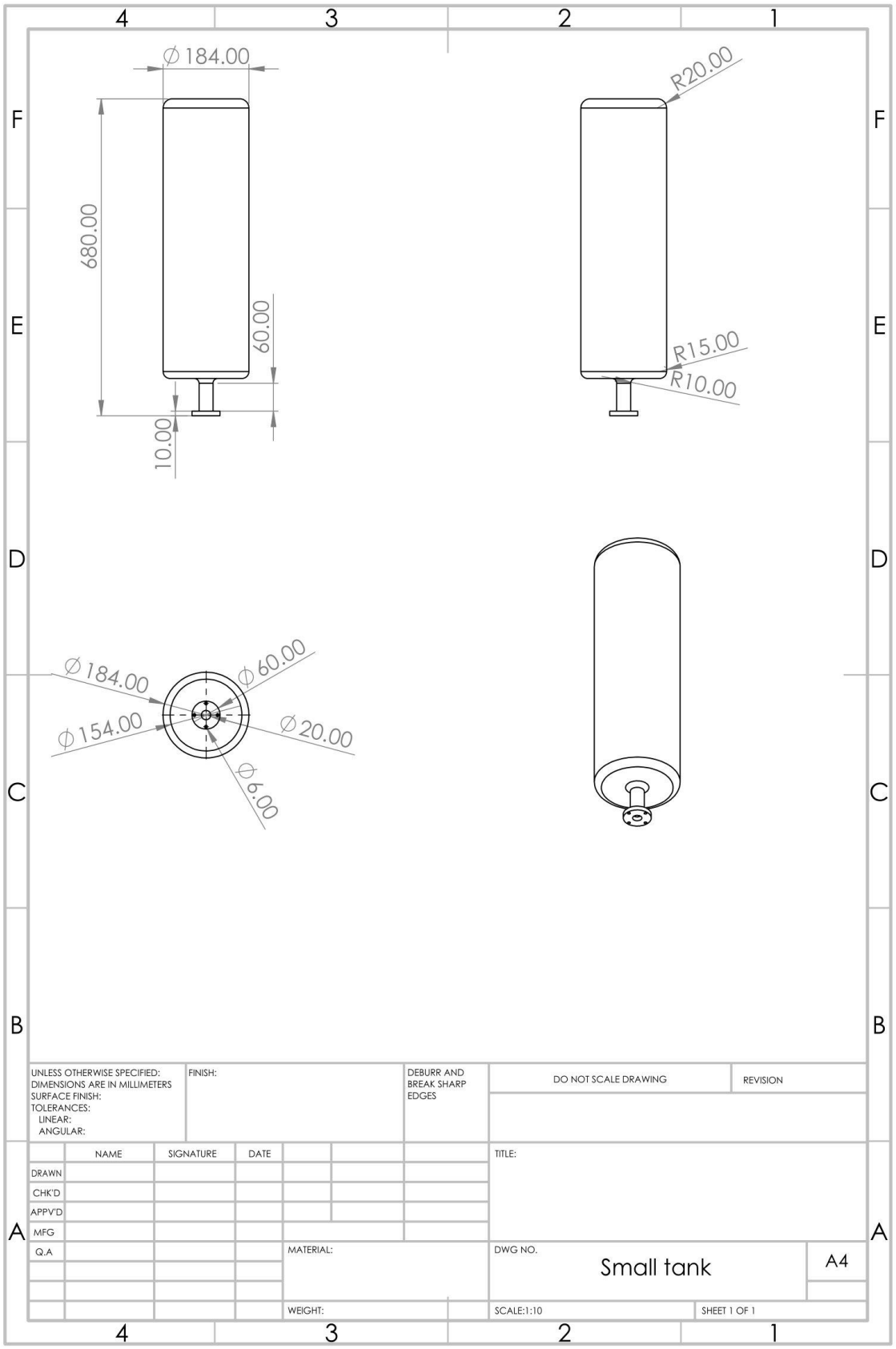

411.00
95.00
317.00
70.00
221.00
∅30.00
R15.00
R5.00
∅30.00
∅26.00
∅30.00
64.00
∅70.00
∅40.00
∅6.00
4
3
2
1
F
E
D
C
B
A
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MILLIMETERS
SURFACE FINISH:
TOLERANCES:
LINEAR:
ANGULAR:
FINISH:
DEBURR AND BREAK SHARP EDGES
DO NOT SCALE DRAWING
REVISION
NAME
SIGNATURE
DATE
TITLE:
DRAWN
CHK'D
APPV'D
MFG
Q.A
MATERIAL:
DWG NO.
Pipe_large_tank
A4
WEIGHT:
SCALE:1:5
SHEET 1 OF 1

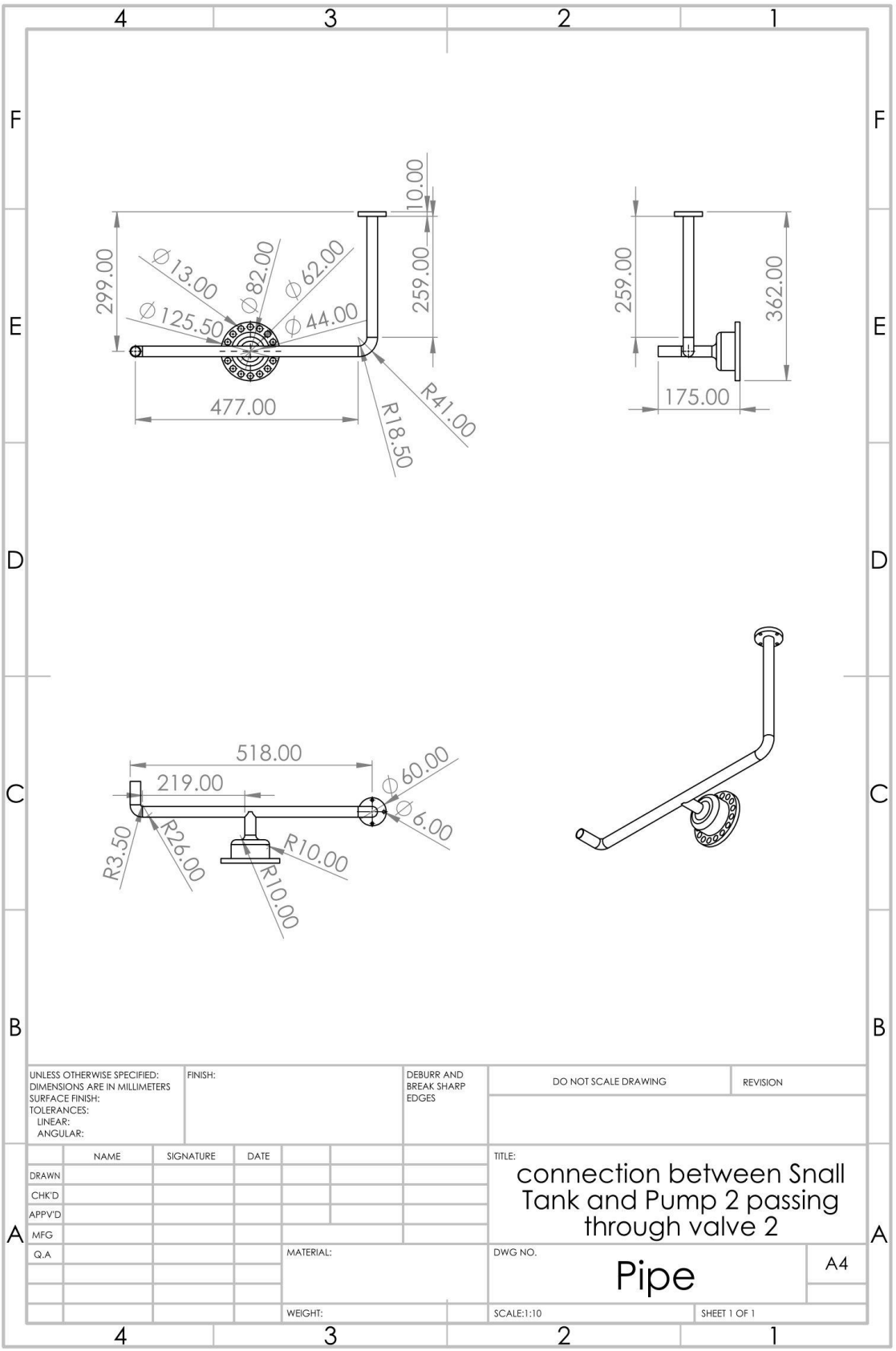


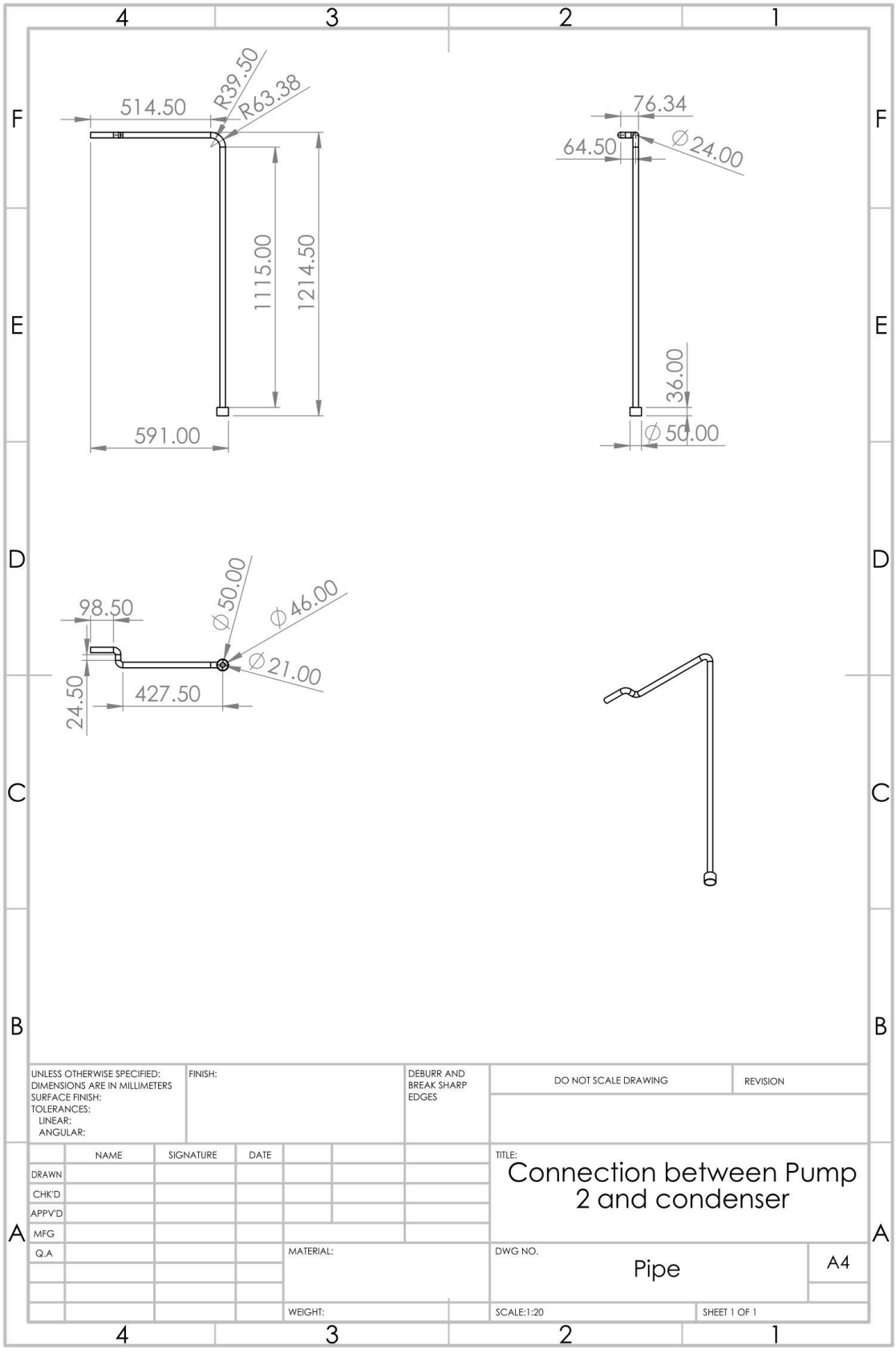
UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES:		FINISH:		DEBURR AND BREAK SHARP EDGES	DO NOT SCALE DRAWING	REVISION
LINEAR: ANGULAR:	NAME	SIGNATURE	DATE		TITLE: Connection between Pump 1 and Pressure Transimtter	
DRAWN						
CHK'D						
APPV'D					DWG NO. Connection between Pump1 and pressure transmitter	A4
MFG				MATERIAL:		
Q.A						
				WEIGHT:	SCALE:1:10	SHEET 1 OF 1

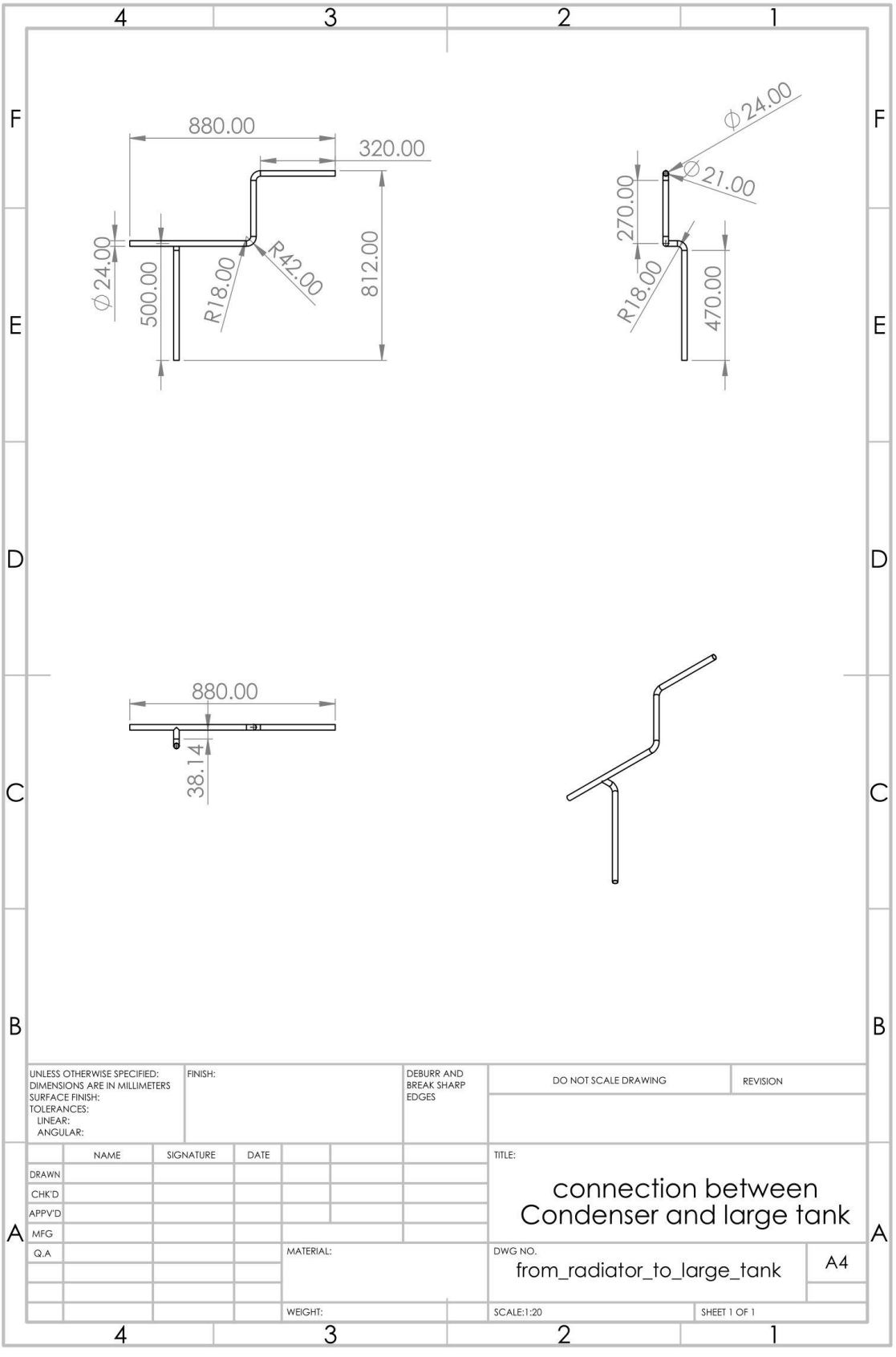

4
3
2
1
F
E
D
C
B
A
240.00
310.00
50.00
R10.00
R10.00
Ø82.00
Ø125.00
Ø63.00
Ø61.00
Ø21.00
Ø8.00
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MILLIMETERS
SURFACE FINISH:
TOLERANCES:
LINEAR:
ANGULAR:
FINISH:
DEBURR AND BREAK SHARP EDGES
DO NOT SCALE DRAWING
REVISION
NAME
SIGNATURE
DATE
DRAWN
CHK'D
APPV'D
MFG
Q.A
TITLE:
From Pressure Transmitter to valve 1
MATERIAL:
DWG NO.
Valve reducer_small_tank
A4
WEIGHT:
SCALE:1:5
SHEET 1 OF 1


4
3
2
1
F
E
D
C
B
A
R16.50
R43.00
583.00
450.00
Ø62.00
Ø126.00
Ø82.00
Ø8.50
333.00
Ø26.50
570.00
10.00
R10.00
50.00
Ø26.50
Ø24.00
383.00
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MILLIMETERS
SURFACE FINISH:
TOLERANCES:
LINEAR:
ANGULAR:
FINISH:
DEBURR AND BREAK SHARP EDGES
DO NOT SCALE DRAWING
REVISION
NAME
SIGNATURE
DATE
DRAWN
CHK'D
APPV'D
MFG
Q.A
TITLE:
connection between valve 1 and Small tank
MATERIAL:
DWG NO.
Pipe
A4
WEIGHT:
SCALE:1:5
SHEET 1 OF 1

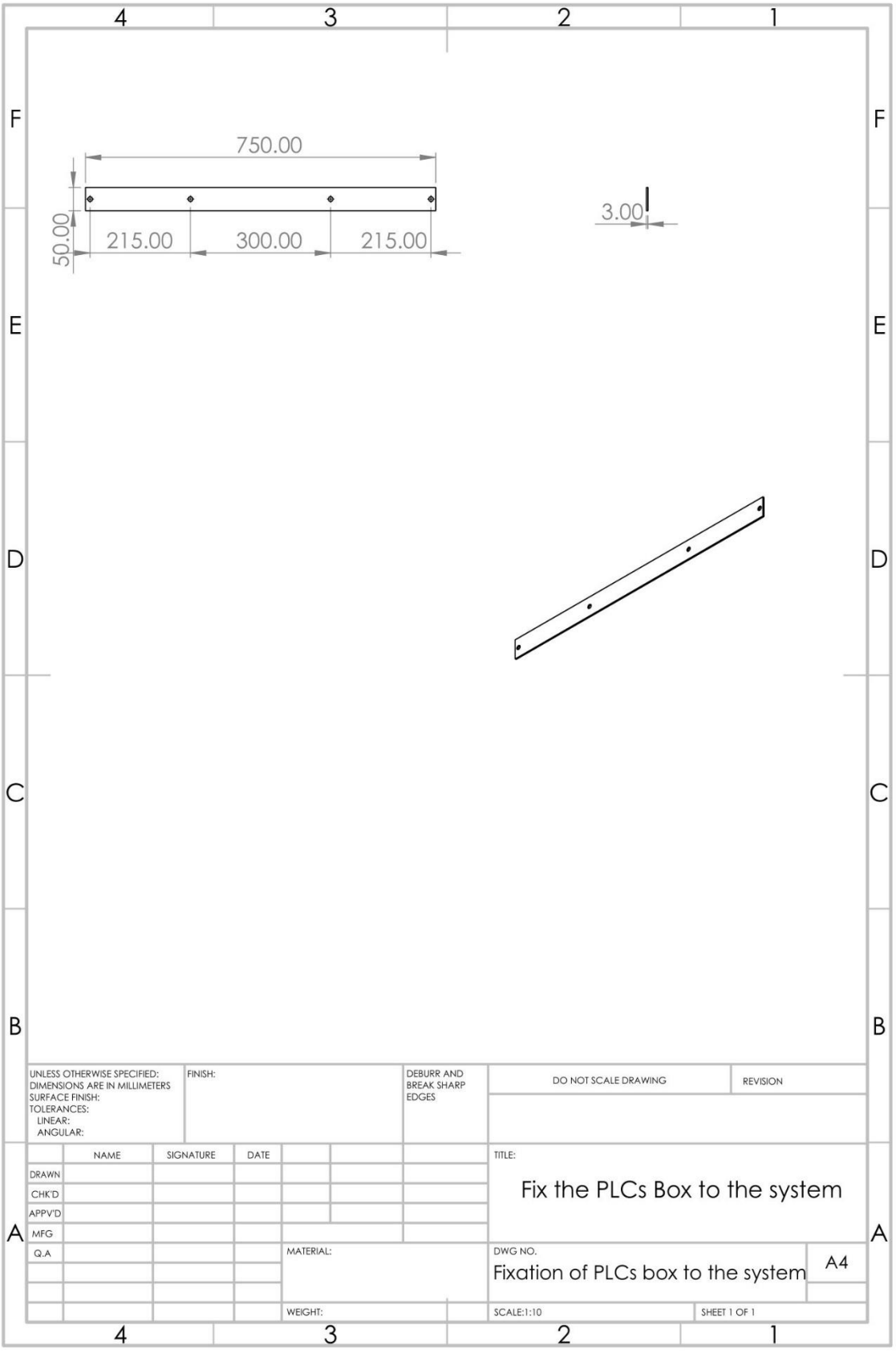

4
3
2
1
F
E
D
C
B
A
∅184.00
680.00
60.00
10.00
R20.00
R15.00
R10.00
∅184.00
∅60.00
∅154.00
∅20.00
∅6.00
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MILLIMETERS
SURFACE FINISH:
TOLERANCES:
LINEAR:
ANGULAR:
FINISH:
DEBURR AND BREAK SHARP EDGES
DO NOT SCALE DRAWING
REVISION
NAME
SIGNATURE
DATE
TITLE:
DRAWN
CHK'D
APPV'D
MFG
Q.A
MATERIAL:
DWG NO.
Small tank
A4
WEIGHT:
SCALE:1:10
SHEET 1 OF 1


4
3
2
1
F
E
D
C
B
A
10.00
299.00
259.00
Ø13.00
Ø82.00
Ø62.00
Ø125.50
Ø44.00
477.00
R18.50
R41.00
259.00
362.00
175.00
518.00
219.00
Ø60.00
Ø6.00
R3.50
R26.00
R10.00
R10.00
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MILLIMETERS
SURFACE FINISH:
TOLERANCES:
LINEAR:
ANGULAR:
FINISH:
DEBURR AND BREAK SHARP EDGES
DO NOT SCALE DRAWING
REVISION
NAME
SIGNATURE
DATE
DRAWN
CHK'D
APPV'D
MFG
Q.A
MATERIAL:
WEIGHT:
TITLE:
connection between Snall Tank and Pump 2 passing through valve 2
DWG NO.
Pipe
A4
SCALE:1:10
SHEET 1 OF 1


514.50
R39.50
R63.38
1115.00
1214.50
591.00
76.34
64.50
Ø24.00
36.00
Ø50.00
98.50
Ø50.00
Ø46.00
Ø21.00
24.50
427.50
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MILLIMETERS
SURFACE FINISH:
TOLERANCES:
LINEAR:
ANGULAR:
FINISH:
DEBURR AND BREAK SHARP EDGES
DO NOT SCALE DRAWING
REVISION
NAME
SIGNATURE
DATE
DRAWN
CHK'D
APPV'D
MFG
Q.A
TITLE:
Connection between Pump 2 and condenser
MATERIAL:
DWG NO.
Pipe
A4
WEIGHT:
SCALE:1:20
SHEET 1 OF 1

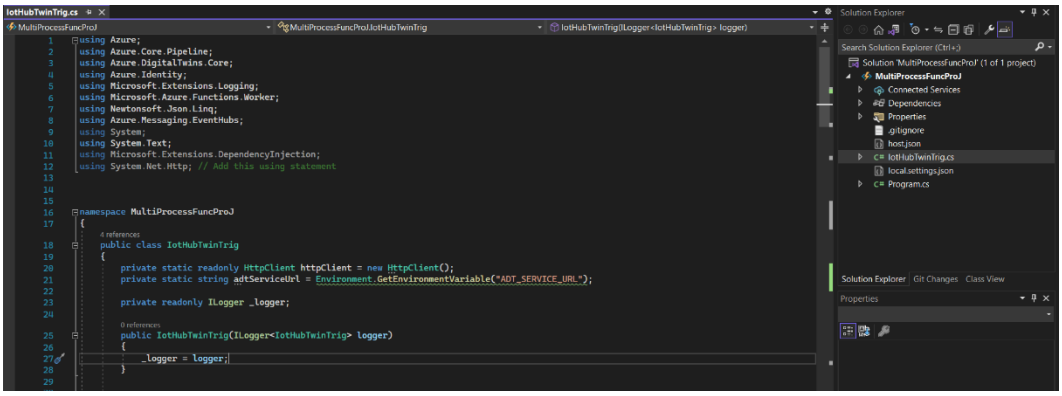


UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:		FINISH:		DEBURR AND BREAK SHARP EDGES	DO NOT SCALE DRAWING	REVISION	
	NAME	SIGNATURE	DATE		TITLE: connection between Condenser and large tank		
DRAWN							
CHK'D							
APPV'D							
MFG							
Q.A				MATERIAL:	DWG NO. from_radiator_to_large_tank	A4	
				WEIGHT:	SCALE:1:20	SHEET 1 OF 1	


4
3
2
1
F
E
D
C
B
A
750.00
50.00
215.00
300.00
215.00
3.00
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MILLIMETERS
SURFACE FINISH:
TOLERANCES:
LINEAR:
ANGULAR:
FINISH:
DEBURR AND BREAK SHARP EDGES
DO NOT SCALE DRAWING
REVISION
NAME
SIGNATURE
DATE
DRAWN
CHK'D
APPV'D
MFG
Q.A
TITLE:
Fix the PLCs Box to the system
MATERIAL:
DWG NO.
Fixation of PLCs box to the system
A4
WEIGHT:
SCALE:1:10
SHEET 1 OF 1

Appendix C

- **Code Of Function "`IotHubTwinTrig`"**



```
using Azure;
using Azure.Core.Pipeline;
using Azure.DigitalTwins.Core;
using Azure.Identity;
using Microsoft.Extensions.Logging;
using Microsoft.Azure.Functions.Worker;
using Newtonsoft.Json.Linq;
using Azure.Messaging.EventHubs;
using System;
using System.Text;
using Microsoft.Extensions.DependencyInjection;
using System.Net.Http; // Add this using statement


namespace MultiProcessFuncProJ
{
    4 references
    public class IotHubTwinTrig
    {
        private static readonly HttpClient httpClient = new HttpClient();
        private static string adtServiceUrl = Environment.GetEnvironmentVariable("ADT_SERVICE_URL");

        private readonly ILogger _logger;

        0 references
        public IotHubTwinTrig(ILogger<IotHubTwinTrig> logger)
        {
            _logger = logger;
        }
```

```
[Function(nameof(IotHubTwinTrig))]
1 reference
public async Task Run([EventHubTrigger("m2ao_eventhub", Connection = "EventHubConnectionString")] EventData[] events)
{
    //Authenticate with Digital Twins
    var credentials = new DefaultAzureCredential();
    DigitalTwinsClient client = new DigitalTwinsClient(
        new Uri(adtServiceUrl), credentials, new DigitalTwinsClientOptions
        { Transport = new HttpClientTransport(httpClient) });
    _logger.LogInformation($"ADT service client connection created.");

    foreach (EventData @event in events)
    {
        try
        {
            string messageBody = Encoding.UTF8.GetString(@event.Body.ToArray());
            JObject deviceMessage = JObject.Parse(messageBody);

            // Ensure deviceMessage is not null
            if (deviceMessage != null)
            {
                _logger.LogInformation($"Deviceissss::::{deviceMessage}");
                var TT1_PLC1 = deviceMessage["TT1_PLC1"];
                //var ST1_PLC1 = deviceMessage["ST1_PLC1"];
                var FT1_PLC2 = deviceMessage["FT1_PLC2"];
                var WT1_PLC2 = deviceMessage["WT1_PLC2"];
                var TT2_PLC3 = deviceMessage["TT2_PLC3"];
                var Fan = deviceMessage["Fan"];
                var Stirrer = deviceMessage["Stirrer"];
                var FC1 = deviceMessage["FC1"];
                var LC1 = deviceMessage["LC1"];
                var Pump1 = deviceMessage["Pump1"];
                var Pump2 = deviceMessage["Pump2"];
                var J1 = deviceMessage["J1"];
```

```
                // Ensure temperature is not null
                if (TT1_PLC1 != null)
                {
                    _logger.LogInformation($"TT1_PLC1 is:{TT1_PLC1}");
                    // _logger.LogInformation($"ST1_PLC1 is:{ST1_PLC1}");
                    _logger.LogInformation($"FT1_PLC2 is:{FT1_PLC2}");
                    _logger.LogInformation($"WT1_PLC2 is:{WT1_PLC2}");
                    _logger.LogInformation($"TT2_PLC3 is:{TT2_PLC3}");
                    _logger.LogInformation($"Fan is:{Fan}");
                    _logger.LogInformation($"Stirrer is:{Stirrer}");
                    _logger.LogInformation($"FC1 is:{FC1}");
                    _logger.LogInformation($"LC1 is:{LC1}");
                    _logger.LogInformation($"Pump1 is:{Pump1}");
                    _logger.LogInformation($"Pump2 is:{Pump2}");
                    _logger.LogInformation($"J1 is:{J1}");



                    // Update twin using device temperature
                    var updateTwinMainTank = new JsonPatchDocument();
                    var updateTwinDataSubTank = new JsonPatchDocument();
                    var updateTwinDataRadiator = new JsonPatchDocument();
                    var updateTwinDataFlowTransmiter = new JsonPatchDocument();
                    var updateTwinDatacontrolvalve1 = new JsonPatchDocument();
                    var updateTwinDatacontrolvalve2 = new JsonPatchDocument();
                    var updateTwinDatapump1 = new JsonPatchDocument();
                    var updateTwinDatapump2 = new JsonPatchDocument();


                    updateTwinMainTank.AppendReplace("/TT1_PLC1", TT1_PLC1.Value<double>());
                    updateTwinDataFlowTransmiter.AppendReplace("/FT1_PLC2", FT1_PLC2.Value<double>());
                    updateTwinDataSubTank.AppendReplace("/WT1_PLC2", WT1_PLC2.Value<double>());
                    updateTwinMainTank.AppendReplace("/TT2_PLC3", TT2_PLC3.Value<double>());
                    updateTwinDataRadiator.AppendReplace("/Fan", Fan.Value<double>());
                    updateTwinMainTank.AppendReplace("/Stirrer", Stirrer.Value<double>());
                    updateTwinDatacontrolvalve1.AppendReplace("/FC1", FC1.Value<double>());
```

```
                    // Update twin using device temperature
                    var updateTwinMainTank = new JsonPatchDocument();
                    var updateTwinDataSubTank = new JsonPatchDocument();
                    var updateTwinDataRadiator = new JsonPatchDocument();
                    var updateTwinDataFlowTransmiter = new JsonPatchDocument();
                    var updateTwinDatacontrolvalve1 = new JsonPatchDocument();
                    var updateTwinDatacontrolvalve2 = new JsonPatchDocument();
                    var updateTwinDatapump1 = new JsonPatchDocument();
                    var updateTwinDatapump2 = new JsonPatchDocument();


                    updateTwinMainTank.AppendReplace("/TT1_PLC1", TT1_PLC1.Value<double>());
                    updateTwinDataFlowTransmiter.AppendReplace("/FT1_PLC2", FT1_PLC2.Value<double>());
                    updateTwinDataSubTank.AppendReplace("/WT1_PLC2", WT1_PLC2.Value<double>());
                    updateTwinMainTank.AppendReplace("/TT2_PLC3", TT2_PLC3.Value<double>());
                    updateTwinDataRadiator.AppendReplace("/Fan", Fan.Value<double>());
                    updateTwinMainTank.AppendReplace("/Stirrer", Stirrer.Value<double>());
                    updateTwinDatacontrolvalve1.AppendReplace("/FC1", FC1.Value<double>());
                    updateTwinDatacontrolvalve2.AppendReplace("/LC1", LC1.Value<double>());
                    updateTwinDatapump1.AppendReplace("/Pump1", Pump1.Value<double>());
                    updateTwinDatapump2.AppendReplace("/Pump2", Pump2.Value<double>());
                    updateTwinMainTank.AppendReplace("/J1", J1.Value<double>());



                    await client.UpdateDigitalTwinAsync("MainTank", updateTwinMainTank);
                    await client.UpdateDigitalTwinAsync("SubTank", updateTwinDataSubTank);
                    await client.UpdateDigitalTwinAsync("Radiator", updateTwinDataRadiator);
                    await client.UpdateDigitalTwinAsync("PressureTransmitterA", updateTwinDataFlowTransmiter);
                    await client.UpdateDigitalTwinAsync("ControlValveA", updateTwinDatacontrolvalve1);
                    await client.UpdateDigitalTwinAsync("ControlValveB", updateTwinDatacontrolvalve2);
                    await client.UpdateDigitalTwinAsync("PumpA", updateTwinDatapump1);
                    await client.UpdateDigitalTwinAsync("PumpB", updateTwinDatapump2);
```

```
                    await client.UpdateDigitalTwinAsync("MainTank", updateTwinMainTank);
                    await client.UpdateDigitalTwinAsync("SubTank", updateTwinDataSubTank);
                    await client.UpdateDigitalTwinAsync("Radiator", updateTwinDataRadiator);
                    await client.UpdateDigitalTwinAsync("PressureTransmitterA", updateTwinDataFlowTransmiter);
                    await client.UpdateDigitalTwinAsync("ControlValveA", updateTwinDatacontrolvalve1);
                    await client.UpdateDigitalTwinAsync("ControlValveB", updateTwinDatacontrolvalve2);
                    await client.UpdateDigitalTwinAsync("PumpA", updateTwinDatapump1);
                    await client.UpdateDigitalTwinAsync("PumpB", updateTwinDatapump2);



                    _logger.LogInformation($"Updated Digital Twin for device: MY System M2OA");

                }



                else
                {
                    _logger.LogWarning("Data is null or invalid.");
                }
            }
            else
            {
                _logger.LogWarning("Device message is null.");
            }
        }
        catch (Exception ex)
        {
            _logger.LogError($"Error processing message: {ex.Message}");
        }
    }
}
```

- **DTDL Models Of System**

```
{
  "@id": "dtmi:example:MainTank;1",
  "@type": "Interface",
  "displayName": "MainTank",
  "@context": "dtmi:dtdl:context;2",
  "contents": [
    {
      "@type": "Relationship",
      "name": "contains",
      "displayName": "contains",
      "target": "dtmi:example:PumpA;1"
    },
    {
      "@type": "Property",
      "name": "TT1_PLC1",
      "schema": "double"
    },
    {
      "@type": "Property",
      "name": "TT2_PLC3",
      "schema": "double"
    },
    {
      "@type": "Property",
      "name": "Stirrer",
      "schema": "double"
    },
    {
      "@type": "Property",
      "name": "J1",
      "schema": "double"
    }
  ]
}
```

```
{
  "@id": "dtmi:example:Structure;1",
  "@type": "Interface",
  "displayName": "Structure",
  "@context": "dtmi:dtdl:context;2",
  "contents": [
    {
      "@type": "Relationship",
      "name": "contains",
      "displayName": "contains",
      "target": "dtmi:example:MainTank;1"
    }
  ]
}
```

```
{
  "@id": "dtmi:example:PumpA;1",
  "@type": "Interface",
  "displayName": "PumpA",
  "@context": "dtmi:dtdl:context;2",
  "contents": [
    {
      "@type": "Relationship",
      "name": "contains",
      "displayName": "contains",
      "target": "dtmi:example:PressureTransmitterA;1"
    },
    {
      "@type": "Property",
      "name": "Pump1",
      "schema": "double"
    }
  ]
}
```

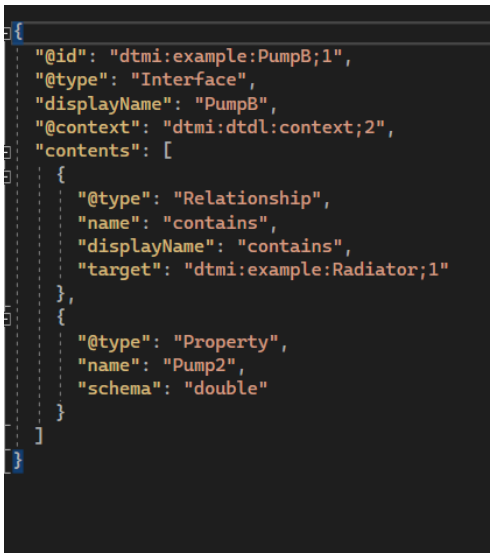
```
{
  "@id": "dtmi:example:ControlValveA;1",
  "@type": "Interface",
  "displayName": "ControlValveA",
  "@context": "dtmi:dtdl:context;2",
  "contents": [
    {
      "@type": "Relationship",
      "name": "contains",
      "displayName": "contains",
      "target": "dtmi:example:SubTank;1"
    },
    {
      "@type": "Property",
      "name": "FC1",
      "schema": "double"
    }
  ]
}
```

```
{
  "@id": "dtmi:example:PressureTransmitterA;1",
  "@type": "Interface",
  "displayName": "PressureTransmitterA",
  "@context": "dtmi:dtdl:context;2",
  "contents": [
    {
      "@type": "Relationship",
      "name": "contains",
      "displayName": "contains",
      "target": "dtmi:example:ControlValveA;1"
    },
    {
      "@type": "Property",
      "name": "FT1_PLC2",
      "schema": "double"
    }
  ]
}
```

```
{
  "@id": "dtmi:example:SubTank;1",
  "@type": "Interface",
  "displayName": "SubTank",
  "@context": "dtmi:dtdl:context;2",
  "contents": [
    {
      "@type": "Relationship",
      "name": "contains",
      "displayName": "contains",
      "target": "dtmi:example:ControlValveB;1"
    },

    {
      "@type": "Property",
      "name": "WT1_PLC2",
      "schema": "double"
    }
  ]
}
```

```
{
  "@id": "dtmi:example:ControlValveB;1",
  "@type": "Interface",
  "displayName": "ControlValveB",
  "@context": "dtmi:dtdl:context;2",
  "contents": [
    {
      "@type": "Relationship",
      "name": "contains",
      "displayName": "contains",
      "target": "dtmi:example:PumpB;1"
    },
    {
      "@type": "Property",
      "name": "LC1",
      "schema": "double"
    }
  ]
}
```



```
{
  "@id": "dtmi:example:PumpB;1",
  "@type": "Interface",
  "displayName": "PumpB",
  "@context": "dtmi:dtdl:context;2",
  "contents": [
    {
      "@type": "Relationship",
      "name": "contains",
      "displayName": "contains",
      "target": "dtmi:example:Radiator;1"
    },
    {
      "@type": "Property",
      "name": "Pump2",
      "schema": "double"
    }
  ]
}
```

```
{
  "@id": "dtmi:example:Radiator;1",
  "@type": "Interface",
  "displayName": "Radiator",
  "@context": "dtmi:dtdl:context;2",
  "contents": [
    {
      "@type": "Property",
      "name": "Fan",
      "schema": "double"
    }
  ]
}
```

Appendix D

Azure_IoT_Hub_ESP32.ino | AzIoTSasToken.cpp | AzIoTSasToken.h | SerialLogger.cpp | SerialLogger.h | iot_configs.h | readme.md

```
// C99 libraries
#include <cstdlib>
#include <string.h>
#include <time.h>

// Libraries for MQTT client and WiFi connection
#include <WiFi.h>
#include <mqtt_client.h>

// Azure IoT SDK for C includes
#include <az_core.h>
#include <az_iot.h>
#include <azure_ca.h>

// Additional sample headers
#include "AzIoTSasToken.h"
#include "SerialLogger.h"
#include "iot_configs.h"
#include <ArduinoJson.h>

//Libraries  of Modbus RTU
#include "UbidotsEsp32Mqtt.h"
#include <ModbusMaster.h>
#include <SoftwareSerial.h>



/*|||||||||||||>Define the Modbus communication parameters|||||||||||||*/
#define BAUD_RATE 115200
```

```
/*|||||||||||||>Define the Modbus communication parameters|||||||||||||*/
#define BAUD_RATE 115200
#define PARITY 0
#define DATA_BITS 8
#define STOP_BITS 1
#define SerialRS485_RX_PIN 16
#define SerialRS485_TX_PIN 17
/*|||||||||||||Define the Modbus communication parameters<|||||||||||||*/




/*|||||||||||||>Define the slave addresses for the two devices||||||||||*/
#define PLC_SLAVE1_Station 1
#define PLC_SLAVE2_Station 2
#define PLC_SLAVE3_Station 3

SoftwareSerial Serial_mod(SerialRS485_RX_PIN, SerialRS485_TX_PIN);
/*|||||||||||||Define the slave addresses for the two devices<||||||||||*/



// When developing for your own Arduino-based platform,
// please follow the format '(ard;<platform>)'.
#define AZURE_SDK_CLIENT_USER_AGENT "c%2F" AZ_SDK_VERSION_STRING "(ard;esp32)"

// Utility macros and defines
#define sizeofarray(a) (sizeof(a) / sizeof(a[0]))
```

```
// please follow the format '(ard;<platform>)'.
#define AZURE_SDK_CLIENT_USER_AGENT "c%2F" AZ_SDK_VERSION_STRING "(ard;esp32)"

// Utility macros and defines
#define sizeofarray(a) (sizeof(a) / sizeof(a[0]))
#define NTP_SERVERS "pool.ntp.org", "time.nist.gov"
#define MQTT_QOS1 1
#define DO_NOT_RETAIN_MSG 0
#define SAS_TOKEN_DURATION_IN_MINUTES 60
#define UNIX_TIME_NOV_13_2017 1510592825

#define PST_TIME_ZONE -8
#define PST_TIME_ZONE_DAYLIGHT_SAVINGS_DIFF 1

#define GMT_OFFSET_SECS (PST_TIME_ZONE * 3600)
#define GMT_OFFSET_SECS_DST ((PST_TIME_ZONE + PST_TIME_ZONE_DAYLIGHT_SAVINGS_DIFF) * 3600)

// Translate iot_configs.h defines into variables used by the sample
static const char* ssid = IOT_CONFIG_WIFI_SSID;
static const char* password = IOT_CONFIG_WIFI_PASSWORD;
static const char* host = IOT_CONFIG_IOTHUB_FQDN;
static const char* mqtt_broker_uri = "mqtts://" IOT_CONFIG_IOTHUB_FQDN;
static const char* device_id = IOT_CONFIG_DEVICE_ID;
static const int mqtt_port = AZ_IOT_DEFAULT_MQTT_CONNECT_PORT;

// Memory allocated for the sample's variables and structures.
static esp_mqtt_client_handle_t mqtt_client;
static az_iot_hub_client client;

static char mqtt_client_id[128];
```

```
static char mqtt_client_id[128];
static char mqtt_username[128];
static char mqtt_password[200];
static uint8_t sas_signature_buffer[256];
static unsigned long next_telemetry_send_time_ms = 0;
static char telemetry_topic[128];
static uint32_t telemetry_send_count = 0;
static String telemetry_payload = "{}";

/*||||||||||||||||||||||||>PLCS_VARIABLE|||||||||||||||||||||||||||||*/
uint16_t TT1_PLC1 =0;
uint16_t ST1_PLC1 = 0;
uint16_t LC1_Virtual=0;
uint16_t FT1_PLC2 = 0;
uint16_t CE1_PLC2 = 0;
uint16_t WT1_PLC2 = 0;
uint16_t PA1_PLC2 = 0;
uint16_t PA2_PLC2 = 0;
uint16_t Fan=0;
uint16_t Stirrer=0;
uint16_t FC1=0;
uint16_t LC1=0;
uint16_t J1=0;
uint16_t TT2_PLC3 = 0;
uint16_t LT1_PLC3 = 0;
uint16_t Pump1=0;
uint16_t Pump2=0;
/*||||||||||||||||||||||||PLCS_VARIABLE<|||||||||||||||||||||||||||||*/

/*||||||||||||>Create an instances of the ModbusMaster class|||||||||||||*/
ModbusMaster Slave1;
ModbusMaster Slave2;
```

```
/*||||||||||||>Create an instances of the ModbusMaster class|||||||||||||*/
ModbusMaster Slave1;
ModbusMaster Slave2;
ModbusMaster Slave3;
/*||||||||||||Create an instances of the ModbusMaster class<|||||||||||||*/


#define INCOMING_DATA_BUFFER_SIZE 128
static char incoming_data[INCOMING_DATA_BUFFER_SIZE];

// Auxiliary functions
#ifndef IOT_CONFIG_USE_X509_CERT
static AzIoTSasToken sasToken(
    &client,
    AZ_SPAN_FROM_STR(IOT_CONFIG_DEVICE_KEY),
    AZ_SPAN_FROM_BUFFER(sas_signature_buffer),
    AZ_SPAN_FROM_BUFFER(mqtt_password));
#endif // IOT_CONFIG_USE_X509_CERT

static void connectToWiFi()
{
  Logger.Info("Connecting to WIFI SSID " + String(ssid));

  WiFi.mode(WIFI_STA);
  WiFi.disconnect();
  delay(100);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED)
  {
    delay(500);
    Serial.print(".");
  }
```

```
  WiFi.mode(WIFI_STA);
  WiFi.disconnect();
  delay(100);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED)
  {
    delay(500);
    Serial.print(".");
  }

  Serial.println("");

  Logger.Info("WiFi connected, IP address: " + WiFi.localIP().toString());
}

static void initializeTime()
{
  Logger.Info("Setting time using SNTP");

  configTime(GMT_OFFSET_SECS, GMT_OFFSET_SECS_DST, NTP_SERVERS);
  time_t now = time(NULL);
  while (now < UNIX_TIME_NOV_13_2017)
  {
    delay(500);
    Serial.print(".");
    now = time(nullptr);
  }
  Serial.println("");
  Logger.Info("Time initialized!");
}
```

```
void receivedCallback(char* topic, byte* payload, unsigned int length)
{
  Logger.Info("Received [");
  Logger.Info(topic);
  Logger.Info("]: ");
  for (int i = 0; i < length; i++)
  {
    Serial.print((char)payload[i]);
  }
  Serial.println("");
}

static esp_err_t mqtt_event_handler(esp_mqtt_event_handle_t event)
{
  switch (event->event_id)
  {
    int i, r;

    case MQTT_EVENT_ERROR:
      Logger.Info("MQTT event MQTT_EVENT_ERROR");
      break;
    case MQTT_EVENT_CONNECTED:
      Logger.Info("MQTT event MQTT_EVENT_CONNECTED");

      r = esp_mqtt_client_subscribe(mqtt_client, AZ_IOT_HUB_CLIENT_C2D_SUBSCRIBE_TOPIC, 1);
      if (r == -1)
      {
        Logger.Error("Could not subscribe for cloud-to-device messages.");
      }
      else
      {
```

```
      {
        Logger.Info("Subscribed for cloud-to-device messages; message id:" + String(r));
      }

      break;
    case MQTT_EVENT_DISCONNECTED:
      Logger.Info("MQTT event MQTT_EVENT_DISCONNECTED");
      break;
    case MQTT_EVENT_SUBSCRIBED:
      Logger.Info("MQTT event MQTT_EVENT_SUBSCRIBED");
      break;
    case MQTT_EVENT_UNSUBSCRIBED:
      Logger.Info("MQTT event MQTT_EVENT_UNSUBSCRIBED");
      break;
    case MQTT_EVENT_PUBLISHED:
      Logger.Info("MQTT event MQTT_EVENT_PUBLISHED");
      break;
    case MQTT_EVENT_DATA:
      Logger.Info("MQTT event MQTT_EVENT_DATA");

      for (i = 0; i < (INCOMING_DATA_BUFFER_SIZE - 1) && i < event->topic_len; i++)
      {
        incoming_data[i] = event->topic[i];
      }
      incoming_data[i] = '\0';
      Logger.Info("Topic: " + String(incoming_data));

      for (i = 0; i < (INCOMING_DATA_BUFFER_SIZE - 1) && i < event->data_len; i++)
      {
        incoming_data[i] = event->data[i];
      }
```

```
      incoming_data[i] = '\0';
      Logger.Info("Data: " + String(incoming_data));

      break;
    case MQTT_EVENT_BEFORE_CONNECT:
      Logger.Info("MQTT event MQTT_EVENT_BEFORE_CONNECT");
      break;
    default:
      Logger.Error("MQTT event UNKNOWN");
      break;
  }

  return ESP_OK;
}

static void initializeIoTHubClient()
{
  az_iot_hub_client_options options = az_iot_hub_client_options_default();
  options.user_agent = AZ_SPAN_FROM_STR(AZURE_SDK_CLIENT_USER_AGENT);

  if (az_result_failed(az_iot_hub_client_init(
          &client,
          az_span_create((uint8_t*)host, strlen(host)),
          az_span_create((uint8_t*)device_id, strlen(device_id)),
          &options)))
  {
    Logger.Error("Failed initializing Azure IoT Hub client");
    return;
  }

  size_t client_id_length;
  if (az_result_failed(az_iot_hub_client_get_client_id(
          &client, mqtt_client_id, sizeof(mqtt_client_id) - 1, &client_id_length)))
```

```
  size_t client_id_length;
  if (az_result_failed(az_iot_hub_client_get_client_id(
          &client, mqtt_client_id, sizeof(mqtt_client_id) - 1, &client_id_length)))
  {
    Logger.Error("Failed getting client id");
    return;
  }

  if (az_result_failed(az_iot_hub_client_get_user_name(
          &client, mqtt_username, sizeofarray(mqtt_username), NULL)))
  {
    Logger.Error("Failed to get MQTT clientId, return code");
    return;
  }

  Logger.Info("Client ID: " + String(mqtt_client_id));
  Logger.Info("Username: " + String(mqtt_username));
}

static int initializeMqttClient()
{
#ifndef IOT_CONFIG_USE_X509_CERT
  if (sasToken.Generate(SAS_TOKEN_DURATION_IN_MINUTES) != 0)
  {
    Logger.Error("Failed generating SAS token");
    return 1;
  }
#endif

  esp_mqtt_client_config_t mqtt_config;
  memset(&mqtt_config, 0, sizeof(mqtt_config));
  mqtt_config.uri = mqtt_broker_uri;
```

```
  mqtt_config.port = mqtt_port;
  mqtt_config.client_id = mqtt_client_id;
  mqtt_config.username = mqtt_username;

#ifdef IOT_CONFIG_USE_X509_CERT
  Logger.Info("MQTT client using X509 Certificate authentication");
  mqtt_config.client_cert_pem = IOT_CONFIG_DEVICE_CERT;
  mqtt_config.client_key_pem = IOT_CONFIG_DEVICE_CERT_PRIVATE_KEY;
#else // Using SAS key
  mqtt_config.password = (const char*)az_span_ptr(sasToken.Get());
#endif

  mqtt_config.keepalive = 30;
  mqtt_config.disable_clean_session = 0;
  mqtt_config.disable_auto_reconnect = false;
  mqtt_config.event_handle = mqtt_event_handler;
  mqtt_config.user_context = NULL;
  mqtt_config.cert_pem = (const char*)ca_pem;

  mqtt_client = esp_mqtt_client_init(&mqtt_config);

  if (mqtt_client == NULL)
  {
    Logger.Error("Failed creating mqtt client");
    return 1;
  }

  esp_err_t start_result = esp_mqtt_client_start(mqtt_client);

  if (start_result != ESP_OK)
  {
    Logger.Error("Could not start mqtt client; error code:" + start_result);
```

```
    return 1;
  }
  else
  {
    Logger.Info("MQTT client started");
    return 0;
  }
}

/*
 * @brief           Gets the number of seconds since UNIX epoch until now.
 * @return uint32_t Number of seconds.
 */
static uint32_t getEpochTimeInSecs() { return (uint32_t)time(NULL); }

static void establishConnection()
{
  connectToWiFi();
  initializeTime();
  initializeIoTHubClient();
  (void)initializeMqttClient();
}

static void generateTelemetryPayload()
{
    StaticJsonDocument<256> jsonDocument; // Adjust the size as needed

    // Clear previous data from JSON document
    jsonDocument.clear();

    // Add telemetry data to JSON document
    jsonDocument["TT1_PLC1"] = TT1_PLC1;
```

```
static void generateTelemetryPayload()
{
    StaticJsonDocument<256> jsonDocument; // Adjust the size as needed

    // Clear previous data from JSON document
    jsonDocument.clear();

    // Add telemetry data to JSON document
    jsonDocument["TT1_PLC1"] = TT1_PLC1;
    jsonDocument["ST1_PLC1"] = ST1_PLC1;
    jsonDocument["FT1_PLC2"] = FT1_PLC2;
    jsonDocument["WT1_PLC2"] = WT1_PLC2;
    jsonDocument["TT2_PLC3"] = TT2_PLC3;
    jsonDocument["Fan"] = Fan;
    jsonDocument["Stirrer"] = Stirrer;
    jsonDocument["FC1"] = FC1;
    jsonDocument["LC1"] = LC1;
    jsonDocument["Pump1"] = Pump1;
    jsonDocument["Pump2"] = Pump2;
    jsonDocument["J1"] = J1;


    // Serialize the JSON document to a string
    String jsonString;
    serializeJson(jsonDocument, jsonString);

    // Convert String to char* for MQTT publish
    telemetry_payload = jsonString;


}
```

```
static void sendTelemetry()
{
  Logger.Info("Sending telemetry ...");

  // The topic could be obtained just once during setup,
  // however if properties are used the topic need to be generated again to reflect the
  // current values of the properties.
  if (az_result_failed(az_iot_hub_client_telemetry_get_publish_topic(
          &client, NULL, telemetry_topic, sizeof(telemetry_topic), NULL)))
  {
    Logger.Error("Failed az_iot_hub_client_telemetry_get_publish_topic");
    return;
  }

  generateTelemetryPayload();

  if (esp_mqtt_client_publish(
          mqtt_client,
          telemetry_topic,
          (const char*)telemetry_payload.c_str(),
          telemetry_payload.length(),
          MQTT_QOS1,
          DO_NOT_RETAIN_MSG)
      == 0)
  {
    Logger.Error("Failed publishing");
  }
  else
  {
    Logger.Info("Message published successfully");
  }
}
```

```
// Arduino setup and loop main functions.

void setup() {

  /*|||||||||||||||||Initialize Speed Rate Of Communication|||||||||||||||||*/
  Serial.begin(115200);
  Serial_mod.begin(BAUD_RATE);

  /*|||||||||||||||||||||||||>Slaves BEGINS|||||||||||||||||||||||||||||||*/
  // Modbus slave1 device
  Slave1.begin(PLC_SLAVE1_Station, Serial_mod);
  //  Modbus slave2 device
  Slave2.begin(PLC_SLAVE2_Station, Serial_mod);
  //Modbus slave3 device
  Slave3.begin(PLC_SLAVE3_Station, Serial_mod);
  /*|||||||||||||||||||||||||Slaves BEGINS<|||||||||||||||||||||||||||||||*/



  ////////////////AZURE IOT HUB ESP32
  establishConnection();


}


void loop()
{
  if (WiFi.status() != WL_CONNECTED)
  {
    connectToWiFi();
  }
```

```
void loop()
{
  if (WiFi.status() != WL_CONNECTED)
  {
    connectToWiFi();
  }
#ifndef IOT_CONFIG_USE_X509_CERT
  else if (sasToken.IsExpired())
  {
    Logger.Info("SAS token expired; reconnecting with a new one.");
    (void)esp_mqtt_client_destroy(mqtt_client);
    initializeMqttClient();
  }
#endif
  else if (millis() > next_telemetry_send_time_ms)
  {
    sendTelemetry();
    next_telemetry_send_time_ms = millis() + TELEMETRY_FREQUENCY_MILLISECS;
  }




  /*|||||||||||||||||||Read Data from PlCS to PLCS|||||||||||||||||||||||*/
  /*|||||||||||| Read analog sensors signals  from PLC1||||||||||||*/
    //Read TT1 and St1  from register D0003 & D0004

  uint8_t result_0 = Slave1.readInputRegisters(3, 3);
  if (result_0 == Slave1.ku8MBSuccess) {
    TT1_PLC1 = Slave1.getResponseBuffer(0);
    ST1_PLC1 = Slave1.getResponseBuffer(1);
```

```
  if (result_0 == Slave1.ku8MBSuccess) {
    TT1_PLC1 = Slave1.getResponseBuffer(0);
    ST1_PLC1 = Slave1.getResponseBuffer(1);
    LC1_Virtual= Slave1.getResponseBuffer(2);
  }

  delay(25);
  /*|||||||||||||||||||| Read data from PLC2|||||||||||||||||||||||||*/
  /*|||||||||||| Read analog sensors signals from plc2|||||||||||*/
    //Read Wt1 from register D00014

  uint8_t result_1 = Slave2.readInputRegisters(14, 1);
  if (result_1 == Slave2.ku8MBSuccess) {
    WT1_PLC2 = Slave2.getResponseBuffer(0);
  }

  // Read FT1 from register D0008
  uint8_t result_2 = Slave2.readInputRegisters(8, 1);
  if (result_2 == Slave2.ku8MBSuccess) {
    FT1_PLC2 = Slave2.getResponseBuffer(0);
  }

  // Read CE1 from register D0002
  uint8_t result_3 = Slave2.readInputRegisters(2, 1);
  if (result_3 == Slave2.ku8MBSuccess) {
    CE1_PLC2 = Slave2.getResponseBuffer(0);
  }

  /*|||||||||||| Read analog actuators signals from plc2 ||||||||||||*/
  // Read  Fan=0; Stirrer=0; FC1=0; LC1=0  from  (9-12)
  uint8_t result_4 = Slave2.readInputRegisters(9, 4);
  if (result_4 == Slave2.ku8MBSuccess) {
```

```
 // Read  Fan=0; Stirrer=0; FC1=0; LC1=0  from  (9-12)
 uint8_t result_4 = Slave2.readInputRegisters(9, 4);
 if (result_4 == Slave2.ku8MBSuccess) {
   Fan = Slave2.getResponseBuffer(0);
   Stirrer = Slave2.getResponseBuffer(1);
   FC1 = Slave2.getResponseBuffer(2);
   LC1 = Slave2.getResponseBuffer(3);
  }
 delay(25);


uint8_t result_71 = Slave2.readInputRegisters(30, 1);
 if (result_71 == Slave2.ku8MBSuccess) {
   Pump1 = Slave2.getResponseBuffer(0);
  }
 delay(5);
  uint8_t result_72 = Slave2.readInputRegisters(35, 1);
 if (result_72 == Slave2.ku8MBSuccess) {
   Pump2 = Slave2.getResponseBuffer(0);
  }
 delay(5);
  uint8_t result_73 = Slave2.readInputRegisters(40, 1);
 if (result_73 == Slave2.ku8MBSuccess) {
   J1 = Slave2.getResponseBuffer(0);
  }

/*
 Serial.println(" monitoring for plc2 data   ^^^^^_____^^^^^ " );
 Serial.print(" J1: " );Serial.print( J1 );Serial.print(" |||| ");
 Serial.print(" Pump1: " );Serial.print( Pump1 );Serial.print(" |||| ");
 Serial.print(" Pump2: " );Serial.print( Pump2 );Serial.print("..................");

*/
```

```
*/
 delay(25);

 /*|||||||||||||||||||| Read data from PLC3|||||||||||||||||||||||||*/
 /*|||||||||||| Read analog sensors signals from plc3 ||||||||||||*/
 // Read TT2 from register D0001

 uint8_t result_5 = Slave3.readInputRegisters(1, 1);
 if (result_5 == Slave3.ku8MBSuccess) {
   TT2_PLC3  = Slave3.getResponseBuffer(0);
 }

 // Read LT1 from register D0009
 uint8_t result_6 = Slave3.readInputRegisters(9, 1);
 if (result_6 == Slave3.ku8MBSuccess) {
   LT1_PLC3  = Slave3.getResponseBuffer(0);
 }

 delay(25);

 /*|||||||||||||||||||||Write Data from PlCS to PLCS|||||||||||||||||||||||*/
 /*|||||||||||||||Write analog actuator to PLC 1|||||||||||||||||*/
 /*_____________________________________________*/
 // Wrtie  [Fan]  to [PLC1 in M0000]:
 //Wrtie  [ Stirrer]  to [PLC1 in M0001]:

 uint8_t result_7 = Slave1.writeSingleRegister(1 , Stirrer);
 delay(25);
 uint8_t result_8 = Slave1.writeSingleRegister(0 , Fan);
 delay(25);
```

```
  /*|||||||||||||||||||||Write analog sensors data to PLC 2|||||||||||||||||||||*/
  /*_____________________________________________*/
    //Wrtie  [TT1_PLC1]  to [PLC2 in M0000]:
    //Wrtie  [ ST1_PLC1]  to [PLC2 in M0001]:
    //Wrtie  [TT2_PLC3]  to [PLC2 in M0002]:
    //Wrtie  [ LT1_PLC3]  to [PLC2 in M0003]:

  uint8_t result_9 = Slave2.writeSingleRegister(0 ,TT1_PLC1 );
  delay(25);
  uint8_t result_10 = Slave2.writeSingleRegister(1  , ST1_PLC1 );
  delay(25);
  uint8_t result_11 = Slave2.writeSingleRegister(2  , TT2_PLC3 );
  delay(25);
  uint8_t result_12 = Slave2.writeSingleRegister(3  , LT1_PLC3 );
  delay(25);


  /*|||||||||||||||||||||Write  analog actuator  to PLC 3|||||||||||||||||||||||*/
  /*_____________________________________________*/
    //Wrtie [FC1]  to [PLC3 in M0000]:
    //Wrtie  [LC1]  to [PLC3 in M0001]:

  uint8_t result_21 = Slave3.writeSingleRegister(1  , LC1 );
  delay(25);
  uint8_t result_22 = Slave3.writeSingleRegister(0  , FC1 );
  delay(25);
```

```
  /*||||||||||||||||||||||Write analog sensors data to PLC 2||||||||||||||||||||||*/
  /*_____________________________________________*/
    //Wrtie  [TT1_PLC1]  to [PLC2 in M0000]:
    //Wrtie  [ ST1_PLC1]  to [PLC2 in M0001]:
    //Wrtie  [TT2_PLC3]  to [PLC2 in M0002]:
    //Wrtie  [ LT1_PLC3]  to [PLC2 in M0003]:

  uint8_t result_9 = Slave2.writeSingleRegister(0 ,TT1_PLC1 );
  delay(25);
  uint8_t result_10 = Slave2.writeSingleRegister(1  , ST1_PLC1 );
  delay(25);
  uint8_t result_11 = Slave2.writeSingleRegister(2  , TT2_PLC3 );
  delay(25);
  uint8_t result_12 = Slave2.writeSingleRegister(3  , LT1_PLC3 );
  delay(25);


  /*||||||||||||||||||||||Write  analog actuator  to PLC 3||||||||||||||||||||||*/
  /*_____________________________________________*/
    //Wrtie [FC1]  to [PLC3 in M0000]:
    //Wrtie  [LC1]  to [PLC3 in M0001]:

  uint8_t result_21 = Slave3.writeSingleRegister(1  , LC1 );
  delay(25);
  uint8_t result_22 = Slave3.writeSingleRegister(0  , FC1 );
  delay(25);

  delay(100);

}
```

Project Video