Measurement System Automation:

Software Design Document

9/26/25

Version 2

Team Name: Thermo-Gen

Sponsor: Steve Miller

Mentor: Jeevana Swaroop Kalapala

Team Members: Olivia Vester, Kameron Napier, Gareth Carew

THERMO-GEN

Table of Contents

1. Introduction

The industry of material manufacturing, particularly insulating material manufacturing, is both expansive and critical. The industry encompasses a wide range of stakeholders such as government entities like the United States military and NASA, and many private manufacturers. A defining characteristic of this industry is its focus on creating materials that provide thermal insulation/protection for specialized equipment, such as heat-resistant uniforms and space vehicles, those of which are integral in research fields like aerospace research. Although often overlooked by the general public, this industry plays a vital role in allowing for essential activities like space exploration, firefighting, and operating in hazardous environments. Our client and sponsor, HeetShield, operates directly within the insulating materials industry, maintaining contracting with the National Forest Service, the US Marine Corps, NASA, and several other agencies. The company specializes in the in-house development and testing of advanced insulative materials. One such material is their Opacified Fibrous Insulation (aka OFI), which is capable of withstanding excessive heat transfer up to 3000°F. HeetShield conducts its operations from a specialized facility based in Flagstaff, Arizona.

The company primarily advances its work through research and development grants that support a focused team of specialists dedicated to material construction and testing. Testing is carried out using a custom-built apparatus that is designed to replicate extreme environmental conditions in a controlled setting. Building on this foundation, we are creating a software application designed to automate the process of achieving precise environmental conditions for testing. This system will read in data from the gauges on the apparatus and adjust the controls for temperature and pressure based on the desired values specified by the user. This system will also have an intuitive user interface that will benefit the user and the testing process as a whole. We are working within a constrained environment that poses the following obstacles: limited physical space for a computer to run our system, limited budget for upgrading to parts more compatible with our system, limited time due to system-component delivery delays, and adapting to new technical environments. Despite these challenges, our system will ultimately meet the goal of allowing the operator to digitally specify the conditions of the testing environment in conjunction with condition-altering machinery that adjusts settings according to user input and data read in.

To give a brief overview, our software will be an application that will be downloaded onto the laboratory computer that is used for materials testing. Once our application is downloaded, the computer will be attached to the power supply and the pressure valve of the testing apparatus. Once all components are connected, the software can then be powered up and used as follows: The technician will open the user interface and input the desired pressure and temperature they want to test with. After inputting the data, the system will receive data from the gauges on the apparatus to get the current state of the environment. From there, the software will work via its connections to the power supply and pressure valve to adjust the pressure and temperature to the desired pressure values that the technician inputted into the system. Once the desired values are met by the testing environment, a window notification will alert the technician via a pop-up and a sound. If the conditions cannot be met by the environment, the technician will also be alerted of that issue.

Overall, the goal of this system we are building is to cut down the amount of time the technician needs to dedicate to testing. Right now, the technician has to manually control a power supply and pressure valve to control the environmental elements which can take hours and lack the precision that they may need in the testing environment. With our system, this task will be eliminated from a manual perspective; the technician can set the parameters and walk away to handle another task and the system will alert when it is ready.

To outline how we are making this system, the details of our implementation overview are specified in the next section.

2. Implementation Overview

This project implements a computer based control system designed to regulate temperature and pressure within a controlled environment. Temperature input is provided through thermocouples attached to the inside of the test environment, while pressure input comes from a rs232 pressure gauge. To adjust these values, the system controls external hardware: a Sorensen DCS80-13E power supply, which regulates heating, and the Bürkert Type 3280 valve which moderates the pressure. The combined setup provides a closed loop feedback system that continuously drives temperature and pressure towards user defined targets while maintaining safe operating conditions.

The user interface is built using the Qt framework via PySide6, chosen for its GUI support and ability to create a responsive and professional desktop application. The interface allows users to enter desired setpoints, select measurement units, and toggle between manual and automatic control modes. The GUI layout and logic are defined in a .ui file and translated into Python code, ensuring consistency and maintainability of the interface design.

To handle background operations alongside user interactions, the system uses QtAsyncio to integrate Python’s AsyncIO event loop with Qt’s signal and slot model. Continuous monitoring and adjustment tasks run in a separate worker thread implemented with QThread, ensuring that the interface remains responsive while calculations and hardware adjustments take place. Adjustment logic incrementally brings temperature and pressure closer to their targets with large correction being applied when values are far from the setpoint and finer corrections being used as the system approaches the desired range. This tiered adjustment strategy balances responsiveness, stability and timeliness.

Real time feedback is provided through Windows Toast notifications. Successful stabilization generates a success message while failures trigger alerts. Failures are defined as either exceeding a maximum time limit without reaching the target to prevent it from getting stuck or entering an unsafe environment where temperature or pressure exceeds defined thresholds. This immediate notification system ensures that operators are kept aware of system status without needing to monitor the GUI continuously.

In summary, the implementation brings together a modern GUI built on PySide6, asynchronous and threaded background processing with QtAsyncio and QThread, and safety oriented real time feedback via Windows Toast notifications. When coupled with thermocouple input, the Sorensen DCS80-13E power supply, and the Bürkert Type 3280 valve the result is a closed loop system that provides reliable safe and user friendly control of temperature and pressure.

With the ins and outs of the hardware setup detailed here, we can move into discussing the software portion in more detail.

3. Architectural Overview

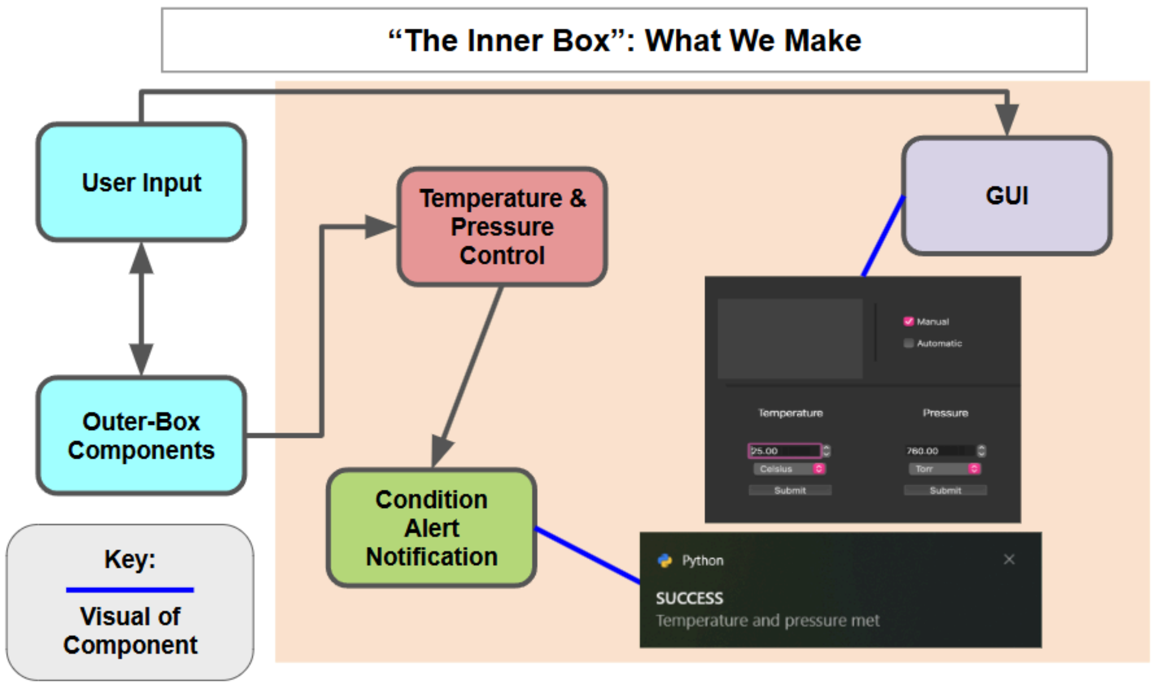
For our platform, we have a Qt user interface that consists of a single pane that lets the user control whether the system actively controls the temperature & pressure or not, input pressure and temperature values, as well as displays the current temperature and pressure values. Our user interface communicates with our main controller to obtain the data to display and to send the data to the other components.

Our main controller routes information from one place to another. It takes in information from the UI and routes it to the temperature and pressure writing components as well as the notification system if needed. It takes information from the temperature and pressure reading components and routes it to the UI as well as writing it out to a CSV file for use after.

We have a general purpose notification system that will display a popup window as well as create windows notification with sound using windows_toasts to alert the user when the desired conditions are met or if something went wrong.

The temperature reading component communicates with the DAQ through USB using NI's NI-DAQmx package to acquire the temperature data. The temperature writing component communicates with the power supply through a USB to GPIB cable using PyVISA to supply the PSU with the required voltage to control the temperature. The pressure reading component communicates with a pressure gauge through an RS232 to USB cable using PyVISA to acquire the pressure data.

Below are two diagrams that depict the hardware components and the software components of the system visually in a high-level way. The purpose of including these graphics is to outline the major moving parts and how they connect to each other.



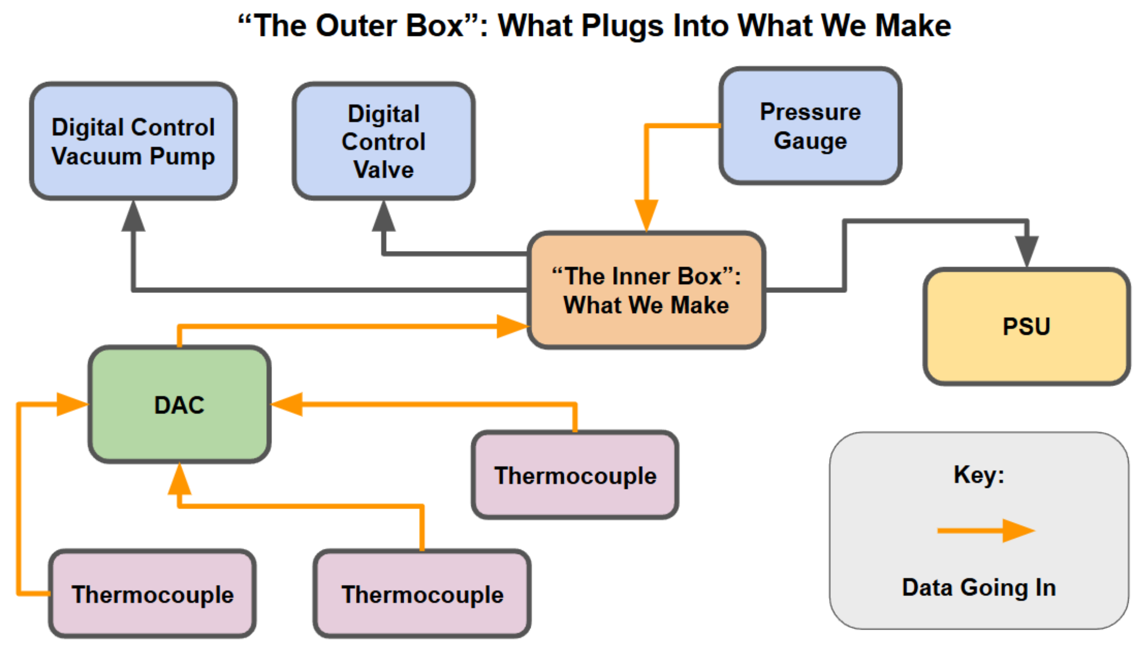
[✓ Manual] Automatic

Temperature: 25.00 Celsius

Pressure: 760.00 Torr

SUCCESS
Temperature and pressure met

3.1 Figure 1: Display of the inner elements that compose the software that we make.


"The Outer Box": What Plugs Into What We Make
Digital Control Vacuum Pump
Digital Control Valve
Pressure Gauge
"The Inner Box": What We Make
PSU
DAC
Thermocouple
Thermocouple
Thermocouple
Key:
Data Going In

3.1 Figure 2: Display of the outer elements that influence the software that we make.

4. Module & Interface Descriptions

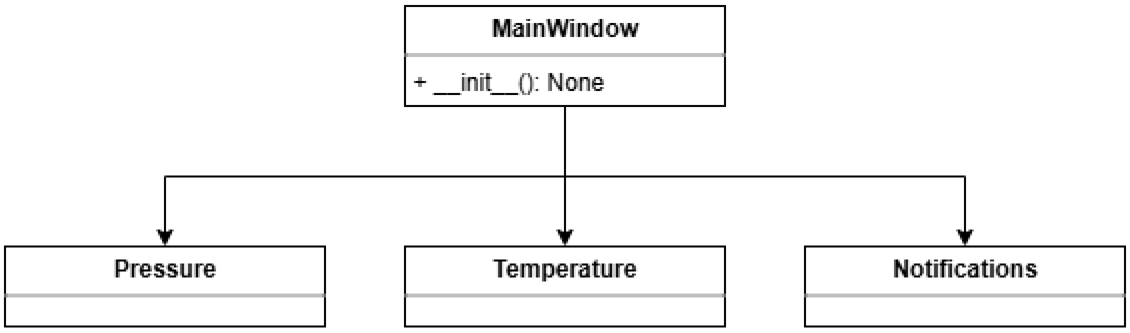
In the section we split our system into seven main modules that are the major components that make up the entire thing: the main controller, reading temperature, writing temperature, reading pressure, writing pressure, notification.

4.1 Module 1: Main Controller

4.1.1 - Description

The main controller module coordinates the overall operation of the system. It initializes the application, sets up the main window, and connects the user interface to the background processes. This module ensures that user inputs are passed correctly to the worker logic and that results are displayed in the interface. Within the architecture, it acts as a central hub, linking the GUI, background processes, and notification systems together. Its role is to maintain application flow rather than perform calculations or hardware interactions directly.

4.1.2 - Diagrams



4.1.2 Figure 1: Main Controller UML Diagram

4.1.3 - Services

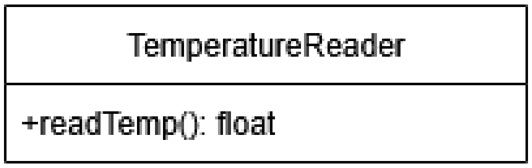
The main controller is implemented through the MainWindow class. Its primary interface is the constructor __init__, which initializes the UI components by calling setupUi. It also defines stop_event_loops(), which shuts down both the Qt and AsyncIO event loops when the application is closed. Together, these methods ensure proper startup, runtime management, and clean shutdown of the application.

4.2 Module 2: Read Temperature

4.2.1 - Description

The temperature read module's purpose is to read in the temperature data in real time from the DAQ. This is done by using NI's NI-DAQmx package. The module transmits all of the data it reads to the main controller for use elsewhere in the program.

4.2.2 - Diagrams



4.2.2 Figure 1: Temperature Read UML Diagram

4.2.3 - Services

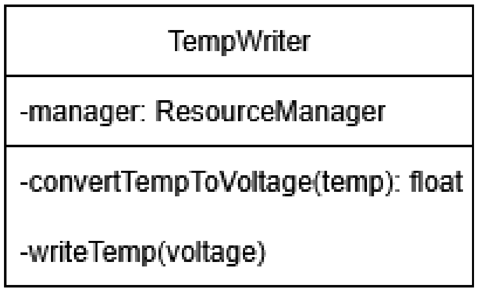
The temperature read module runs the readTemp() function which collects data in real time through the use of the API provided by NI. This data is then sent to the main controller for use in other parts of the program such as display and comparison for adjusting the write temperature module.

4.3 Module 3: Write Temperature

4.3.1 - Description

The temperature write module's purpose is to communicate with the PSU through a GPIB cable with the use of the PyVISA package to control the PSU's voltage, which in turn determines the temperature of the test chamber. The temperature write module gets a temperature value from the main controller and performs a calculation to convert it into a voltage value for the PSU.

4.3.2 - Diagrams



4.3.2 Figure 1: Temperature Write UML Diagram

4.3.3 - Services

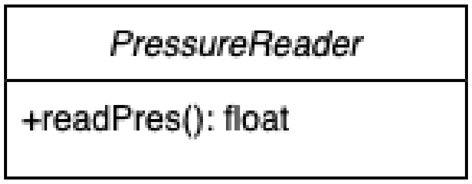
The temperature write module uses a ResourceManager from PyVISA to connect with the GPIB port. Then whenever the module is given a new temperature, it converts that temperature to the corresponding voltage and writes the voltage to the PSU through the GPIB port.

4.4 Module 4: Read Pressure

4.4.1 - Description

The purpose of this module for reading pressure is to read in the live pressure data from the digitized pressure gauges. This will be done via the PyVISA package that is created to control and manage instruments such as the pressure gauge we are using here. Like with temperature, this data read will be transmitted to the main controller where it can be utilized in the writing module.

4.4.2 - Diagrams



4.4.2 Figure 1: Pressure Read UML Diagram

4.4.3 - Services

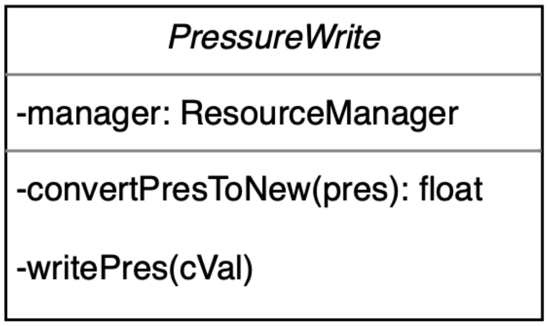
This module will provide our system with the live pressure data. This will utilize a simple readPres() function that will take in a float that will be run through the system as follows: if the live value is in the correct range specified by the user, then no work needs to be done, other wise, the value will be outside the range and the write function will take over.

4.5 Module 5: Write Pressure

4.5.1 - Description

The purpose of this module is to use the user-input range for pressure and convert the value to a new pressure value that fits in the range. The gauge we are using, [insert name here], works in conjunction with the pressure valve, [insert name here], to adjust the pressure chamber to reach the desired pressure value. Once the pressure has met the range requirements, the write function will signal the notification module to notify the user of the changes made.

4.5.2 - Diagrams



4.5.2 Figure 1: Pressure Write UML Diagram

4.5.3 - Services

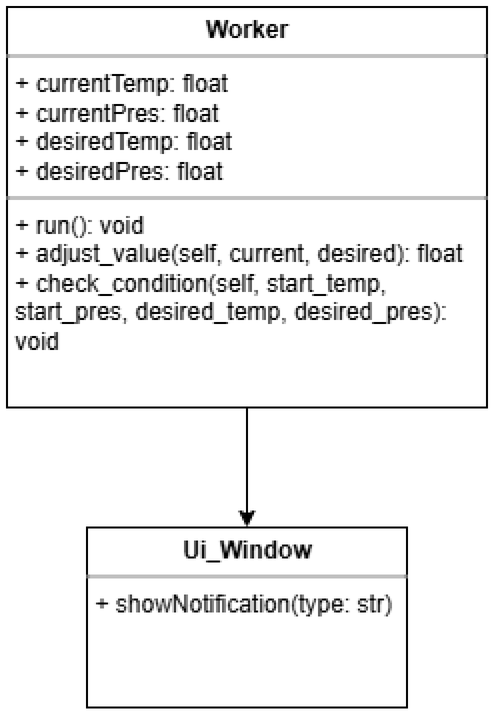
The pressure writing module will function with a resource manager, similar to that of the temperature writing module; the manager will note the values of the resources (temperature, pressure) and send them to a function convertPresToNew that will send a signal to the pressure value to adjust the pressure to increase or decrease the value which will be a float value. This will then be utilized by the write function to finalize the pressure value to notify the user of the current written state of the pressure.

4.6 Module 6: Notifications

4.6.1 - Description

The notifications module is dedicated to providing real time feedback to the user regarding the state of the system. It acts as the communication channel between the program and user by using Windows Toast notifications which ensures alerts are visible and audible. Specifically, the notification is responsible for informing the operator when temperature and pressure targets have been successfully reached, when the system fails to stabilize within a reasonable amount of time, or when unsafe conditions occur. In the overall architecture, it serves as the feedback layer while the main controller and the read/write modules handle regulation which the notifications translate the results into clear user alerts. This separation of concerns keeps control logic focused on adjustments and assigns communication to this module.

4.6.2 - Diagrams


Worker
+ currentTemp: float
+ currentPres: float
+ desiredTemp: float
+ desiredPres: float
+ run(): void
+ adjust_value(self, current, desired): float
+ check_condition(self, start_temp, start_pres, desired_temp, desired_pres): void
Ui_Window
+ showNotification(type: str)

4.6.2 Figure 1: Notifications UML Diagram

4.6.3 - Services

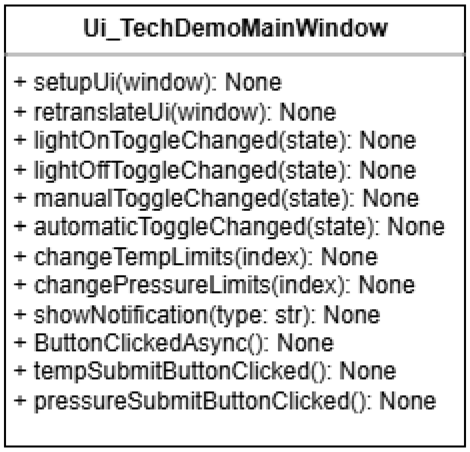
The module operates through two key components, the Worker class and the showNotifications method. The Worker class runs in a separate thread and emits a finished signal with one of three results, success, unsafe or timeout. The showNotification method in the main window uses these signals and displays toast alerts with the corresponding status message. Together, these interfaces ensure that the system's outcomes are clearly communicated to the user.

4.7 Module 7: User Interface

4.7.1 - Description

The user interface module provides the primary point of interaction between the operator and the system. It is responsible for displaying the current temperature and pressure values, accepting user input for desired environments, and allowing the operator to toggle between manual and automatic control modes. The UI is implemented using the Qt framework through PySide6. Within the larger architecture the UI module acts as both a data entry and a data presentation layer. Its design emphasizes usability and clarity, ensuring that system operations can be performed safely and efficiently.

4.7.2 - Diagrams



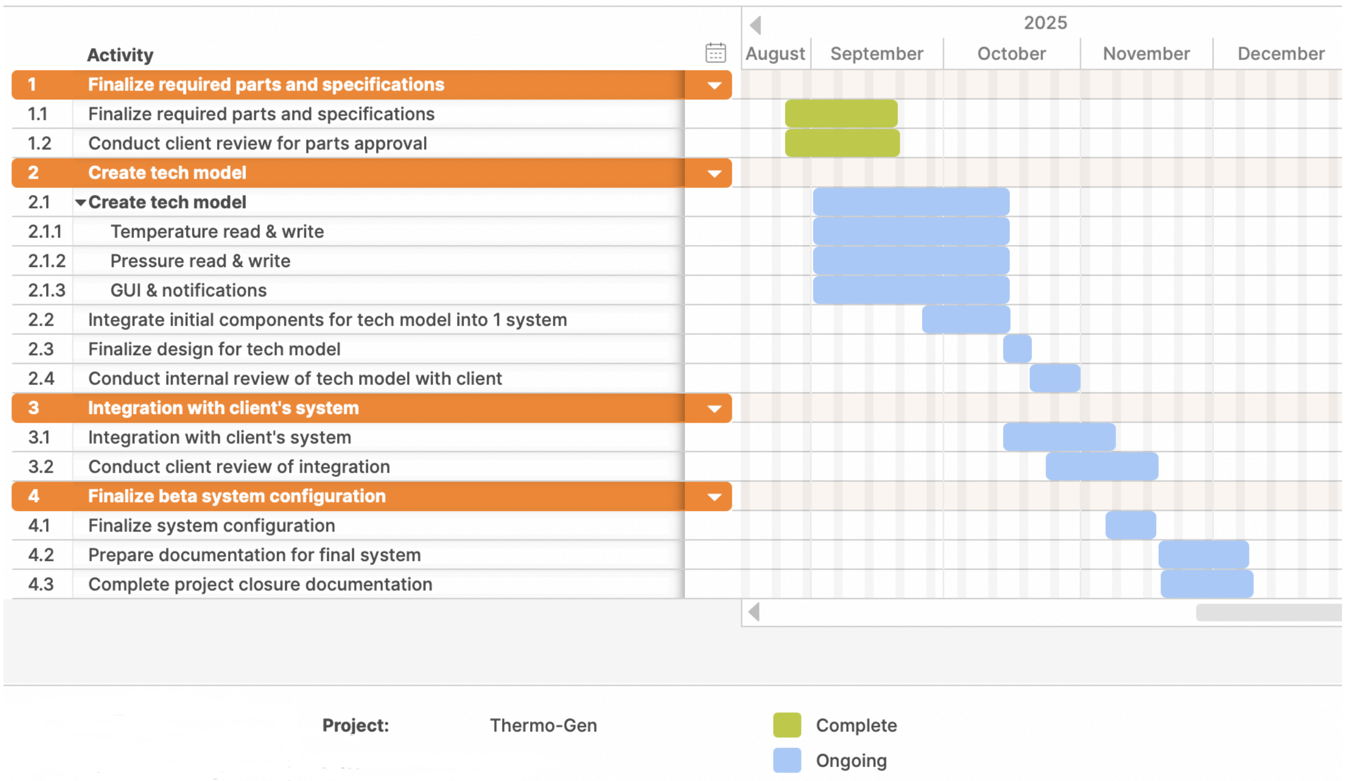
```
Ui_TechDemoMainWindow
---------------------------------------------
+ setupUi(window): None
+ retranslateUi(window): None
+ lightOnToggleChanged(state): None
+ lightOffToggleChanged(state): None
+ manualToggleChanged(state): None
+ automaticToggleChanged(state): None
+ changeTempLimits(index): None
+ changePressureLimits(index): None
+ showNotification(type: str): None
+ ButtonClickedAsync(): None
+ tempSubmitButtonClicked(): None
+ pressureSubmitButtonClicked(): None
```

4.7.3 - Services

The UI module is implemented through the Ui_TechDemoMainWindow class, which provides methods for both setup and interaction. The setupUi method initializes and arranges all GUI elements, while retranslateUi sets the display text and labels. User interaction is handled through methods that manage toggle states for light control and manual or automatic modes, as well as methods that adjust input ranges when different temperature or pressure units are selected. The interface also includes functions to submit desired values, create background processing adjustments, and display toast notifications that inform the user of success, failure, or unsafe conditions. Together these methods form a clear and cohesive public interface that allows the operator to input desired parameters, monitor results, and receive feedback within the application.

5. Implementation Plan

Going forward in finishing our project, we have created the following timeline dedicated to keeping track of our path to completing the system:



	Activity
1	**Finalize required parts and specifications**
1.1	Finalize required parts and specifications
1.2	Conduct client review for parts approval
2	**Create tech model**
2.1	**Create tech model**
2.1.1	Temperature read & write
2.1.2	Pressure read & write
2.1.3	GUI & notifications
2.2	Integrate initial components for tech model into 1 system
2.3	Finalize design for tech model
2.4	Conduct internal review of tech model with client
3	**Integration with client's system**
3.1	Integration with client's system
3.2	Conduct client review of integration
4	**Finalize beta system configuration**
4.1	Finalize system configuration
4.2	Prepare documentation for final system
4.3	Complete project closure documentation

Project: Thermo-Gen

🟩 Complete
🟦 Ongoing

5.1 Figure 1: Gantt Chart of the team's goals for the remainder of the project

Each team member is designated a specific task that will bring forth the final result. Olivia is responsible for completing the GUI and making it both user-friendly and intuitive for the system it represents; this also includes it working in conjunction with notifications. Kameron is responsible for working with the code for temperature reading and writing; the code here will act as the backend for the temperature input in the GUI. The user will enter their desired temperature and the code written will read what the actual temperature is from the digital gauges and adjust it according to the user input. Gareth is responsible for working with the code for pressure reading and writing. This code will function with the same structure as the code for temperature reading and writing. Though we each have different tasks, we all are looking at each part of the code so that all members understand what each part does and can give help to another team-member when needed. To help expedite this process and make sure our work is both accurate and effective, we are going in-person to HeetShield Lab 1-2 times a week (more if necessary). We will also test remotely between the lab visits with our own simulated test cases to make sure we are bringing the best version of our system with us each time to our in-person lab visits.

6. Conclusion

This project delivers an automated control system that regulates temperature and pressure for HeetShield’s testing, combining thermocouple and pressure inputs to reach a desired environment. Using PySide6 for the GUI, QtAsyncio for asynchronous operation, and Windows Toast notifications for real time feedback, the system integrates modern tools into a cohesive, closed loop solution. By breaking the design into clear modules the architecture ensures flexibility, safety, and reliability. Ultimately, this work enhances HeetShield’s ability to conduct precise, consistent, and safe testing, contributing to their mission of advancing materials critical to aerospace, firefighting, and more.