**Project Documentation: Test Automation of HMI and PLC**

**Project Overview**

This project focuses on the **test automation of Human-Machine Interface (HMI) and Programmable Logic Controller (PLC)** systems. The primary goal is to develop a robust Python-based test automation framework to streamline and validate feature updates in a SCADA-based application used to program HMIs and PLCs.

**Objectives**

1. Automate the testing process for HMI and PLC systems.
2. Ensure consistent validation of features added with every monthly release.
3. Minimize manual intervention to enhance testing efficiency and reliability.

**Progress Overview**

As of now, the project is in the research and prototyping phase. Below is a summary of completed tasks:

**Research**

* **Understanding HMI and PLC Systems**: Gained a comprehensive understanding of HMI and PLC systems and their role in industrial automation.
* **SCADA Application Analysis**: Studied the SCADA-based application to identify critical areas for automation testing.
* **Test Automation Strategies**: Researched methodologies and best practices for automating testing in similar environments.

**Python Scripts Developed**

Two Python scripts have been developed as part of the initial prototyping phase:

* Script\_1: A basic script to simulate HMI interactions and GUI testing.
* Script\_2: A more advanced script focusing on HMI-PLC communication tests using the Modbus protocol.

**Scripts Overview**

**1. Script\_1:**

**Purpose:**

Simulates basic user interactions with an HMI interface using the tkinter library.

Features:

* Input Handling: Allows user input through an entry field.
* Dialog Interaction: Displays confirmation dialogs upon user input.
* Automated Testing: Validates HMI behaviour with pre-programmed test cases.

Key Components:

* Entry Widget: Captures user input.
* Button Widget: Triggers events such as dialog display.
* Test Functionality: The test\_hmi function simulates user actions and verifies the expected outcomes.

Example Test Case:

* Prepares input ("Hello") in the entry field.
* Simulates a button click.
* Checks if the dialog box behaves as expected, confirming successful input processing.

Execution:

Run the script directly to simulate the HMI and execute its built-in test.

**CODE:**

import tkinter as tk

from tkinter import messagebox

import time

def on\_button\_click():

    user\_input = entry.get()

    if user\_input:

        messagebox.showinfo("Input Received", f"You entered: {user\_input}")

def test\_hmi():

    entry.insert(0, "Hello")

    on\_button\_click()

    time.sleep(1)

    if entry.get() == "Hello":

        print("Test Passed: Dialog box appeared and processed input correctly.")

    else:

        print("Test Failed: Dialog box did not appear as expected.")

root = tk.Tk()

root.title("HMI Simulation")

entry = tk.Entry(root)

entry.pack(pady=10)

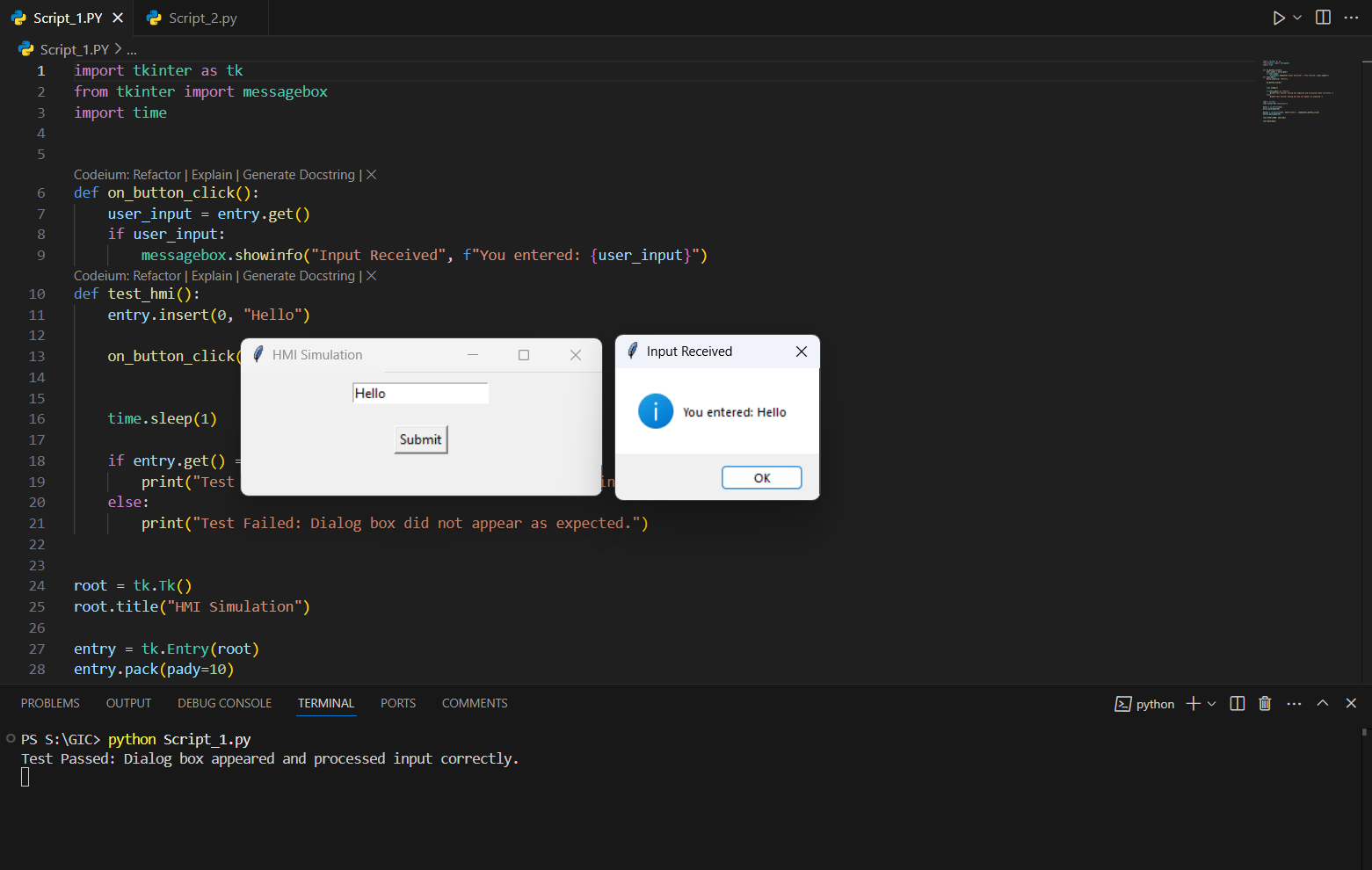
button = tk.Button(root, text="Submit", command=on\_button\_click)

button.pack(pady=10)

root.after(1000, test\_hmi)

root.mainloop()

**OUTPUT:**



**2. Script\_2:**

Purpose:

Conducts detailed functional testing of HMI components and ensures proper interaction with PLCs via Modbus communication. This script uses PyQt5 for building a GUI interface.

Features:

* Modbus Communication: Sends commands to and reads responses from a PLC.
* GUI Testing:
  + Button Tests: Verifies the functionality of simulated button presses.
  + Zoom Tests: Tests zoom-in and zoom-out behaviours.
  + Low Power Indicator Tests: Simulates low-power scenarios and validates system alerts.
* Log Feedback: Real-time logs for test outcomes and debugging.

Key Components:

* PyQt5 GUI: Provides a user-friendly interface for initiating tests.
* Modbus Integration: Implements Modbus client functionalities to send and verify commands.
* Error Handling: Captures and logs Modbus-related errors for debugging.

Example Test Workflow:

1. Button Test: Sends Modbus commands to activate and verify specific buttons (e.g., Button 1).
2. Zoom Test: Sequentially simulates zoom-in and zoom-out actions, verifying expected behavior.
3. Low Power Indicator Test: Simulates a low-power condition using Modbus commands and confirms system responses.

Execution:

Ensure the PLC is connected and configured before running the script. Execute the script to open the testing GUI.

**CODE:**

import sys

import time

from PyQt5.QtWidgets import QApplication, QWidget, QPushButton, QVBoxLayout, QTextEdit

from pymodbus.client import ModbusSerialClient as ModbusClient

from PyQt5.QtCore import Qt

class HMITestApp(QWidget):

    def \_\_init\_\_(self):

        super().\_\_init\_\_()

        # Setup Modbus client (adjust the serial port, baudrate, etc. as needed)

        self.client = ModbusClient(port='/dev/ttyUSB0', baudrate=9600, timeout=1, stopbits=1, bytesize=8, parity='N')

        self.client.connect()

        # GUI Setup

        self.setWindowTitle("HMI Testing Application")

        self.layout = QVBoxLayout()

        # Button 1 test

        self.button1 = QPushButton("Test Button 1", self)

        self.button1.clicked.connect(self.test\_button1)

        self.layout.addWidget(self.button1)

        # Button 2 test

        self.button2 = QPushButton("Test Button 2", self)

        self.button2.clicked.connect(self.test\_button2)

        self.layout.addWidget(self.button2)

        # Zoom In/Out test

        self.zoom\_in\_out\_button = QPushButton("Test Zoom In/Out", self)

        self.zoom\_in\_out\_button.clicked.connect(self.test\_zoom\_in\_out)

        self.layout.addWidget(self.zoom\_in\_out\_button)

        # Low Power Indicator test

        self.low\_power\_button = QPushButton("Test Low Power Indicator", self)

        self.low\_power\_button.clicked.connect(self.test\_low\_power\_indicator)

        self.layout.addWidget(self.low\_power\_button)

        # Log area for feedback

        self.log\_area = QTextEdit(self)

        self.log\_area.setReadOnly(True)

        self.layout.addWidget(self.log\_area)

        # Set layout and show window

        self.setLayout(self.layout)

    def log(self, message):

        """Log message to the text area."""

        self.log\_area.append(message)

    def test\_button1(self):

        self.log("Testing Button 1...")

        success = self.send\_modbus\_command(coil\_address=1)   # Send command to HMI via Modbus (replace with actual address and coil)

        if success:

            self.log("Button 1 tested successfully.")

        else:

            self.log("Button 1 test failed.")

    def test\_button2(self):

        self.log("Testing Button 2...")

        success = self.send\_modbus\_command(coil\_address=2)

        if success:

            self.log("Button 2 tested successfully.")

        else:

            self.log("Button 2 test failed.")

    def test\_zoom\_in\_out(self):

        self.log("Testing Zoom In/Out...")

        zoom\_in\_success = self.send\_modbus\_command(coil\_address=3)  # Send Zoom In command

        time.sleep(1)                                               # Simulate delay

        zoom\_out\_success = self.send\_modbus\_command(coil\_address=4) # Send Zoom Out command

        if zoom\_in\_success and zoom\_out\_success:

            self.log("Zoom In/Out tested successfully.")

        else:

            self.log("Zoom In/Out test failed.")

    def test\_low\_power\_indicator(self):

        self.log("Testing Low Power Indicator...")

        success = self.send\_modbus\_command(coil\_address=5)           # Simulate low power condition

        if success:

            self.log("Low Power Indicator tested successfully.")

        else:

            self.log("Low Power Indicator test failed.")

    def send\_modbus\_command(self, coil\_address):

        """Send a command to the HMI via Modbus to activate a coil (e.g., simulate a button press)."""

        try:

            self.client.write\_coil(coil\_address, True)              # Send a Modbus write command to activate the coil

            time.sleep(1)                                           # Wait for action to complete

            result = self.client.read\_coils(coil\_address, 1)        # Read back the coil to verify it was set

            if result.bits[0]:

                return True

            else:

                return False

        except Exception as e:

            self.log(f"Modbus command failed: {e}")

            return False

    def closeEvent(self, event):

        """Clean up when closing the app."""

        self.client.close()

        event.accept()

if \_\_name\_\_ == "\_\_main\_\_":

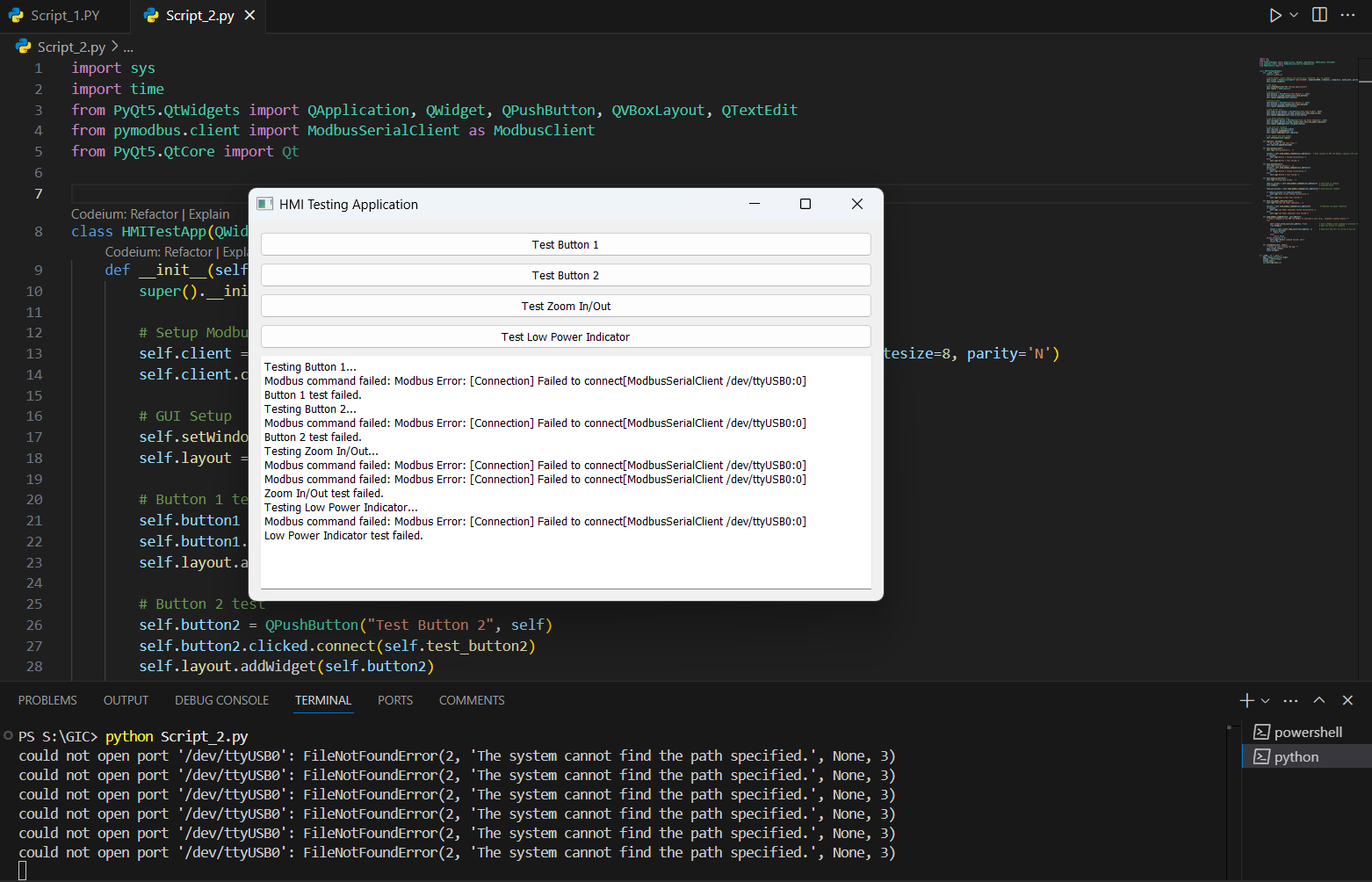
    app = QApplication(sys.argv)

    window = HMITestApp()

    window.show()

    sys.exit(app.exec\_())

**OUTPUT:**

****

**System Requirements**

* Python Version: 3.8 or higher
* Libraries:
  + tkinter
  + PyQt5
  + pymodbus
* Hardware: PLC connected and configured for Modbus communication.

**Benefits**

* Automated Testing: Eliminates manual effort in repetitive tasks.
* Real-Time Feedback: Logs provide immediate insights into test results.
* Improved Efficiency: Ensures quicker validation of HMI and PLC functionality.

**Challenges Encountered**

* **Integration with SCADA Application**: Establishing seamless interaction between Python scripts and the SCADA application required extensive troubleshooting.
* **Dynamic Feature Updates**: Adapting to frequent changes in features necessitates a flexible automation framework.
* **Limited Resources**: Current scripts are prototypes and need additional development for comprehensive testing.

**Next Steps**

1. Enhance the developed Python scripts for scalability and robustness.
2. Begin integration of the automation framework with the SCADA application.
3. Design test scenarios to cover broader feature sets of HMI and PLC systems.
4. Explore advanced libraries and tools to optimize test automation.

**Tools and Technologies Used**

* **Programming Language**: Python
* **Libraries**:
  + Pywinauto (for GUI automation)
  + Pytest (for test case management)
  + Pandas (for data handling and reporting)
* **Environment**: Windows (for SCADA application compatibility)

**Conclusion**

This project has laid a strong foundation for test automation of HMI and PLC systems through initial research and prototype development. The next phases will focus on refining these prototypes and building a comprehensive framework to address the dynamic requirements of the SCADA-based application.