Backend Development Specification

GRP Pipe Ring Stiffness Test Machine

Version 2.0 - With Direct Hardware I/O Reading

> ⟳ **KEY CHANGE IN v2.0:**
> Simple ON/OFF signals (Servo_Ready, Servo_Error, etc.) are now read DIRECTLY from hardware I/O.
> **This approach requires NO PLC programming for status signals!**

1. System Architecture

1.1 Data Reading Strategy

Data Type	Source	Method
Digital Status (ON/OFF)	Direct Hardware (PE)	read_area(Areas.PE, ...)
Analog Values (Raw)	Direct Hardware (PE)	read_area(Areas.PE, ...)
Calculated Values	Data Blocks (DB)	db_read(...)
Commands	Data Blocks (DB)	db_write(...)

1.2 Connection Parameters

Parameter	Value
PLC IP Address	192.168.0.100
Rack	0
Slot	1
Port	102 (S7 Protocol)

1.3 Technology Stack

Component	Technology	Version
Web Framework	FastAPI	0.109+
WebSocket	python-socketio	5.x
PLC Communication	python-snap7	1.3+
Database	SQLite + SQLAlchemy	2.x
PDF Generation	ReportLab	4.x
Excel Export	openpyxl	3.x

2. Snap7 Memory Areas

2.1 Area Codes Reference

Area	Hex Code	Snap7 Constant	Description
PE	0x81	Areas.PE	Process Inputs (I0.0, IW64, ...)
PA	0x82	Areas.PA	Process Outputs (Q0.0, ...)
MK	0x83	Areas.MK	Markers (M0.0, ...)
DB	0x84	Areas.DB	Data Blocks (DB1, DB2, ...)

2.2 read_area() Function Syntax

```
data = plc.read_area(area, db_number, start, size)

# Parameters:
#   area      = snap7.client.Areas.PE (for inputs)
#   db_number = 0 (not used for PE/PA, always 0)
#   start     = byte offset (0 for I0.x, 64 for IW64)
#   size      = number of bytes to read
```

3. Complete PLC Connector Class

```python
# plc_connector.py
import snap7
from snap7.util import get_bool, set_bool, get_real, set_real, get_int
import threading


class PLCConnector:
    def __init__(self, ip='192.168.0.100', rack=0, slot=1):
        self.ip = ip
        self.rack = rack
        self.slot = slot
        self.client = snap7.client.Client()
        self.lock = threading.Lock()
        self.connected = False

    def connect(self):
        try:
            self.client.connect(self.ip, self.rack, self.slot)
            self.connected = self.client.get_connected()
            return self.connected
        except Exception as e:
            print(f'Connection error: {e}')
            return False


    # ======================================================
    # DIRECT HARDWARE I/O - No PLC Programming Needed!
    # ======================================================

    def read_input_bit(self, byte_offset: int, bit: int) -> bool:
        """Read digital input directly from hardware (I0.0, I0.1, etc.)"""
        with self.lock:
            data = self.client.read_area(snap7.client.Areas.PE, 0, byte_offset, 1)
            return get_bool(data, 0, bit)

    def read_input_byte(self, byte_offset: int) -> int:
        """Read full input byte (IB0, IB1, etc.)"""
        with self.lock:
            data = self.client.read_area(snap7.client.Areas.PE, 0, byte_offset, 1)
            return data[0]

    def read_analog_input(self, address: int) -> int:
        """Read analog input word (IW64, IW66, etc.) - returns 0-27648"""
        with self.lock:
            data = self.client.read_area(snap7.client.Areas.PE, 0, address, 2)
            return get_int(data, 0)
```

3.1 PLC Connector (continued) - DB Operations

```python
# ============================================================
# DATA BLOCK OPERATIONS - For Calculated Values & Commands
# ============================================================

def read_db_real(self, db_number: int, offset: int) -> float:
    """Read Real (float) value from Data Block"""
    with self.lock:
        data = self.client.db_read(db_number, offset, 4)
        return get_real(data, 0)

def write_db_real(self, db_number: int, offset: int, value: float):
    """Write Real (float) value to Data Block"""
    with self.lock:
        data = bytearray(4)
        set_real(data, 0, value)
        self.client.db_write(db_number, offset, data)

def read_db_int(self, db_number: int, offset: int) -> int:
    """Read Int (16-bit) value from Data Block"""
    with self.lock:
        data = self.client.db_read(db_number, offset, 2)
        return get_int(data, 0)

def read_db_bool(self, db_number: int, byte_offset: int, bit: int) -> bool:
    """Read Bool from Data Block"""
    with self.lock:
        data = self.client.db_read(db_number, byte_offset, 1)
        return get_bool(data, 0, bit)

def write_db_bool(self, db_number: int, byte_offset: int, bit: int, value: bool):
    """Write Bool to Data Block (read-modify-write)"""
    with self.lock:
        data = self.client.db_read(db_number, byte_offset, 1)
        set_bool(data, 0, bit, value)
        self.client.db_write(db_number, byte_offset, data)
```

3.1 PLC Connector (continued) - DB Operations

4. Data Service - get_live_data()

Main function to get all live data for the dashboard:

```python
# data_service.py

class DataService:
    def __init__(self, plc: PLCConnector):
        self.plc = plc

    def get_live_data(self) -> dict:
        """Get all live data for dashboard - WebSocket broadcast"""
        return {

            # ================================================
            # DIRECT FROM HARDWARE - No PLC Code Needed!
            # ================================================
            'servo_ready': self.plc.read_input_bit(0, 0),        # I0.0
            'servo_error': self.plc.read_input_bit(0, 1),        # I0.1
            'at_home': self.plc.read_input_bit(0, 2),            # I0.2
            'upper_limit': self.plc.read_input_bit(0, 3),        # I0.3
            'lower_limit': self.plc.read_input_bit(0, 4),        # I0.4
            'e_stop': self.plc.read_input_bit(0, 6),             # I0.6
            'start_button': self.plc.read_input_bit(0, 7),       # I0.7

            # Load Cell - Direct Analog (scale in Python)
            'load_cell_raw': self.plc.read_analog_input(64),     # IW64

            # ================================================
            # FROM DATA BLOCKS - Calculated/Processed Values
            # ================================================
            'actual_force': self.plc.read_db_real(2, 0),         # DB2.DBD0
            'actual_deflection': self.plc.read_db_real(2, 4),    # DB2.DBD4
            'target_deflection': self.plc.read_db_real(2, 8),    # DB2.DBD8
            'ring_stiffness': self.plc.read_db_real(2, 12),      # DB2.DBD12
            'force_at_target': self.plc.read_db_real(2, 16),     # DB2.DBD16
            'sn_class': self.plc.read_db_int(2, 20),             # DB2.DBW20
            'test_status': self.plc.read_db_int(2, 22),          # DB2.DBW22
            'test_passed': self.plc.read_db_bool(2, 24, 0),      # DB2.DBX24.0
        }
```

4.1 Load Cell Scaling Function

```python
def scale_load_cell(raw_value: int) -> float:
    """
    Scale raw analog value (0-27648) to kN (0-200)
    S7-1200 analog: 0V=0, 10V=27648
    """
    MAX_RAW = 27648
    MAX_FORCE = 200.0  # kN

    if raw_value < 0:
        raw_value = 0

    return (raw_value / MAX_RAW) * MAX_FORCE
```

5. Command Service

```python
# command_service.py

class CommandService:
    def __init__(self, plc: PLCConnector):
        self.plc = plc

    # All commands write to DB3

    def set_enable(self, state: bool):
        """Enable/Disable servo - DB3.DBX0.0"""
        self.plc.write_db_bool(3, 0, 0, state)

    def jog_forward(self, state: bool):
        """Jog forward (down) - DB3.DBX0.1"""
        self.plc.write_db_bool(3, 0, 1, state)

    def jog_backward(self, state: bool):
        """Jog backward (up) - DB3.DBX0.2"""
        self.plc.write_db_bool(3, 0, 2, state)

    def start_test(self):
        """Start test (pulse) - DB3.DBX0.3"""
        self.plc.write_db_bool(3, 0, 3, True)

    def stop(self):
        """Emergency stop (pulse) - DB3.DBX0.4"""
        self.plc.write_db_bool(3, 0, 4, True)

    def home(self):
        """Go to home position - DB3.DBX0.5"""
        self.plc.write_db_bool(3, 0, 5, True)

    def alarm_reset(self):
        """Reset servo alarm (pulse) - DB3.DBX0.6"""
        self.plc.write_db_bool(3, 0, 6, True)

    def set_jog_velocity(self, velocity: float):
        """Set jog speed - DB3.DBD2"""
        self.plc.write_db_real(3, 2, velocity)
```

6. REST API Endpoints

6.1 Status Endpoints

Method	Endpoint	Description
GET	/api/status	Get all live data
GET	/api/status/connection	Check PLC connection
GET	/api/parameters	Get test parameters
POST	/api/parameters	Set test parameters

6.2 Command Endpoints

Method	Endpoint	Description
POST	/api/servo/enable	Enable servo
POST	/api/servo/disable	Disable servo
POST	/api/servo/reset	Reset servo alarm
POST	/api/command/start	Start test
POST	/api/command/stop	Emergency stop
POST	/api/command/home	Go to home
POST	/api/jog/speed	Set jog velocity

7. WebSocket Events

7.1 Server → Client

Event	Data
live_data	All live data object - sent every 100ms
test_complete	{test_id, ring_stiffness, sn_class, passed}
alarm	{code, message, timestamp}

7.2 Client → Server

Event	Data
jog_forward	{state: bool} - true on mousedown, false on mouseup
jog_backward	{state: bool} - true on mousedown, false on mouseup
subscribe	{} - start receiving live_data

Document Version: 2.0 | Updated: January 2026
Key Change: Direct Hardware I/O reading for status signals