

**Plataforma SCADA23**

Documento de Ingeniería de Detalle

FECHA: 7 de junio de 2025  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**CONTACTO**

Rafael Ausejo Prieto [rafael.ausejo@confianza23.es](mailto:rafael.ausejo@confianza23.es)

**ÍNDICE**

[1 Documento de Ingeniería de Detalle 3](#_Toc200197550)

[1.1 Introducción 3](#_Toc200197551)

[1.2 Requisitos Funcionales 3](#_Toc200197552)

[1.3 Requisitos No Funcionales 4](#_Toc200197553)

[2 Proceso de Sistema de Agua Potable 6](#_Toc200197554)

[2.1 Descripción Funcional del Proceso (FDS - Functional Design Specification) 6](#_Toc200197555)

[2.2 Hojas de Datos de Instrumentos 7](#_Toc200197556)

[2.3 Componentes 8](#_Toc200197557)

[3 Planificación del Desarrollo (Metodología SCRUM) 9](#_Toc200197558)

[3.1 Roles del Equipo 9](#_Toc200197559)

[3.2 Backlog del Producto 9](#_Toc200197560)

[3.3 Hitos de Desarrollo (Fase 1: Módulo Agua Potable) 10](#_Toc200197561)

[3.4 Tareas Detalladas y Nombres de Ficheros Python (.py) 11](#_Toc200197562)

[3.5 Consideraciones Adicionales 12](#_Toc200197563)

[4 Fichero README.md en Github 13](#_Toc200197564)

[5 Diseño Técnico de Módulos Software 17](#_Toc200197565)

[5.1 Estructura y Convenciones 17](#_Toc200197566)

[5.2 Biblioteca c23framework.py 18](#_Toc200197567)

[5.3 Módulo de Adquisición de Datos: data\_acquisition.py 22](#_Toc200197568)

[5.4 Módulo de Control de Actuadores: actuator\_control.py 26](#_Toc200197569)

[5.5 Módulo de Lógica de Control: control\_logic.py 29](#_Toc200197570)

[6 ANEXO: Sectores Críticos 32](#_Toc200197571)

# Documento de Ingeniería de Detalle

## Introducción

Este documento detalla los requisitos funcionales y no funcionales para el desarrollo de la plataforma SCADA23.

SCADA23 actuará como laboratorio de pruebas y simulación de PLCs, monitorizará y leerá tráfico de PLCs reales y funcionará como un sistema SCADA para la supervisión, control y adquisición de datos de sensores y actuadores reales.

La fase inicial se centrará en el soporte del protocolo Modbus/TCP y el sector de Agua Potable, permitiendo la extensión a otros protocolos y sectores mediante configuración.

El primer módulo implementado por defecto será el del sector de Agua Potable, simulando un sistema de tres tanques (Tank1, Tank2, Tank3) con sus respectivos sensores y actuadores.

**Debido a las limitaciones de tiempo, se realizará una versión limitada de la metodología expuesta para poder disponer de una primera versión funcional lo antes posible, en un fin de semana, realizando uso intensivo de la IA.**

El estado actual del proyecto se encuentra en la siguienteURL:

<https://github.com/rausejop/SCADA23>

Debido a que se trata de un proyecto Open Source, se permite y promueve la colaboración en dicho proyecto, si bien se ruega ponerse en contacto con la coordinación del proyecto para alinear las contribuciones a la situación en cada momento.

## Requisitos Funcionales

Se muestran a continuación los requerimientos funcionales

**RFUNC001: Simulación y Emulación de PLCs**

* **Descripción:** La plataforma SCADA23 debe ser capaz de simular y emular el comportamiento de PLCs. Esto incluye la inyección de tráfico Modbus/TCP simulando lecturas de sensores y respuestas a comandos de actuadores, utilizando la librería Scapy.
* **Comentarios:** Se espera que los scripts de simulación sean modulares, permitiendo la fácil adición de nuevos tipos de PLCs o comportamientos.

**RFUNC002: Monitorización de Tráfico PLC Real**

* **Descripción:** SCADA23 debe poder capturar y analizar el tráfico Modbus/TCP de PLCs reales del mercado para su monitorización.
* **Comentarios:** La captura de tráfico debe ser no intrusiva y la plataforma debe ser capaz de interpretar y visualizar los datos Modbus/TCP de manera significativa.

**RFUNC003: Sistema SCADA para Supervisión y Control**

* **Descripción:** La plataforma debe proporcionar una interfaz gráfica de usuario (HMI) para la supervisión, control y adquisición de datos de sensores y actuadores. Esta interfaz se construirá utilizando tcl/tk y representará un Diagrama de Tuberías e Instrumentación (P&ID).
* **Comentarios:** El HMI debe ser intuitivo y mostrar claramente el estado de los sensores y actuadores, permitiendo la interacción con los actuadores.

**RFUNC004: Soporte de Protocolos Industriales**

* **Descripción:** En esta primera fase, la plataforma debe implementar y soportar completamente el protocolo Modbus/TCP para la comunicación con PLCs simulados y reales.
* **Comentarios:** Aunque se mencionan otros protocolos (PROFINET, S7, OPC-UA), su implementación se pospondrá para fases posteriores. La arquitectura debe ser extensible para permitir su futura integración.

**RFUNC005: Configuración de Sectores Industriales**

* **Descripción:** La plataforma debe cargar por defecto el módulo para el sector "Agua Potable" y permitir la carga de otros sectores predefinidos a través de un archivo de configuración.
* **Comentarios:** El archivo de configuración debe ser de fácil edición y permitir la especificación de los parámetros relevantes para cada sector (e.g., tipos de sensores, actuadores, rangos, etc.).

## Requisitos No Funcionales

**RNFUNC001: Tecnología de Desarrollo**

* **Descripción:** La plataforma debe ser desarrollada íntegramente en Python 3.13.4 y ejecutarse en Windows 11. La comunicación con PLCs (simulados o reales) debe realizarse mediante scripts Python que utilicen la librería Scapy para la inyección y lectura de tráfico Modbus/TCP. La interfaz gráfica se desarrollará con tcl/tk.
* **Comentarios:** Se debe asegurar la compatibilidad con las versiones especificadas de Python y Windows. El uso de Scapy debe ser eficiente para evitar latencias significativas.

**RNFUNC002: Modularidad del Código**

* **Descripción:** El código fuente debe ser modular, bien estructurado y documentado, permitiendo el desarrollo en solitario o en equipo.
* **Comentarios:** Se deben seguir las mejores prácticas de codificación Python (PEP 8), utilizar comentarios claros y dividir el código en funciones y clases lógicas.

**RNFUNC003: Rendimiento**

* **Descripción:** La plataforma debe ser capaz de procesar y visualizar datos Modbus/TCP en tiempo real con una latencia mínima para garantizar un control y supervisión efectivos.
* **Comentarios:** Se deben realizar pruebas de rendimiento para asegurar que la actualización de los datos en el HMI y la respuesta a los comandos de actuadores sea fluida y sin retrasos perceptibles.

**RNFUNC004: Escalabilidad**

* **Descripción:** La arquitectura de la plataforma debe ser escalable para permitir la futura integración de nuevos protocolos industriales, tipos de sensores, actuadores y sectores.
* **Comentarios:** Se recomienda un diseño que desacople las capas de comunicación, lógica de negocio y presentación para facilitar futuras expansiones.

**RNFUNC005: Seguridad**

* **Descripción:** La plataforma debe estar diseñada con consideraciones de ciberseguridad industrial, minimizando las vulnerabilidades.
* **Comentarios:** Aunque Modbus/TCP no incluye seguridad intrínseca, se deben implementar las mejores prácticas de desarrollo seguro a nivel de aplicación para proteger la integridad y disponibilidad de los datos y el control. Se debe considerar la segregación de la red y el uso de firewalls.

# Proceso de Sistema de Agua Potable

## Descripción Funcional del Proceso (FDS - Functional Design Specification)

El sistema de agua potable simulará el llenado y vaciado de tres tanques, con la posibilidad de monitorizar sus parámetros (presión, temperatura, flujo y nivel) y controlar sus válvulas de entrada y salida.

**a. Rangos Operativos Normales**

* **Presión:** 2 a 6 bar
* **Temperatura:** 15 a 30 °C
* **Flujo:** 30 a 70 L/min
* **Nivel:** 20% a 80%

**b. Puntos de Ajuste para Alarmas y Control**

* **Alarma de Presión Alta:** > 8 bar
* **Alarma de Presión Baja:** < 1 bar
* **Alarma de Temperatura Alta:** > 90 °C
* **Alarma de Temperatura Baja:** < 5 °C
* **Alarma de Flujo Alto:** > 95 L/min
* **Alarma de Flujo Bajo:** < 10 L/min
* **Alarma de Nivel Alto:** > 95% (Detener llenado, abrir salida si es necesario)
* **Alarma de Nivel Bajo:** < 5% (Iniciar llenado, cerrar salida si es necesario)
* **Control de Válvulas:** El usuario podrá abrir/cerrar las válvulas de entrada y salida de cada tanque desde el HMI.

## Hojas de Datos de Instrumentos

A continuación, se detallan las especificaciones para los sensores y actuadores del sistema de Agua Potable. Estos datos servirán como base para la simulación y la representación en el P&ID.

**a. Sensores de Presión (PS/PT)**

* **Identificadores:** PS-101, PT-101 (Tanque 1); PS-102, PT-102 (Tanque 2); PS-103, PT-103 (Tanque 3)
* **Descripción:** Mide la presión del líquido dentro del tanque.
* **Rango de Medición:** 0 a 10 bar
* **Unidades:** bar
* **Punto de Ajuste (Setpoint):** Presión máxima para alarma (ej. 8 bar)
* **Precisión:** ±0.5% del fondo de escala
* **Conexión:** Modbus/TCP Holding Register

**b. Sensores de Temperatura (TS)**

* **Identificadores:** TS-101 (Tanque 1); TS-102 (Tanque 2); TS-103 (Tanque 3)
* **Descripción:** Mide la temperatura del líquido dentro del tanque.
* **Rango de Medición:** 0 a 100 °C
* **Unidades:** °C
* **Punto de Ajuste (Setpoint):** Temperatura mínima/máxima para alarma (ej. 5 °C, 90 °C)
* **Precisión:** ±1°C
* **Conexión:** Modbus/TCP Holding Register

**c. Sensores de Flujo (FS)**

* **Identificadores:** FS-101 (Tanque 1); FS-102 (Tanque 2); FS-103 (Tanque 3)
* **Descripción:** Mide el caudal de entrada o salida del tanque.
* **Rango de Medición:** 0 a 100 L/min
* **Unidades:** L/min
* **Punto de Ajuste (Setpoint):** Caudal mínimo/máximo para alarma (ej. 10 L/min, 95 L/min)
* **Precisión:** ±1% del fondo de escala
* **Conexión:** Modbus/TCP Holding Register

**d. Sensores de Nivel (LS)**

* **Identificadores:** LS-101 (Tanque 1); LS-102 (Tanque 2); LS-103 (Tanque 3)
* **Descripción:** Mide el nivel de llenado del tanque.
* **Rango de Medición:** 0% a 100%
* **Unidades:** Porcentaje (%)
* **Punto de Ajuste (Setpoint):** Nivel mínimo/máximo para alarma (ej. 5%, 95%)
* **Precisión:** ±1%
* **Conexión:** Modbus/TCP Holding Register

**e. Válvulas de Entrada (XV)**

* **Identificadores:** XV-101A (Tanque 1); XV-102A (Tanque 2); XV-103A (Tanque 3)
* **Descripción:** Actuador On/Off que controla el flujo de entrada al tanque.
* **Estado:** Abierta (True) / Cerrada (False)
* **Tiempo de Apertura/Cierre:** Instantáneo para simulación (en la realidad, podría ser milisegundos).
* **Conexión:** Modbus/TCP Coil

**f. Válvulas de Salida (XV)**

* **Identificadores:** XV-101B (Tanque 1); XV-102B (Tanque 2); XV-103B (Tanque 3)
* **Descripción:** Actuador On/Off que controla el flujo de salida del tanque.
* **Estado:** Abierta (True) / Cerrada (False)
* **Tiempo de Apertura/Cierre:** Instantáneo para simulación.
* **Conexión:** Modbus/TCP Coil

**g. Tanques de Agua Potable**

* **Identificadores:** T-101 (Tanque 1); T-102 (Tanque 2); T-103 (Tanque 3)
* **Descripción:** Depósitos de almacenamiento de agua potable.
* **Volumen/Capacidad:** Se definirá en el archivo de configuración por sector (ej. 1000 Litros).

## Componentes

**Sistema de Control (SCADA/PLC)**

El módulo de Agua Potable en SCADA23 se encargará de:

* **Adquisición de Datos:** Leerá los valores de los sensores (presión, temperatura, flujo, nivel) de los PLCs (simulados o reales) a través de Modbus/TCP (Holding Registers).
* **Visualización:** Los valores adquiridos se mostrarán en tiempo real en el HMI (P&ID) de forma gráfica. Se utilizarán indicadores visuales para el estado de las válvulas y el nivel de los tanques.
* **Control:** Enviará comandos a los actuadores (válvulas de entrada y salida) a los PLCs (simulados o reales) a través de Modbus/TCP (Coils) para cambiar su estado (abrir/cerrar).
* **Alertas:** Detectará y mostrará alarmas cuando los valores de los sensores superen o caigan por debajo de los puntos de ajuste predefinidos.
* **Registro de Datos:** La plataforma debería tener la capacidad básica de registrar los datos de los sensores a lo largo del tiempo, aunque esto podría ser una mejora futura.
* **Configuración:** Los rangos de medición, unidades y puntos de ajuste se cargarán desde el archivo de configuración del sector, permitiendo una fácil adaptación.

# Planificación del Desarrollo (Metodología SCRUM)

Se detalla la planificación para el desarrollo de la plataforma SCADA23, siguiendo una metodología ágil SCRUM. El equipo de desarrollo estará compuesto por tres personas. La fase inicial se centrará en el módulo de Agua Potable y la comunicación Modbus/TCP.

## Roles del Equipo

* **Product Owner:** Rafael Ausejo Prieto (contacto: rafael.ausejo@confianza23.es) - Responsable de la visión del producto y la priorización del Backlog.
* **Scrum Master:** [Nombre del Scrum Master] - Facilitador, elimina impedimentos y asegura la adherencia a la metodología SCRUM. (Rol asignado a este interlocutor para la generación de este documento)
* **Equipo de Desarrollo:**
  + Desarrollador 1: Rafael Ausejo Prieto
  + Desarrollador 2: J.S
  + Desarrollador 3:

## Backlog del Producto

El Backlog del Producto será gestionado y priorizado por el Product Owner. A continuación, se listan algunos ítems iniciales para la primera fase:

* Como usuario, quiero que la plataforma simule el comportamiento de PLCs para pruebas.
* Como usuario, quiero que la plataforma pueda capturar y analizar tráfico Modbus/TCP de PLCs reales.
* Como usuario, quiero una interfaz gráfica (HMI) para supervisar y controlar sensores y actuadores del sistema de agua potable.
* Como usuario, quiero que la plataforma soporte completamente el protocolo Modbus/TCP.
* Como administrador, quiero que la plataforma cargue por defecto el módulo de "Agua Potable" y sea configurable para otros sectores.
* Como desarrollador, quiero que el código sea modular y esté documentado.
* Como usuario, quiero que la HMI muestre claramente el estado de los sensores y actuadores del sistema de agua potable.
* Como usuario, quiero poder controlar las válvulas de entrada y salida de los tanques desde el HMI.
* Como desarrollador, quiero que la plataforma sea desarrollada en Python 3.13.4 y se ejecute en Windows 11, utilizando Scapy y tcl/tk.
* Como usuario, quiero que se visualicen los datos de los sensores (presión, temperatura, flujo, nivel) en tiempo real en el HMI.
* Como usuario, quiero que se detecten y muestren alarmas cuando los valores de los sensores superen o caigan por debajo de los puntos de ajuste.
* Como desarrollador, quiero que los rangos de medición y puntos de ajuste se carguen desde un archivo de configuración.
* Como usuario, quiero simular el llenado y vaciado de tres tanques de agua potable.

## Hitos de Desarrollo (Fase 1: Módulo Agua Potable)

Se proponen los siguientes hitos para la fase inicial del proyecto:

* **Hito 1 (Sprint 1-2): Configuración y Simulación Básica de PLC**
  + Definición del archivo de configuración inicial para Agua Potable.
  + Desarrollo del esqueleto del PLC simulado con Modbus/TCP y manejo de Holding Registers y Coils.
  + Simulación de lecturas básicas de sensores (presión, temperatura, flujo, nivel) y control de válvulas (On/Off).
  + Validación de la comunicación Modbus/TCP entre PLC simulado y un cliente de prueba.
* **Hito 2 (Sprint 3-4): HMI Básico y Visualización de Datos**
  + Implementación de la estructura básica del HMI con tcl/tk.
  + Conexión del HMI con el PLC simulado para la adquisición de datos de sensores.
  + Visualización en tiempo real de los valores de los sensores en el HMI.
  + Representación visual básica del estado de las válvulas y el nivel de los tanques.
* **Hito 3 (Sprint 5-6): Control y Alarmas**
  + Implementación de la funcionalidad de control de válvulas desde el HMI.
  + Desarrollo del módulo de detección y visualización de alarmas (presión, temperatura, flujo, nivel).
  + Integración de los puntos de ajuste y rangos operativos desde el archivo de configuración.
  + Mejora de la interactividad del HMI.
* **Hito 4 (Sprint 7): Refinamiento y Documentación**
  + Refinamiento del código y aseguramiento de la modularidad y documentación (PEP 8).
  + Pruebas de rendimiento y optimización para baja latencia.
  + Revisión de requisitos no funcionales (escalabilidad, seguridad).
  + Generación de la documentación técnica y de usuario inicial.

## Tareas Detalladas y Nombres de Ficheros Python (.py)

A continuación, se presenta un desglose de tareas por componentes, incluyendo los nombres de ficheros Python sugeridos.

**3.4.1. Módulo de Configuración y Utilidades**

* **Tarea:** Definir la estructura y cargar el archivo de configuración (config.json o config.ini).
  + **Ficheros:** config\_manager.py (Clase para cargar/manejar la configuración).
* **Tarea:** Implementar funciones utilitarias generales.
  + **Ficheros:** utils.py

**3.4.2. PLC Simulado (Simulador Modbus/TCP)**

Este componente será la base para la simulación de los tanques de agua.

* **Tarea:** Crear la clase base para un dispositivo Modbus/TCP (manejo de Holding Registers y Coils).
  + **Ficheros:** modbus\_device.py
* **Tarea:** Implementar la lógica específica de los tanques de agua (T-101, T-102, T-103) con sus sensores y actuadores.
  + **Ficheros:** tank\_simulator.py (Contendrá la lógica de llenado/vaciado, actualización de valores de sensores).
* **Tarea:** Crear scripts para simular la inyección de tráfico Modbus/TCP utilizando Scapy.
  + **Ficheros:** plc\_emulator.py (Orquestador de los tanques simulados y el servidor Modbus/TCP).
  + sensor\_data\_generator.py (Módulo para generar datos realistas de sensores).

**3.4.3. Consola SCADA (Cliente Modbus/TCP y HMI)**

Este componente será la interfaz de usuario para supervisión y control.

* **Tarea:** Desarrollar el cliente Modbus/TCP para leer Holding Registers y escribir Coils.
  + **Ficheros:** modbus\_client.py
* **Tarea:** Implementar la interfaz gráfica de usuario (HMI) utilizando tcl/tk.
  + **Ficheros:** scada\_hmi.py (Contendrá la lógica principal de la interfaz gráfica).
  + hmi\_elements.py (Clases para los elementos gráficos: tanques, válvulas, indicadores de sensores).
  + alarm\_manager.py (Lógica para detectar y mostrar alarmas).
* **Tarea:** Integrar la adquisición de datos y el control de actuadores en el HMI.
  + **Ficheros:** data\_acquisition.py (Módulo para gestionar la lectura periódica de datos del PLC).
  + control\_logic.py (Módulo para gestionar el envío de comandos al PLC).
* **Tarea:** Lógica para monitorizar y analizar tráfico de PLCs reales (captura no intrusiva). (Considerado para futuras iteraciones en detalle, pero la base estará en el cliente).
  + **Ficheros:** traffic\_monitor.py (Utilizará Scapy para la captura y análisis).

**3.4.4. Punto de Entrada Principal**

* **Tarea:** Crear los scripts principales para iniciar la simulación del PLC y la consola SCADA.
  + **Ficheros:** main\_plc\_simulator.py (Ejecuta el PLC simulado).
  + main\_scada\_console.py (Ejecuta la consola SCADA).

## Consideraciones Adicionales

* **Control de Versiones:** Se utilizará Git para el control de versiones
* **Pruebas:** Se implementarán pruebas unitarias y de integración para asegurar la calidad del código.
* **Reuniones SCRUM:**
  + **Daily Stand-ups:** Reuniones diarias de 15 minutos para sincronización.
  + **Sprint Planning:** Al inicio de cada Sprint para planificar las tareas.
  + **Sprint Review:** Al final de cada Sprint para revisar el incremento.
  + **Sprint Retrospective:** Al final de cada Sprint para identificar mejoras en el proceso.
* **Herramientas:** Se utilizarán entornos de desarrollo compatibles con Python (e.g., VS Code, PyCharm).
* **Documentación:** La documentación del código se realizará de forma continua, siguiendo las mejores prácticas de Python.

# Fichero README.md en Github

# <p align="center">SCADA23: Plataforma de Inteligencia y Seguridad Industrial 🛡️</p>

<p align="center">

<img src="https://github.com/rausejop/SCADA23/blob/main/docs/img/confianza23\_logo.png?raw=true" alt="CONFIANZA 23 Logo" width="200"/>

</p>

## 📄 Introducción

Este repositorio contiene la \*\*Plataforma SCADA23\*\*, un entorno de laboratorio y simulación diseñado para la supervisión, control y adquisición de datos en sistemas SCADA/PLC. Su propósito principal es simular y emular el comportamiento de PLCs, monitorizar el tráfico de PLCs reales y funcionar como un sistema SCADA completo.

La fase inicial de desarrollo se enfoca en el soporte del protocolo \*\*Modbus/TCP\*\* y el sector de \*\*Agua Potable\*\*, con una arquitectura extensible para futuras integraciones. El primer módulo implementado simula un sistema de tres tanques (Tank1, Tank2, Tank3) con sus respectivos sensores y actuadores.

Este proyecto es de \*\*código abierto\*\* y se fomenta la colaboración. Si deseas contribuir, por favor, contacta con la coordinación del proyecto para alinear tus aportaciones.

## ✨ Características Principales (Fase 1)

\* \*\*Simulación y Emulación de PLCs\*\*: Capacidad para simular y emular el comportamiento de PLCs, inyectando tráfico Modbus/TCP para lecturas de sensores y respuestas a actuadores utilizando la librería Scapy.

\* \*\*Monitorización de Tráfico PLC Real\*\*: Captura y análisis no intrusivo del tráfico Modbus/TCP de PLCs reales, con visualización significativa de los datos.

\* \*\*Sistema SCADA con HMI\*\*: Interfaz gráfica de usuario (HMI) desarrollada con `tcl/tk` para la supervisión, control y adquisición de datos, representando un Diagrama de Tuberías e Instrumentación (P&ID).

\* \*\*Soporte Modbus/TCP\*\*: Implementación completa del protocolo Modbus/TCP para la comunicación con PLCs simulados y reales.

\* \*\*Configuración de Sectores Industriales\*\*: Carga por defecto del módulo de "Agua Potable" y capacidad de configurar otros sectores predefinidos a través de un archivo de configuración.

\* \*\*Desarrollo en Python\*\*: Construido íntegramente en Python 3.13.4 y diseñado para ejecutarse en Windows 11.

## 🚀 Puesta en Marcha (Fase 1: Módulo Agua Potable)

### 📋 Prerrequisitos

\* Python 3.13.4

\* Sistema Operativo: Windows 11

\* Librerías Python: `Scapy`, `tcl/tk` (asegúrate de que `tcl/tk` esté disponible con tu instalación de Python).

### ⚙️ Instalación

1. Clona el repositorio:

```bash

git clone [https://github.com/rausejop/SCADA23.git](https://github.com/rausejop/SCADA23.git)

cd SCADA23

```

El estado actual del proyecto se encuentra en `https://github.com/rausejop/SCADA23`.

2. Crea un entorno virtual (opcional pero recomendado):

```bash

python -m venv venv

.\venv\Scripts\activate # En Windows

source venv/bin/activate # En Linux/macOS

```

3. Instala las dependencias:

```bash

pip install scapy

# Puede que necesites instalar tk/tcl por separado si no viene con tu distribución de Python

```

### 🏃 Ejecución

Para iniciar la simulación y la consola SCADA, ejecuta los siguientes scripts desde la raíz del proyecto:

1. Iniciar el PLC simulado:

```bash

python src/main\_plc\_simulator.py

```

2. Iniciar la consola SCADA (HMI):

```bash

python src/main\_scada\_console.py

```

## 📂 Estructura de Ficheros del Proyecto

La estructura del proyecto está diseñada para ser modular y extensible.

```

SCADA23/

├── src/

│ ├── config\_manager.py # 🔧 Clase para cargar y manejar la configuración del sistema (e.g., config.json).

│ ├── utils.py # 🛠️ Funciones utilitarias generales para el proyecto.

│ │

│ ├── modbus\_device.py # 🔌 Clase base para un dispositivo Modbus/TCP (manejo de Holding Registers y Coils).

│ ├── tank\_simulator.py # 💧 Lógica específica de los tanques de agua (T-101, T-102, T-103), incluyendo llenado/vaciado y actualización de sensores.

│ ├── plc\_emulator.py # 🤖 Orquestador de los tanques simulados y el servidor Modbus/TCP para emulación de PLC.

│ ├── sensor\_data\_generator.py # 📊 Módulo para generar datos realistas de sensores para la simulación.

│ │

│ ├── modbus\_client.py # 📡 Cliente Modbus/TCP para leer Holding Registers y escribir Coils en PLCs (simulados o reales).

│ ├── scada\_hmi.py # 🖥️ Lógica principal de la interfaz gráfica de usuario (HMI) construida con tcl/tk.

│ ├── hmi\_elements.py # 🎨 Clases para los elementos gráficos de la HMI: tanques, válvulas, indicadores de sensores.

│ ├── alarm\_manager.py # 🚨 Lógica para detectar y mostrar alarmas.

│ ├── data\_acquisition.py # 📈 Módulo para gestionar la lectura periódica de datos del PLC.

│ ├── control\_logic.py # 🕹️ Módulo para gestionar el envío de comandos al PLC.

│ ├── traffic\_monitor.py # 🕵️‍♂️ Módulo para la captura y análisis no intrusivo de tráfico Modbus/TCP de PLCs reales (utiliza Scapy).

│ │

│ ├── main\_plc\_simulator.py # ▶️ Script principal para iniciar el PLC simulado.

│ └── main\_scada\_console.py # ▶️ Script principal para iniciar la consola SCADA.

│

├── docs/ # 📚 Carpeta para la documentación del proyecto.

│ └── img/ # 🖼️ Imágenes utilizadas en la documentación (e.g., logos).

│

├── config/

│ └── water\_system.json # ⚙️ Archivo de configuración por defecto para el módulo de Agua Potable.

│

└── README.md # 📖 Este mismo archivo.

```

## 🤝 Colaboración

¡Agradecemos cualquier contribución! Si tienes ideas o quieres participar, por favor:

1. Ponte en contacto con el Product Owner: \*\*Rafael Ausejo Prieto\*\* (rafael.ausejo@confianza23.es).

2. Revisa el Backlog del Producto para las tareas priorizadas.

3. Sigue las mejores prácticas de codificación Python (PEP 8) y asegura una buena documentación del código.

Utilizamos \*\*Git\*\* para el control de versiones y seguimos una metodología \*\*SCRUM\*\* con Daily Stand-ups, Sprint Planning, Sprint Review y Sprint Retrospective.

## 📞 Contacto

\*\*Rafael Ausejo Prieto\*\*

\* \*\*Email\*\*: rafael.ausejo@confianza23.es

\* \*\*Organización\*\*: CONFIANZA 23 Inteligencia y Seguridad

## 📜 Licencia

Este proyecto es de código abierto.

# Diseño Técnico de Módulos Software

## Estructura y Convenciones

Se seguirán las siguientes convenciones para la documentación y el desarrollo de cada módulo:

* **Nomenclatura de Ficheros:** nombre\_del\_fichero.py en minúsculas y snake\_case. Es importante asegúrate de que las rutas (src/modules/...) reflejan la estructura real del proyecto en GitHub.
* **Propósito:** Breve descripción de la función principal del fichero dentro del sistema.
* **Dependencias:** Otros módulos internos o librerías externas de las que este fichero depende.
* **Convenciones de Codificación:** Se seguirá estrictamente el estilo de codificación PEP 8 y se incluirán comentarios y docstrings claros para todas las clases, funciones y métodos.
* **Archivos de Configuración:** Se proporciona un ejemplo detallado del archivo config/water\_system.json en el documento o en un apéndice, ya que el comportamiento de estos scripts depende en gran medida de él.

**Futuro**

* **Módulos Adicionales:** Dependiendo de la complejidad de SCADA23, se podrían necesitar módulos adicionales como:
  + hmi\_interface.py: Para interactuar con la interfaz de usuario (ej., usando Flask/Django para un web HMI, o PyQt/Tkinter para uno de escritorio).
  + data\_historian.py: Para almacenar datos históricos en una base de datos.
  + alarm\_manager.py: Un módulo más complejo para la gestión, notificación y acuse de recibo de alarmas.
* **Pruebas Unitarias:** Se considerarán pruebas unitarias robustas que verifiquen su funcionalidad de manera aislada.
* **Realismo Modbus:** Los ejemplos de Modbus asumen ciertos mapeos de registros. Se deberán eemplazarlos con los mapeos exactos de los PLCs o del simulador.
* **Seguridad:** En un sistema SCADA real, la seguridad (autenticación, autorización, cifrado de comunicaciones) es una preocupación primordial que debería abordar en el diseño de estos módulos, especialmente en el c23framework.py. En unfuturo, existirá un módulo 62443\_4\_4\_test.py que automatizará las validaciones de seguridad de la plataformas SCADA.

## Biblioteca c23framework.py

Esta biblioteca contendrá funciones y clases utilitarias comunes que serán utilizadas por varios componentes de la plataforma SCADA23, promoviendo la reutilización de código y la consistencia.

* **Ruta del Fichero:** src/c23framework.py
* **Descripción de la Función:** Este módulo actúa como una biblioteca central para la plataforma SCADA23, encapsulando funcionalidades recurrentes como la gestión de logs, la lectura de configuraciones, la interacción con Modbus/TCP y la estandarización de operaciones comunes. Su propósito es reducir la duplicación de código y centralizar las operaciones de bajo nivel.
* **Entradas (Inputs):**
  + Archivos de configuración (e.g., config/water\_system.json).
  + Parámetros pasados a las funciones (e.g., dirección Modbus, valor a escribir).
* **Salidas (Outputs):**
  + Mensajes de log.
  + Valores leídos de dispositivos Modbus.
  + Estado de operaciones (éxito/fracaso).

**Código Script Final en Python:**

Python

# src/c23framework.py

import logging

import json

from pymodbus.client import ModbusTcpClient

from pymodbus.payload import BinaryPayloadDecoder, BinaryPayloadBuilder

from pymodbus.constants import Endian

import os

import time

# --- Configuración de Logging ---

LOG\_FILE = 'scada23.log'

logging.basicConfig(

level=logging.INFO,

format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',

handlers=[

logging.FileHandler(LOG\_FILE),

logging.StreamHandler()

]

)

logger = logging.getLogger(\_\_name\_\_)

class C23Framework:

"""

Clase que encapsula funcionalidades comunes para la plataforma SCADA23.

"""

def \_\_init\_\_(self, config\_path='config/water\_system.json'):

self.config = self.\_load\_config(config\_path)

self.modbus\_clients = {} # Cache de clientes Modbus

def \_load\_config(self, config\_path):

"""Carga el archivo de configuración JSON."""

try:

# Asegura que la ruta es absoluta o relativa a la ubicación del script

base\_dir = os.path.dirname(os.path.abspath(\_\_file\_\_))

abs\_config\_path = os.path.join(base\_dir, '..', config\_path)

with open(abs\_config\_path, 'r') as f:

config\_data = json.load(f)

logger.info(f"Configuración cargada exitosamente desde: {abs\_config\_path}")

return config\_data

except FileNotFoundError:

logger.error(f"Error: Archivo de configuración no encontrado en {abs\_config\_path}")

return {}

except json.JSONDecodeError:

logger.error(f"Error: Archivo de configuración JSON inválido en {abs\_config\_path}")

return {}

def get\_modbus\_client(self, ip, port):

"""Obtiene o crea un cliente Modbus TCP."""

client\_key = f"{ip}:{port}"

if client\_key not in self.modbus\_clients:

client = ModbusTcpClient(ip, port)

try:

if client.connect():

logger.info(f"Conexión Modbus exitosa a {ip}:{port}")

self.modbus\_clients[client\_key] = client

else:

logger.error(f"Fallo al conectar Modbus a {ip}:{port}")

return None

except Exception as e:

logger.error(f"Excepción al conectar Modbus a {ip}:{port}: {e}")

return None

return self.modbus\_clients[client\_key]

def read\_holding\_registers(self, ip, port, address, count):

"""Lee un número de Holding Registers de un dispositivo Modbus."""

client = self.get\_modbus\_client(ip, port)

if client:

try:

result = client.read\_holding\_registers(address, count, unit=1) # unit=1 es común para PLCs

if not result.isError():

logger.debug(f"Lectura Modbus exitosa: {result.registers}")

return result.registers

else:

logger.error(f"Error en lectura Modbus de Holding Registers (IP:{ip}, Puerto:{port}, Dir:{address}, Cant:{count}): {result}")

return None

except Exception as e:

logger.error(f"Excepción durante lectura Modbus: {e}")

return None

return None

def write\_coil(self, ip, port, address, value):

"""Escribe un valor booleano en un Coil de un dispositivo Modbus."""

client = self.get\_modbus\_client(ip, port)

if client:

try:

result = client.write\_coil(address, value, unit=1)

if not result.isError():

logger.info(f"Escritura Modbus exitosa en Coil (IP:{ip}, Puerto:{port}, Dir:{address}, Valor:{value})")

return True

else:

logger.error(f"Error en escritura Modbus en Coil (IP:{ip}, Puerto:{port}, Dir:{address}, Valor:{value}): {result}")

return False

except Exception as e:

logger.error(f"Excepción durante escritura Modbus en Coil: {e}")

return False

return False

def decode\_16bit\_float(self, registers):

"""Decodifica un valor de punto flotante de 16 bits de registros Modbus."""

if not registers or len(registers) < 1:

return None

decoder = BinaryPayloadDecoder.fromRegisters(

registers, byteorder=Endian.Big, wordorder=Endian.Big

)

# Asumiendo que 16-bit float es un tipo específico.

# PyModbus por defecto soporta 32 y 64 bits. Para 16 bits, a menudo se usa un entero y se divide.

# Aquí asumimos que el registro es un entero que debe ser escalado.

# Adaptar según la implementación específica del PLC para 16-bit floats.

return decoder.decode\_16bit\_int() / 100.0 # Ejemplo: Si es un int que representa 2 decimales.

def decode\_32bit\_float(self, registers):

"""Decodifica un valor de punto flotante de 32 bits (IEEE 754) de registros Modbus."""

if not registers or len(registers) < 2: # Floats de 32 bits requieren 2 registros de 16 bits

return None

decoder = BinaryPayloadDecoder.fromRegisters(

registers, byteorder=Endian.Big, wordorder=Endian.Big

)

return decoder.decode\_32bit\_float()

def encode\_32bit\_float(self, value):

"""Codifica un valor de punto flotante de 32 bits a registros Modbus."""

builder = BinaryPayloadBuilder(byteorder=Endian.Big, wordorder=Endian.Big)

builder.add\_32bit\_float(value)

return builder.to\_registers()

def generate\_alarm(self, alarm\_type, message, severity="HIGH"):

"""Genera una alerta o alarma para el sistema."""

logger.warning(f"!!! ALARMA ({severity}) - {alarm\_type}: {message} !!!")

# Aquí se podría añadir lógica para enviar la alarma a un HMI, base de datos, etc.

# Por ejemplo, una cola de mensajes o una API de HMI.

pass

# Instancia global del framework para ser importada

c23\_framework = C23Framework()

**Consideraciones de Diseño:**

* + **Modularidad:** Centraliza la lógica de interacción con Modbus y la gestión de configuración/logging, facilitando que otros módulos se enfoquen en su lógica de negocio.
  + **Configuración:** La configuración se carga desde un archivo water\_system.json, permitiendo una fácil adaptación a diferentes entornos.
  + **Manejo de Errores:** Incluye try-except blocks para manejar fallos de conexión y comunicación Modbus.
  + **Optimización:** El cliente Modbus se cachea para evitar reconexiones costosas en cada operación.
  + **Escalabilidad:** Funciones como generate\_alarm están diseñadas para ser expandidas fácilmente para integrar con sistemas de notificación de alarmas más sofisticados.
  + **Conversión de Datos:** Se incluyen funciones de decodificación para tipos de datos comunes en Modbus (flotantes), asumiendo una representación típica. Se requerirá ajuste si los PLCs utilizan conversiones no estándar.

## Módulo de Adquisición de Datos: data\_acquisition.py

Este módulo será responsable de la lectura periódica de datos de los PLCs (simulados o reales) a través de Modbus/TCP y de su preprocesamiento.

**Ficha del Componente: data\_acquisition.py**

* **Ruta del Fichero:** src/modules/data\_acquisition.py (ejemplo de ruta)
* **Descripción de la Función:** El módulo data\_acquisition.py es el encargado de interactuar directamente con los dispositivos Modbus/TCP (PLCs). Su función principal es leer los valores de los sensores (presión, temperatura, flujo, nivel) y el estado de los actuadores (válvulas, bombas) desde los registros Modbus definidos. Además, realiza una validación básica y una conversión de tipos de datos, preparando la información para el módulo de procesamiento o el HMI.
* **Entradas (Inputs):**
  + Configuración del sistema de agua (water\_system.json) a través de c23framework, que incluye direcciones IP de PLCs, puertos y mapeo de registros Modbus.
  + Parámetros internos de la función (e.g., intervalo de lectura).
* **Salidas (Outputs):**
  + Un diccionario estructurado o un objeto con los datos de los sensores y el estado de los actuadores, listos para ser consumidos por otros módulos (e.g., data\_processing.py o el HMI).
  + Mensajes de log para el estado de la adquisición.

**Código Script Final en Python:**

Python

# src/modules/data\_acquisition.py

import time

from c23framework import c23\_framework, logger

class DataAcquisition:

"""

Gestiona la adquisición de datos de los dispositivos Modbus/TCP.

"""

def \_\_init\_\_(self):

self.modbus\_config = c23\_framework.config.get('modbus\_devices', {})

self.sensor\_mappings = c23\_framework.config.get('sensor\_mappings', {})

self.alarm\_thresholds = c23\_framework.config.get('alarm\_thresholds', {})

logger.info("Módulo DataAcquisition inicializado.")

def \_get\_sensor\_value(self, device\_ip, device\_port, register\_map):

"""

Lee un registro Modbus y decodifica el valor según el tipo de sensor.

"""

address = register\_map['address']

data\_type = register\_map['data\_type']

count = register\_map.get('count', 1) # Por defecto 1 registro

registers = c23\_framework.read\_holding\_registers(device\_ip, device\_port, address, count)

if registers is None:

logger.warning(f"No se pudieron leer los registros para el sensor en IP:{device\_ip}, Dir:{address}")

return None

value = None

if data\_type == 'float32':

value = c23\_framework.decode\_32bit\_float(registers)

elif data\_type == 'int16':

value = registers[0] # Asume un entero de 16 bits en el primer registro

# Añadir más tipos de datos si es necesario (e.g., 'float16', 'bool', 'string')

else:

logger.warning(f"Tipo de dato '{data\_type}' no soportado para decodificación del sensor.")

return value

def acquire\_all\_data(self):

"""

Adquiere datos de todos los sensores y actuadores configurados.

"""

acquired\_data = {

'sensors': {},

'actuators': {}

}

for device\_name, device\_info in self.modbus\_config.items():

ip = device\_info['ip']

port = device\_info['port']

# Leer sensores

for sensor\_name, sensor\_map in self.sensor\_mappings.get(device\_name, {}).items():

value = self.\_get\_sensor\_value(ip, port, sensor\_map)

if value is not None:

acquired\_data['sensors'][sensor\_name] = {

'value': value,

'unit': sensor\_map.get('unit', '')

}

# Comprobación de umbrales para alarmas

thresholds = self.alarm\_thresholds.get(sensor\_name)

if thresholds:

if value > thresholds.get('high\_alarm', float('inf')):

c23\_framework.generate\_alarm(

f"Sensor {sensor\_name}",

f"Valor ({value:.2f}) excede el umbral alto ({thresholds['high\_alarm']}).",

severity="CRITICAL"

)

elif value < thresholds.get('low\_alarm', float('-inf')):

c23\_framework.generate\_alarm(

f"Sensor {sensor\_name}",

f"Valor ({value:.2f}) por debajo del umbral bajo ({thresholds['low\_alarm']}).",

severity="CRITICAL"

)

else:

logger.error(f"Fallo al adquirir datos para sensor {sensor\_name} en dispositivo {device\_name}")

# Leer estado de actuadores (ejemplo: Coils o Holding Registers para estado)

# Esto es un ejemplo, se necesitarían mappings específicos para actuadores

if 'actuator\_mappings' in device\_info:

for actuator\_name, actuator\_map in device\_info['actuator\_mappings'].items():

# Suponemos que los actuadores tienen un tipo de registro específico (e.g., Coil para estado on/off)

if actuator\_map['type'] == 'coil':

# Esto necesitaría una función de lectura de Coils en c23framework

# Por ahora, solo simular, o implementar read\_coils en c23framework

logger.debug(f"Simulando lectura de actuador {actuator\_name} en {ip}:{port}")

# is\_on = c23\_framework.read\_coil(ip, port, actuator\_map['address']) # Necesita ser implementado

# acquired\_data['actuators'][actuator\_name] = is\_on

pass

return acquired\_data

# Ejemplo de uso (para testing o un script principal)

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

# Asegúrate de tener un archivo config/water\_system.json

# Ejemplo de water\_system.json:

"""

{

"modbus\_devices": {

"plc\_entrada": {

"ip": "127.0.0.1",

"port": 502

},

"plc\_salida": {

"ip": "127.0.0.1",

"port": 503

}

},

"sensor\_mappings": {

"plc\_entrada": {

"pressure\_in": {"address": 0, "data\_type": "float32", "unit": "bar"},

"temp\_in": {"address": 2, "data\_type": "float32", "unit": "°C"}

},

"plc\_salida": {

"flow\_out": {"address": 4, "data\_type": "float32", "unit": "m³/h"},

"level\_tank": {"address": 6, "data\_type": "float32", "unit": "m"}

}

},

"alarm\_thresholds": {

"pressure\_in": {"high\_alarm": 5.0, "low\_alarm": 1.0},

"temp\_in": {"high\_alarm": 30.0, "low\_alarm": 5.0},

"level\_tank": {"high\_alarm": 4.5, "low\_alarm": 1.5}

}

}

"""

data\_acq = DataAcquisition()

while True:

current\_data = data\_acq.acquire\_all\_data()

logger.info(f"Datos adquiridos: {current\_data}")

time.sleep(5) # Adquirir datos cada 5 segundos

**Consideraciones de Diseño:**

* + **Configurable:** Depende fuertemente del archivo water\_system.json para la configuración de dispositivos Modbus y mapeos de sensores, lo que lo hace flexible sin necesidad de modificar el código.
  + **Manejo de Errores:** Utiliza el framework para manejar errores de comunicación Modbus y registra advertencias para sensores no accesibles.
  + **Alertas:** Incluye una lógica básica de generación de alarmas basada en umbrales definidos en la configuración.
  + **Extensibilidad:** Fácilmente adaptable para añadir nuevos sensores o tipos de datos modificando únicamente la configuración y, si es necesario, las funciones de decodificación en c23framework.py.

## Módulo de Control de Actuadores: actuator\_control.py

Este módulo se encargará de enviar comandos a los PLCs para controlar los actuadores (ej., válvulas, bombas) basándose en las decisiones del sistema o entradas del usuario.

**Ficha del Componente: actuator\_control.py**

* **Ruta del Fichero:** src/modules/actuator\_control.py
* **Descripción de la Función:** El módulo actuator\_control.py proporciona la interfaz para manipular los actuadores del sistema de agua potable. Recibe comandos para cambiar el estado de válvulas o bombas y traduce estos comandos en operaciones Modbus/TCP de escritura (generalmente Coils o Holding Registers). Es crucial para la interacción bidireccional con el sistema físico.
* **Entradas (Inputs):**
  + Comandos específicos de control (ej., valve\_open, pump\_start).
  + Configuración del sistema de agua (water\_system.json) a través de c23framework, que incluye el mapeo de los actuadores a las direcciones Modbus.
* **Salidas (Outputs):**
  + Confirmación del éxito o fracaso de la operación de escritura Modbus.
  + Mensajes de log.

**Código Script Final en Python:**

Python

# src/modules/actuator\_control.py

from c23framework import c23\_framework, logger

class ActuatorControl:

"""

Gestiona el control de actuadores a través de Modbus/TCP.

"""

def \_\_init\_\_(self):

self.modbus\_config = c23\_framework.config.get('modbus\_devices', {})

self.actuator\_mappings = c23\_framework.config.get('actuator\_mappings', {})

logger.info("Módulo ActuatorControl inicializado.")

def control\_actuator(self, actuator\_name, state\_value):

"""

Envía un comando para cambiar el estado de un actuador.

:param actuator\_name: Nombre lógico del actuador (e.g., 'valve\_entry', 'pump\_main').

:param state\_value: Valor a escribir (e.g., True para abrir/encender, False para cerrar/apagar).

:return: True si la operación fue exitosa, False en caso contrario.

"""

actuator\_map = None

device\_ip = None

device\_port = None

# Buscar el mapeo del actuador en la configuración

for device\_name, device\_info in self.modbus\_config.items():

if actuator\_name in self.actuator\_mappings.get(device\_name, {}):

actuator\_map = self.actuator\_mappings[device\_name][actuator\_name]

device\_ip = device\_info['ip']

device\_port = device\_info['port']

break

if not actuator\_map:

logger.error(f"Actuador '{actuator\_name}' no encontrado en la configuración.")

return False

address = actuator\_map['address']

modbus\_type = actuator\_map['type'] # 'coil' o 'holding\_register'

if modbus\_type == 'coil':

success = c23\_framework.write\_coil(device\_ip, device\_port, address, state\_value)

if success:

logger.info(f"Comando '{'ON' if state\_value else 'OFF'}' enviado al actuador {actuator\_name} (Coil {address}).")

else:

logger.error(f"Fallo al enviar comando a actuador {actuator\_name} (Coil {address}).")

return success

elif modbus\_type == 'holding\_register':

# Si el actuador se controla vía Holding Register (ej., un setpoint, o un valor PWM)

# Se necesitaría una función write\_holding\_register en c23framework

if isinstance(state\_value, float):

registers = c23\_framework.encode\_32bit\_float(state\_value)

# c23\_framework.write\_holding\_registers(device\_ip, device\_port, address, registers) # Necesita ser implementado

logger.warning(f"Control de actuador '{actuator\_name}' vía Holding Register (Float) no implementado completamente.")

return False

else: # Asumir entero si no es float

# c23\_framework.write\_holding\_register(device\_ip, device\_port, address, state\_value) # Necesita ser implementado

logger.warning(f"Control de actuador '{actuator\_name}' vía Holding Register (Int) no implementado completamente.")

return False

else:

logger.error(f"Tipo de actuador '{modbus\_type}' no soportado para el actuador {actuator\_name}.")

return False

# Ejemplo de uso

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

# Asegúrate de que water\_system.json incluye la sección 'actuator\_mappings'

# Ejemplo de water\_system.json (sección de actuadores):

"""

{

"modbus\_devices": {

"plc\_entrada": {

"ip": "127.0.0.1",

"port": 502

}

},

"actuator\_mappings": {

"plc\_entrada": {

"valve\_entry": {"address": 0, "type": "coil"},

"pump\_main": {"address": 1, "type": "coil"}

}

}

}

"""

act\_ctrl = ActuatorControl()

logger.info("Abriendo válvula de entrada...")

success = act\_ctrl.control\_actuator('valve\_entry', True)

if success:

logger.info("Válvula de entrada abierta.")

else:

logger.error("No se pudo abrir la válvula de entrada.")

time.sleep(2)

logger.info("Encendiendo bomba principal...")

success = act\_ctrl.control\_actuator('pump\_main', True)

if success:

logger.info("Bomba principal encendida.")

else:

logger.error("No se pudo encender la bomba principal.")

time.sleep(2)

logger.info("Cerrando válvula de entrada...")

success = act\_ctrl.control\_actuator('valve\_entry', False)

if success:

logger.info("Válvula de entrada cerrada.")

else:

logger.error("No se pudo cerrar la válvula de entrada.")

**Consideraciones de Diseño:**

* + **Abstracción:** Proporciona una interfaz de alto nivel (control\_actuator) para que otros módulos no necesiten conocer los detalles de las direcciones Modbus o el tipo de registro.
  + **Configurable:** Al igual que data\_acquisition.py, su comportamiento es impulsado por la configuración JSON, permitiendo añadir o modificar actuadores sin cambiar el código.
  + **Seguridad y Fiabilidad:** En un sistema SCADA real, esta capa debería incluir validaciones de comandos y confirmaciones de estado de los actuadores, así como mecanismos de seguridad para evitar operaciones no autorizadas.
  + **Extensibilidad:** Puede expandirse fácilmente para soportar otros tipos de registros Modbus o lógicas de control más complejas.

## Módulo de Lógica de Control: control\_logic.py

Este módulo contendría la lógica de negocio para la automatización, tomando decisiones basadas en los datos adquiridos y enviando comandos a los actuadores.

**Ficha del Componente: control\_logic.py**

* **Ruta del Fichero:** src/modules/control\_logic.py
* **Descripción de la Función:** El módulo control\_logic.py implementa la lógica de control automatizado para el sistema de agua potable. Monitorea los datos de los sensores (presión, nivel, flujo) y, basándose en reglas predefinidas o algoritmos de control, toma decisiones para operar los actuadores (válvulas, bombas) a través del módulo actuator\_control. Por ejemplo, podría mantener el nivel de un tanque entre ciertos límites o ajustar el flujo en función de la demanda.
* **Entradas (Inputs):**
  + Datos procesados de los sensores (acquired\_data del módulo data\_acquisition).
  + Parámetros de reglas de control definidos en la configuración (water\_system.json).
* **Salidas (Outputs):**
  + Comandos enviados al módulo actuator\_control.
  + Mensajes de log sobre las acciones de control tomadas.

**Código Script Final en Python:**

Python

# src/modules/control\_logic.py

import time

from c23framework import c23\_framework, logger

from modules.data\_acquisition import DataAcquisition

from modules.actuator\_control import ActuatorControl

class ControlLogic:

"""

Implementa la lógica de control automatizado del sistema de agua.

"""

def \_\_init\_\_(self):

self.data\_acquisition = DataAcquisition()

self.actuator\_control = ActuatorControl()

self.control\_parameters = c23\_framework.config.get('control\_parameters', {})

logger.info("Módulo ControlLogic inicializado.")

def \_monitor\_and\_control\_level(self, current\_data):

"""

Lógica para mantener el nivel del tanque entre umbrales.

"""

level\_tank = current\_data['sensors'].get('level\_tank', {}).get('value')

if level\_tank is None:

logger.warning("Nivel del tanque no disponible para el control.")

return

min\_level = self.control\_parameters.get('tank\_level\_min', 2.0)

max\_level = self.control\_parameters.get('tank\_level\_max', 4.0)

valve\_entry\_open = self.actuator\_control.control\_actuator('valve\_entry', True) # Asumiendo que es una acción

valve\_entry\_close = self.actuator\_control.control\_actuator('valve\_entry', False)

if level\_tank < min\_level:

logger.info(f"Nivel del tanque ({level\_tank:.2f}m) bajo. Abriendo válvula de entrada.")

if not valve\_entry\_open:

c23\_framework.generate\_alarm("Control de Nivel", "Fallo al abrir válvula de entrada.")

elif level\_tank > max\_level:

logger.info(f"Nivel del tanque ({level\_tank:.2f}m) alto. Cerrando válvula de entrada.")

if not valve\_entry\_close:

c23\_framework.generate\_alarm("Control de Nivel", "Fallo al cerrar válvula de entrada.")

else:

logger.info(f"Nivel del tanque ({level\_tank:.2f}m) óptimo. Asegurando válvula de entrada cerrada.")

# Asegurar que la válvula esté cerrada si el nivel es óptimo

if not valve\_entry\_close: # Lógica para garantizar cierre si está dentro del rango

c23\_framework.generate\_alarm("Control de Nivel", "Fallo al asegurar cierre de válvula de entrada.")

def run\_control\_loop(self, interval=10):

"""

Ejecuta el bucle principal de control.

:param interval: Intervalo de tiempo en segundos entre ciclos de control.

"""

while True:

logger.info("Iniciando ciclo de control...")

current\_data = self.data\_acquisition.acquire\_all\_data()

if current\_data:

logger.debug(f"Datos para control: {current\_data}")

self.\_monitor\_and\_control\_level(current\_data)

# Añadir más lógicas de control aquí (e.g., control de presión, flujo)

else:

logger.warning("No se pudieron adquirir datos para el ciclo de control.")

time.sleep(interval)

# Ejemplo de uso

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

# Asegúrate de que water\_system.json incluye la sección 'control\_parameters'

# Ejemplo de water\_system.json (sección de control):

"""

{

"control\_parameters": {

"tank\_level\_min": 2.0,

"tank\_level\_max": 4.0

}

}

"""

control\_system = ControlLogic()

control\_system.run\_control\_loop(interval=5) # Ejecuta el bucle de control cada 5 segundos

**Consideraciones de Diseño:**

* + **Acoplamiento Flexible:** Se comunica con data\_acquisition y actuator\_control a través de sus interfaces, manteniendo un acoplamiento bajo.
  + **Modularidad de Reglas:** La lógica de control se puede segmentar en funciones separadas (\_monitor\_and\_control\_level, etc.), facilitando la adición de nuevas reglas de automatización.
  + **Configurable:** Los parámetros de control (umbrales, setpoints) se cargan desde la configuración, permitiendo ajustes sin modificar el código.
  + **Manejo de Errores:** Reporta fallos en la adquisición de datos o en el control de actuadores a través del sistema de alarmas del framework.

# ANEXO: Sectores Críticos

Tras la implementación del primer sector de Agua Potable, se implementará el resto de sectores soportados, que coinciden con la trasposición en España de la Directiva Europea NIS2 de Seguridad en Redes y Comunicaciones:

|  |  |  |
| --- | --- | --- |
| **Categoría** | **Código de Sector** | **Descripción del Sector** |
| **Energía** | SAC01 | Energía (General) |
|  | SAC01a | Electricidad |
|  | SAC01b | Sistemas urbanos de calefacción y refrigeración |
|  | SAC01c | Crudo |
|  | SAC01d | Gas |
|  | SAC01e | Hidrógeno |
| **Transporte** | SAC02 | Transporte (General) |
|  | SAC02a | Transporte aéreo |
|  | SAC02b | Transporte por ferrocarril |
|  | SAC02c | Transporte marítimo y fluvial |
|  | SAC02d | Transporte por carretera |
| **Banca** | SAC03 | Banca |
| **Finanzas** | SAC04 | Infraestructura de los mercados financieros |
| **Sanidad** | SAC05 | Sector sanitario |
| **Agua Potable** | SAC06 | Agua potable (Módulo inicial implementado por defecto) |
| **Aguas Residuales** | SAC07 | Aguas residuales |
| **Infraestructura digital** | SAC08 | Infraestructura digital |
| **Servicios TIC** | SAC09 | Gestión de servicios de TIC (de empresa a empresa) |
| **AGE** | SAC10 | Entidades de la Administración pública (excl. judicial, parlamentos y bancos centrales) |
| **Espacio** | SAC11 | Espacio |
| **Nuclear** | SAC12 | Industria Nuclear |
| **Otros Sectores** | OTR01 | Servicios postales y de mensajería |
|  | OTR02 | Gestión de residuos |
|  | OTR03 | Fabricación, producción y distribución de sustancias y mezclas químicas |
|  | OTR04 | Producción, transformación y distribución de alimentos |
| **Fabricación** | OTR05 | Fabricación (General) |
|  | OTR5a | Fabricación de productos sanitarios y productos sanitarios para diagnóstico in vitro |
|  | OTR5b | Fabricación de productos informáticos, electrónicos y ópticos |
|  | OTR5c | Fabricación de material eléctrico |
|  | OTR5d | Fabricación de maquinaria y equipo n.c.o.p. |
|  | OTR5e | Fabricación de vehículos de motor, remolques y semirremolques |
|  | OTR5f | Fabricación de otro material de transporte |
|  | OTR06 | Proveedores de servicios digitales |
|  | OTR07 | Investigación |
|  | OTR08 | Seguridad Privada |