

**Plataforma SCADA23**

Documento de Ingeniería de Detalle

FECHA: 11 de junio de 2025  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**CONTACTO**

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

**ÍNDICE**

[1 INTRODUCCIÓN Y RESUMEN EJECUTIVO 6](#_Toc200552980)

[1.1 Introducción 6](#_Toc200552981)

[1.2 Resumen Ejecutivo 7](#_Toc200552982)

[1.3 Hoja de ruta: 8](#_Toc200552983)

[1.4 Idoneidad para el Sector de Agua Potable 10](#_Toc200552984)

[1.5 Sobre la plataforma SCADA23 11](#_Toc200552985)

[2 REQUISITOS FUNCIONALES Y NO FUNCIONALES 16](#_Toc200552986)

[2.1 Requisitos Funcionales 16](#_Toc200552987)

[2.2 Requisitos No Funcionales 25](#_Toc200552988)

[2.3 Requisitos de Ciberseguridad 30](#_Toc200552989)

[3 PROCESO DEL SECTOR AGUA POTABLE 38](#_Toc200552990)

[3.1 Introducción al proceso ETAP 38](#_Toc200552991)

[3.2 Subprocesos y Puntos de Control en el Sector del Agua Potable 39](#_Toc200552992)

[3.3 Diagramas de flujo del proceso 43](#_Toc200552993)

[3.4 Estados operaciones del Sistema y transiciones 45](#_Toc200552994)

[3.5 Reglas de negocio generales y validaciones del sistema 46](#_Toc200552995)

[3.6 Hojas de Datos de Instrumentos 48](#_Toc200552996)

[3.7 Descripción Funcional del Proceso (FDS - Functional Design Specification) 52](#_Toc200552997)

[3.8 Lógica de control específica 54](#_Toc200552998)

[4 CASOS DE USO 56](#_Toc200552999)

[4.1 Diagrama de Casos de Uso (UML) 56](#_Toc200553000)

[4.2 Formato de Casos de Uso 56](#_Toc200553001)

[4.3 Categoría: Gestión de Bombas 57](#_Toc200553002)

[4.4 Categoría: Visualización y Monitorización 59](#_Toc200553003)

[4.5 Categoría: Gestión de Alarmas 59](#_Toc200553004)

[4.6 Categoría: Gestión de Usuarios y Seguridad 59](#_Toc200553005)

[4.7 Otros casos de uso 60](#_Toc200553006)

[5 DISEÑO DE LA ARQUITECTURA DE LA INFRAESTRUCTURA 66](#_Toc200553007)

[5.1 Arquitectura general 66](#_Toc200553008)

[5.2 Arquitectura de la Fase 1 (TRL4) 67](#_Toc200553009)

[5.3 Arquitectura de la Fase 2 (TRL7) 69](#_Toc200553010)

[5.4 Componentes generales 75](#_Toc200553011)

[5.5 Componentes principales y sus responsabilidades 77](#_Toc200553012)

[5.6 Interfaces y drivers de conexión 83](#_Toc200553013)

[5.7 Protocolos utilizados 84](#_Toc200553014)

[6 DISEÑO DE LA BASE DE DATOS 86](#_Toc200553015)

[6.1 Estrategia de Almacenamiento de Datos 86](#_Toc200553016)

[6.2 Diseño de la Base de Datos Relacional (PostgreSQL) 86](#_Toc200553017)

[6.3 Diseño de la Base de Datos de Series Temporales (InfluxDB) 90](#_Toc200553018)

[6.4 Volúmenes de Datos y Crecimiento Esperado 91](#_Toc200553019)

[6.5 Estrategias de Backup y Restauración 92](#_Toc200553020)

[7 DISEÑO DEL INTERFAZ DE USUARIO Y HMI 93](#_Toc200553021)

[7.1 Introducción 93](#_Toc200553022)

[7.2 Principios de Diseño UI/UX para SCADA23 93](#_Toc200553023)

[7.3 Elementos Clave de la Interfaz de Usuario 94](#_Toc200553024)

[7.4 Wireframes y Maquetas 96](#_Toc200553025)

[7.5 Consideraciones de Usabilidad y Accesibilidad 97](#_Toc200553026)

[7.6 HMI (Human Management Interface) 98](#_Toc200553027)

[8 INTEGRACIÓN CON OTRAS TECNOLOGÍAS 106](#_Toc200553028)

[8.1 OpenPLC 106](#_Toc200553029)

[8.2 ScadaBR 108](#_Toc200553030)

[8.3 Factory I/O 111](#_Toc200553031)

[9 DISEÑO DE LA INTERFAZ DE PROGRAMACIÓN DE APLICACIONES (API) 115](#_Toc200553032)

[9.1 Principios de Diseño de la API 115](#_Toc200553033)

[9.2 Autenticación y Autorización de la API 115](#_Toc200553034)

[9.3 Endpoints RESTful Clave (API v1) 116](#_Toc200553035)

[9.4 Comunicación en Tiempo Real (WebSockets) 119](#_Toc200553036)

[9.5 Estructura de Respuesta de Errores (JSON) 119](#_Toc200553037)

[9.6 Documentación de la API (OpenAPI/Swagger) 120](#_Toc200553038)

[10 DISEÑO DE LA ARQUITECTURA DE SOFTWARE 121](#_Toc200553039)

[10.1 Convenciones de Codificación 121](#_Toc200553040)

[10.2 Estructura y Convenciones 122](#_Toc200553041)

[10.3 Bibliotecas y Frameworks Clave 123](#_Toc200553042)

[10.4 Identificación de Módulos y Componentes Principales 124](#_Toc200553043)

[10.5 Clases y Estructuras de Datos Fundamentales 125](#_Toc200553044)

[10.6 Estrategia de Manejo de Errores y Excepciones 126](#_Toc200553045)

[10.7 Estrategia de Pruebas 127](#_Toc200553046)

[10.8 Gestión de Dependencias 128](#_Toc200553047)

[11 PLANIFICACIÓN DEL DESARROLLO 130](#_Toc200553048)

[11.1 Enfoque por Fases (TRL4 y TRL7) 130](#_Toc200553049)

[11.2 Roles del Equipo 131](#_Toc200553050)

[11.3 Backlog del Producto 131](#_Toc200553051)

[11.4 Planificación (Fase 1) 132](#_Toc200553052)

[11.5 Planificación (Fase 2) 135](#_Toc200553053)

[11.6 Tareas Detalladas y Nombres de Ficheros Python (.py) 139](#_Toc200553054)

[11.7 Consideraciones Adicionales 140](#_Toc200553055)

[12 METODOLOGÍA DE DESARROLLO DE SOFTWARE 141](#_Toc200553056)

[12.1 Metodología de Desarrollo 141](#_Toc200553057)

[12.2 Fases de Desarrollo y Entregables Clave 141](#_Toc200553058)

[12.3 Definición de Terminado (Definition of Done - DoD) 144](#_Toc200553059)

[12.4 Equipo de Desarrollo (Roles Clave) 144](#_Toc200553060)

[12.5 Gestión de Cambios 145](#_Toc200553061)

[13 ESPECIFICACIÓN DE LA ARQUITECTURA DETALLADA DE SOFTWARE 146](#_Toc200553062)

[13.1 Arquitectura Detallada de Software - Fase 1 (TRL4) 146](#_Toc200553063)

[13.2 Arquitectura Detallada de Software - Fase 2 (TRL7) 149](#_Toc200553064)

[14 ARQUITECTURA HARDWARE 154](#_Toc200553065)

[14.1 Arquitectura Hardware e Infraestructuras Asociadas - Fase 1 (TRL4) 154](#_Toc200553066)

[14.2 Arquitectura Hardware e Infraestructuras Asociadas - Fase 2 (TRL7) 156](#_Toc200553067)

[15 CIBERSEGURIDAD DEL SISTEMA 160](#_Toc200553068)

[16 PRUEBAS DEL SISTEMA 165](#_Toc200553069)

[17 IMPLEMENTACIÓN Y DESPLIEGUE 170](#_Toc200553070)

[17.1 Metodología de despliegue 170](#_Toc200553071)

[17.2 Despliegue en Windows 111 174](#_Toc200553072)

[18 CÓDIGO FUENTE DEL REPOSITORIO 178](#_Toc200553073)

[18.1 Fichero README.md 178](#_Toc200553074)

[18.2 Estructura del repositorio de código 183](#_Toc200553075)

[18.3 Biblioteca c23framework.py 184](#_Toc200553076)

[18.4 Módulo de Adquisición de Datos: data\_acquisition.py 188](#_Toc200553077)

[18.5 Módulo de Control de Actuadores: actuator\_control.py 192](#_Toc200553078)

[18.6 Módulo de Lógica de Control: control\_logic.py 195](#_Toc200553079)

[19 ANEXO: SECTORES CRÍTICOS 198](#_Toc200553080)

[20 ANEXO: PATRONES DE DISEÑO 200](#_Toc200553081)

[20.1 Patrones de Diseño de Software utilizados en SCADA23 200](#_Toc200553082)

[20.2 Patrones Creacionales 201](#_Toc200553083)

[20.3 Patrones Estructurales 202](#_Toc200553084)

[20.4 Patrones de Comportamiento 202](#_Toc200553085)

[21 ANEXO: TECHNOLOGY READINESS LEVELS 204](#_Toc200553086)

# INTRODUCCIÓN Y RESUMEN EJECUTIVO

## Introducción

**Sobre el documento**

El presente Documento de Ingeniería de Detalle DID) pretende definir las especificaciones técnicas para el diseño, desarrollo e implementación de la plataforma **SCADA23**, una solución de monitorización y control industrial (SCADA - Supervisory Control And Data Acquisition) diseñada específicamente para la gestión eficiente y segura de infraestructuras críticas, comenzando por sector del agua potable.

El documento servirá como una guía detallada para el diseño, desarrollo, implementación y mantenimiento de la Plataforma SCADA23, proporcionando una visión clara de sus capacidades, requisitos y la hoja de ruta para su evolución.

**Objetivo**

* El objetivo principal es proporcionar una base sólida de especificaciones técnicas para el equipo de desarrollo, hasta alcanzar un nivel de desarrollo TRL4 (Technology Rediness Level), equivalente a la “Validación a nivel de componentes en entorno de laboratorio”, con fines de investigación, como prueba de concepto tecnológico.
* La intención futura sería poder elevarlo hasta TRL7 “Demostración de sistema o prototipo en un entorno operativo real” como proyecto de Innovación, como prototipo demostrador de desarrollo tecnológico llegando a convertirse en producto o servicio comercializable, certificable con pruebas específicas. SCADA23 se concibe como una plataforma modular y flexible, capaz de evolucionar y adaptarse a las necesidades cambiantes de la industria. Su desarrollo se estructurará en dos fases bien diferenciadas, cada una con un Nivel de Madurez Tecnológica (TRL) específico y un conjunto de tecnologías y funcionalidades adaptadas a dicho nivel. Esta aproximación incremental garantiza la viabilidad del proyecto, la validación temprana de conceptos y la optimización de recursos.

**Contenido**

Este documento establece y abarca:

* la visión global del proyecto
* los requisitos funcionales y no funcionales
* el proceso de sistema de agua potable
* los casos de uso
* la arquitectura del sistema
* la selección de tecnologías,
* el modelado de datos y el diseño de la base de datos
* los patrones de diseño aplicados
* las interfaces de usuario y programación
* las consideraciones de ciberseguridad
* las metodologías de prueba e implementación.
* y el plan de hitos para su desarrollo.
* así como diversos anexos con información relacionada.

**Fases de desarrollo**

* La Fase 1, en un TRL4 (Tecnología Validada en Entorno Relevante), se centrará en la demostración de la funcionalidad básica y la interconexión de componentes clave. Se implementará una versión inicial del SCADA con un alcance limitado en cuanto a protocolos de comunicación y una interfaz de usuario simplificada.
* La Fase 2, alcanzando un TRL7 (Demostración de Prototipo en Entorno Operacional), implicará el desarrollo completo de la plataforma, incorporando arquitecturas avanzadas, una mayor variedad de protocolos y una interfaz de usuario profesional y completa, diseñada para operar en un entorno real.

## Resumen Ejecutivo

La Plataforma SCADA23 es una solución integral que busca optimizar la operación y el mantenimiento de infraestructuras críticas, garantizando la fiabilidad, la eficiencia y la seguridad de los procesos. A través de este documento, se establece una base sólida para el desarrollo, despliegue y operación del sistema, asegurando que cumpla con los estándares más exigentes de la industria.

**Puntos Clave del Sistema SCADA23:**

* **Arquitectura Robusta y Escalable:** El sistema se fundamenta en una arquitectura multicapa, distribuyendo las responsabilidades para mejorar el rendimiento, la disponibilidad y la escalabilidad. Esto incluye la capa de presentación (HMI/UI), la capa de aplicación/lógica de negocio, la capa de comunicación y la capa de datos.
* **Gestión Eficiente de Procesos:** Se ha priorizado la capacidad de control y monitorización en tiempo real, permitiendo la supervisión de elementos críticos como bombas y válvulas, así como la gestión de estados de proceso (e.g., 'Parada', 'Arranque', 'Funcionando', 'Fallida').
* **Ciberseguridad Integrada:** Desde el diseño inicial, se han incorporado medidas de ciberseguridad basadas en el estándar ISA IEC 62443-4-2 para proteger la integridad, confidencialidad y disponibilidad de los datos y operaciones del sistema. Esto incluye el control de acceso basado en roles (RBAC), mecanismos de cifrado, auditorías de seguridad y gestión de parches.
* **Aplicación de Patrones de Diseño:** Para garantizar la calidad del código, la mantenibilidad y la extensibilidad, se han seleccionado y aplicado patrones de diseño de software. Ejemplos incluyen el patrón **Command** para la ejecución de operaciones y el patrón **State** para la gestión del comportamiento de los objetos en función de su estado interno, simplificando la lógica condicional y facilitando la adición de nuevas funcionalidades.
* **Flexibilidad Tecnológica:** La selección de tecnologías se ha realizado buscando un equilibrio entre rendimiento, compatibilidad y madurez, asegurando que la plataforma sea adaptable a futuros requisitos y evoluciones tecnológicas.
* **Centrado en el Usuario:** La interfaz de usuario (HMI) se ha diseñado para ser intuitiva y permitir una visualización clara y completa del estado del sistema, así como un control eficiente de las operaciones.
* **Conformidad con Estándares:** El diseño y la implementación se alinean con las mejores prácticas de la industria y los estándares relevantes, garantizando la interoperabilidad y la calidad del sistema. Entre ellos se encuentran ISA 101 par el diseño de HMI e ISA IEC 62443 para controles de ciberseguridad.

## Hoja de ruta:

Se muestra a continuación de la hoja de ruta de la plataforma

**Fase 1: TRL4 – Demostración de Concepto y Viabilidad Básica**

Esta fase inicial se enfocará en establecer la base funcional de la plataforma, validando los principios de conectividad y control en un entorno de laboratorio o prototipo.

* **Alcance:** Demostración de funcionalidades SCADA básicas: adquisición de datos, visualización y control elemental.
* **Tecnologías Clave:**
  + **Cliente:** Se utilizarán unos pocos scripts en Python ejecutados en línea de comandos. Estos scripts simularán la interacción con el sistema, permitiendo enviar comandos y recibir datos de forma rudimentaria.
  + **SCADA (Servidor):** Un programa desarrollado en Python con interfaz gráfica utilizando Tkinter (Tcl/Tk). Esta interfaz proporcionará una visualización básica de los datos y controles, pero su propósito principal será la demostración de la conectividad y el procesamiento de la información.
  + **Protocolos de Comunicación:** Exclusivamente Modbus TCP. Esta elección se basa en su ubicuidad y facilidad de implementación, lo que permite una validación rápida de la capacidad de comunicación con dispositivos industriales.
* **Objetivos:**
  + Confirmar la viabilidad de la arquitectura propuesta para la adquisición y control de datos.
  + Validar la comunicación Modbus TCP entre el SCADA y los dispositivos simulados o reales.
  + Establecer las bases para el desarrollo futuro de funcionalidades más complejas.

**Fase 2: TRL7 – Plataforma completa para Entorno Operacional (OT)**

Esta fase representa el desarrollo completo y profesional de la Plataforma SCADA23, orientada a su despliegue en entornos productivos reales. Se incorporarán funcionalidades avanzadas, una arquitectura robusta y una interfaz de usuario de alto nivel.

* **Alcance:** Sistema SCADA completo con funcionalidades avanzadas de visualización, control, alarmas, tendencias, gestión de usuarios, ciberseguridad integrada y alta disponibilidad.
* **Tecnologías Clave:**
  + **Backend (API y Lógica de Negocio):** Django y FastAPI. Django proveerá la estructura de un framework web robusto, ideal para la gestión de usuarios, autenticación y la lógica de negocio compleja. FastAPI se utilizará para desarrollar APIs de alto rendimiento para la adquisición de datos en tiempo real y la comunicación eficiente con los clientes.
  + **API REST (para comunicación):** Django REST Framework (DRF). DRF facilitará la creación de APIs RESTful que permitan una comunicación eficiente y estandarizada entre el backend y las interfaces de usuario, así como con otros sistemas externos.
  + **Bases de Datos:**
    - **MySQL:** Para la gestión de datos estructurados, configuración del sistema, usuarios, permisos, históricos de eventos, alarmas y metadatos.
    - **InfluxDB:** Para el almacenamiento de datos de series temporales (mediciones de sensores, actuadores, etc.) debido a su alta eficiencia en la ingesta y consulta de datos en tiempo real y su capacidad para manejar grandes volúmenes de información.
  + **Protocolos de Comunicación:** Además de Modbus TCP, se añadirá soporte para PROFINET y OPC-UA. Esta expansión de protocolos permitirá la integración con una gama mucho más amplia de equipos industriales y sistemas de automatización, aumentando la interoperabilidad de la plataforma.
* **Objetivos:**
  + Desarrollar una plataforma SCADA completa, escalable y cibersegura.
  + Proveer una interfaz de usuario intuitiva y rica en funcionalidades para la operación y supervisión de procesos.
  + Garantizar la interoperabilidad con una amplia variedad de dispositivos y sistemas industriales.
  + Optimizar el rendimiento y la fiabilidad del sistema para entornos críticos.

## Idoneidad para el Sector de Agua Potable

La elección del sistema de aguas potables como caso de aplicación principal se debe a que representa un escenario ideal para la implementación de un sistema SCADA.

Este sector se caracteriza por la presencia de múltiples equipos PLC distribuidos, el uso extendido del protocolo de comunicaciones Modbus TCP, y una gran cantidad de sensores heredados que requieren una monitorización continua con un tráfico de datos relativamente ligero, incluyendo la integración de *smart meters* para la medición del flujo de agua. Esto permite demostrar la capacidad del sistema para gestionar infraestructuras complejas con requisitos de conectividad y procesamiento específicos.

La primera fase de SCADA23 ha sido diseñada y optimizada específicamente para abordar los desafíos y requisitos únicos del sector del agua potable, asegurando su compatibilidad y cumplimiento con los estándares operativos y regulaciones aplicables. Su idoneidad se fundamenta en las siguientes características:

* **Gestión Integral del Ciclo del Agua:** SCADA23 permite supervisar y controlar todas las fases del ciclo integral del agua, desde la captación en fuentes naturales, pasando por las Estaciones de Tratamiento de Agua Potable (ETAP), las redes de distribución, los depósitos, las Estaciones de Bombeo (EB), hasta las Estaciones Depuradoras de Aguas Residuales (EDAR).
* **Gestión Hídrica Específica:** Incorpora modelos de datos predefinidos para elementos hídricos comunes (depósitos, bombas, válvulas, caudalímetros, cloradores, etc.) y lógicas de control adaptadas a la operación de estaciones de tratamiento de agua potable (ETAP), estaciones de bombeo (EBAP) y redes de distribución.
* **Adquisición de Datos Multifuente:** La capacidad de integrar múltiples protocolos (Modbus TCP, PROFINET, OPC-UA en la Fase 2) asegura la compatibilidad con una vasta gama de equipos de campo, incluyendo PLCs, RTUs, sensores de caudal, presión, nivel, calidad del agua, y actuadores como válvulas y bombas, independientemente de su fabricante.
* **Visualización en Tiempo Real:** Las interfaces gráficas (Tkinter en Fase 1, web con Django/DRF en Fase 2) proporcionarán representaciones visuales claras y actualizadas de los procesos, permitiendo a los operadores monitorear el estado de las instalaciones, identificar anomalías y tomar decisiones rápidas.
* **Control Remoto y Automatización:** La plataforma facilitará el control remoto de equipos y la implementación de lógicas de control para automatizar operaciones críticas, como el ajuste de niveles en depósitos, la regulación de caudales o el arranque/parada de bombas, optimizando el consumo energético y la gestión de recursos hídricos.
* **Gestión de Alarmas y Eventos:** Un sistema robusto de alarmas y eventos permitirá la detección proactiva de situaciones anómalas, enviando notificaciones a los operadores y registrando todos los incidentes para un análisis posterior. Esto es crucial para la prevención de averías, fugas o interrupciones en el suministro.
* **Análisis Histórico y Tendencias:** El uso de bases de datos de series temporales como InfluxDB (en Fase 2) permitirá el almacenamiento eficiente de grandes volúmenes de datos históricos. Esto facilitará el análisis de tendencias, la identificación de patrones de consumo o producción, la optimización de procesos y la generación de informes de cumplimiento normativo.
* **Cumplimiento Normativo:** La capacidad de registrar y analizar datos operativos ayudará a las empresas de agua a cumplir con las regulaciones de calidad del agua, eficiencia energética y gestión ambiental. El sistema ha sido diseñado considerando las directrices de la UNE-EN 62443 (Ciberseguridad para Sistemas de Control Industrial y Automatización) y las mejores prácticas de la industria del agua, garantizando la resiliencia y la seguridad operativa de las infraestructuras críticas.
* **Ciberseguridad Reforzada:** Dada la criticidad de las infraestructuras de agua, SCADA23 integrará medidas de ciberseguridad desde el diseño (Security by Design), abordando la autenticación, autorización, cifrado de comunicaciones y segmentación de red para proteger el sistema contra ataques maliciosos y accesos no autorizados.
* **Fiabilidad y Redundancia:** Implementa mecanismos de alta disponibilidad para asegurar la continuidad de las operaciones críticas, como la monitorización ininterrumpida y el control remoto, minimizando el tiempo de inactividad.
* **Escalabilidad y Flexibilidad:** La arquitectura modular y el uso de tecnologías modernas (Django, FastAPI) aseguran que SCADA23 pueda crecer y adaptarse a medida que las redes de agua potable se expanden o evolucionan, incorporando nuevas instalaciones o tecnologías sin requerir una reingeniería completa.
* **Entorno Robusto:** Capacidad de operar en entornos industriales exigentes, integrándose con una variedad de dispositivos de campo y protocolos de comunicación estándar en la industria del agua, como Modbus/TCP.
* **Optimización de Recursos:** las funcionalidades de monitorización y análisis permiten optimizar el consumo energético de las bombas, reducir las pérdidas de agua y mejorar la calidad del agua distribuida.

## Sobre la plataforma SCADA23

La plataforma SCADA23 es un sistema de supervisión, control y adquisición de datos que habilita la operación optimizada y la toma de decisiones informada en las instalaciones de infraestructuras críticas. Su diseño se centra en la alta disponibilidad, la ciberseguridad robusta y la capacidad de adaptación a las particularidades del sector del agua. SCADA23 actúa como laboratorio de pruebas y simulación de PLCs, monitoriza y lee tráfico de PLCs reales y funciona 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.

**Funcionalidades**

Las funcionalidades clave de SCADA23 incluyen:

* **Adquisición de Datos en Tiempo Real:** Recopilación de datos de múltiples dispositivos de campo (sensores, actuadores, PLCs) con baja latencia.
* **Control Remoto de Activos:** Ejecución segura y fiable de comandos sobre equipos críticos como bombas, válvulas y compuertas.
* **Visualización de Procesos:** Presentación intuitiva del estado de las instalaciones a través de sinópticos dinámicos y paneles de control.
* **Gestión de Alarmas y Eventos:** Detección, notificación y registro de condiciones anormales y sucesos operativos.
* **Historización y Análisis:** Almacenamiento persistente de datos operativos para análisis de tendencias, informes y optimización de procesos.
* **Ciberseguridad Integrada:** Implementación de medidas de seguridad desde el diseño para proteger la infraestructura y los datos contra amenazas cibernéticas.

SCADA23 está concebida para ser una plataforma escalable y mantenible, capaz de gestionar desde pequeñas estaciones de bombeo hasta redes de distribución complejas, contribuyendo significativamente a la eficiencia operativa, la sostenibilidad y la seguridad en la gestión del agua potable.

**Principios de diseño**

La Plataforma SCADA23 es un desarrollo ambicioso que busca establecer un nuevo estándar en la supervisión y control de infraestructuras críticas. Su concepción se basa en principios de apertura, modularidad, escalabilidad y seguridad, aprovechando las ventajas de las últimas tecnologías en software de código abierto y arquitecturas distribuidas. Los principios de diseño en los que se basa son:

* **Modularidad:** El sistema está diseñado con una arquitectura modular que permite añadir o modificar componentes de forma independiente, facilitando el mantenimiento, la escalabilidad y la adaptación a futuras necesidades.
* **Escalabilidad:** SCADA23 está diseñado para crecer junto con las necesidades del cliente, pudiendo manejar desde pequeñas instalaciones hasta complejas redes distribuidas con miles de puntos de datos. La arquitectura de microservicios y el uso de bases de datos optimizadas para grandes volúmenes de datos en la Fase 2 son clave en este aspecto.
* **Flexibilidad:** La plataforma es agnóstica a la marca de los equipos de campo en la medida de lo posible, gracias a su soporte para múltiples protocolos industriales. Esto proporciona a los operadores la libertad de elegir los dispositivos que mejor se adapten a sus necesidades sin estar atados a un único proveedor.
* **Ciberseguridad desde el Diseño (Security by Design):** La seguridad es un pilar fundamental en el desarrollo de SCADA23. Desde las primeras etapas de diseño, se han incorporado consideraciones de ciberseguridad para proteger la integridad, confidencialidad y disponibilidad de los datos y el control de los sistemas. Esto incluye autenticación robusta, autorización basada en roles, cifrado de comunicaciones y registro de auditoría.
* **Orientación a Datos (Data-Driven):** La capacidad de recolectar, almacenar y analizar grandes volúmenes de datos en tiempo real y de forma histórica es una característica central. Esto permite a los operadores obtener información valiosa para la optimización de procesos, el mantenimiento predictivo y la toma de decisiones estratégicas.
* **Interfaz de Usuario Intuitiva:** Aunque la Fase 1 ofrece una interfaz básica con Tkinter, la Fase 2 pondrá un énfasis significativo en el desarrollo de una interfaz de usuario web (basada en Django y DRF) que sea intuitiva, fácil de usar y que proporcione una visión clara y completa del estado de los procesos.

**Componentes Principales**

Aunque los detalles se desarrollarán en capítulos posteriores, la Plataforma SCADA23, en su estado final (Fase 2 TRL7), se compondrá de los siguientes elementos clave:

* **Servidor SCADA:** El corazón del sistema, encargado de la adquisición de datos, el procesamiento, la gestión de alarmas, la lógica de control y la interacción con las bases de datos. En la Fase 2, esto se materializará principalmente a través de aplicaciones construidas con Django y FastAPI.
* **Módulos de Comunicación:** Responsables de la interacción con los dispositivos de campo utilizando los protocolos industriales pertinentes (Modbus TCP, PROFINET, OPC-UA).
* **Bases de Datos:** MySQL para datos relacionales y de configuración, e InfluxDB para series temporales.
* **Interfaz de Usuario (HMI):** Un cliente web interactivo que permite la visualización de datos, el control de procesos, la gestión de alarmas, el análisis de tendencias y la generación de informes.
* **Módulo de Ciberseguridad:** Componentes dedicados a la autenticación, autorización, cifrado y auditoría.
* **APIs RESTful:** Puntos de acceso para la integración con otros sistemas empresariales (ERP, GMAO, etc.) y para el desarrollo de aplicaciones cliente personalizadas.

La trayectoria de SCADA23, desde una validación inicial en TRL4 hasta una implementación completa en TRL7, demuestra un compromiso con la innovación, la fiabilidad y la excelencia tecnológica, con el objetivo de ofrecer una solución SCADA de vanguardia para la gestión de infraestructuras críticas.

**Repositorio**

El estado actual del proyecto se encuentra en la siguiente URL:

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

**Licencia**

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. La licencia de uso es de tipo Apache2.

**Beneficios de la Plataforma SCADA23 en el Sector del Agua Potable**

La implementación de SCADA23 en el sector del agua potable aportará beneficios sustanciales que se incrementarán drásticamente de la Fase 1 a la Fase 2:

**Beneficios en la Fase 1 (TRL4):**

* **Validación de Conectividad:** Demostración de la capacidad de comunicación con equipos industriales Modbus TCP.
* **Conocimiento Básico:** Permite a los desarrolladores y operadores familiarizarse con la adquisición y control de datos en un entorno controlado.
* **Base para el Futuro:** Establece la infraestructura de software inicial para el desarrollo de la Fase 2.
* **Comprensión de Procesos:** Ayuda a visualizar de forma básica el impacto de las operaciones de control.

**Beneficios en la Fase 2 (TRL7):**

* **Optimización Operacional:** Control preciso y automatizado de los procesos, reduciendo el consumo energético, el uso de productos químicos y las pérdidas de agua.
* **Mejora de la Eficiencia:** Identificación de ineficiencias mediante el análisis de datos históricos y en tiempo real.
* **Reducción de Costos:** Minimización de intervenciones manuales, optimización de recursos y reducción de fallos.
* **Aumento de la Fiabilidad del Suministro:** Monitorización constante y control proactivo que minimiza interrupciones en el servicio.
* **Mejora de la Calidad del Agua:** Monitoreo continuo de parámetros de calidad y control en tiempo real de los procesos de tratamiento.
* **Toma de Decisiones Informada:** Acceso a datos precisos y actualizados, así como herramientas de análisis e informes.
* **Ciberseguridad Reforzada:** Protección de infraestructuras críticas contra amenazas cibernéticas, asegurando la continuidad y la integridad de las operaciones.
* **Mantenimiento Predictivo:** Detección temprana de anomalías en equipos, permitiendo un mantenimiento proactivo y reduciendo el tiempo de inactividad.
* **Cumplimiento Normativo:** Facilidad para recopilar y reportar datos requeridos por las regulaciones de calidad del agua y ambientales.
* **Escalabilidad y Adaptabilidad:** Capacidad de integrar nuevas instalaciones, tecnologías y expandir el sistema a medida que crecen las necesidades de la empresa.

# REQUISITOS FUNCIONALES Y NO FUNCIONALES

Este capítulo detalla los requisitos que la Plataforma SCADA23 debe cumplir, categorizados en funcionales y no funcionales, incluyendo un apartado específico para la ciberseguridad. Se distinguirán los requisitos aplicables a la Fase 1 (TRL4) y aquellos que se alcanzarán en la Fase 2 (TRL7), reflejando la progresión y madurez de la plataforma.

## Requisitos Funcionales

Los requisitos funcionales generales describen las capacidades y funciones que el sistema SCADA23 debe realizar para cumplir con su propósito fundamental de supervisión, control y adquisición de datos.

**Pendiente, unificar y homogeneizar todos los requerimientos funcionales**

**Formato de los requisitos funcionales**

Los requisitos funcionales describen las funciones específicas que el sistema SCADA23 DEBE realizar. Cada requisito se presenta con un ID único, una descripción detallada, los actores involucrados, precondiciones, postcondiciones y criterios de aceptación verificables.

* **Formato de Requisito Funcional (RF-XXX):**
* **ID:** RF-YYY (Ej. RF-001)
* **Descripción:** Detalle claro y conciso de la funcionalidad.
* **Actor(es):** Entidades (humanas o de sistema) que interactúan con la funcionalidad.
* **Precondiciones:** Condiciones que deben cumplirse antes de ejecutar la funcionalidad.
* **Postcondiciones:** Estado del sistema después de la ejecución exitosa de la funcionalidad.
* **Criterios de Aceptación:** Medidas o pruebas específicas que demuestran que el requisito ha sido cumplido.

Se contemplan, para ambas fases los siguientes requerimientos funcionales.

**Para la Fase 1 (TRL4 - Demostración de Concepto):**

* **RFG-1.1:** Adquisición de Datos Básica. El sistema debe ser capaz de adquirir datos de dispositivos de campo a través del protocolo Modbus TCP.
  + *Detalle TRL4:* Limitado a la lectura de registros numéricos (Holding Registers, Input Registers, Coils, Discrete Inputs) de unos pocos dispositivos simulados o reales. La configuración de los puntos de datos será manual y estática.
* **RFG-1.2:** Visualización de Datos Básica. El sistema debe mostrar los datos adquiridos de forma rudimentaria.
  + *Detalle TRL4:* Una interfaz de usuario simple (Tkinter) que muestre los valores actuales de los puntos de datos configurados, posiblemente en formato de texto o tablas básicas.
* **RFG-1.3:** Control Básico. El sistema debe permitir el envío de comandos simples a dispositivos de campo.
  + *Detalle TRL4:* Posibilidad de escribir valores en Coils o Holding Registers a través de la interfaz de Tkinter, con confirmación visual de la acción.
* **RFG-1.4:** Comunicación Modbus TCP. El sistema debe establecer y mantener la comunicación con dispositivos que utilicen el protocolo Modbus TCP.
  + *Detalle TRL4:* Implementación de un cliente Modbus TCP para la lectura y escritura de datos. Manejo básico de errores de comunicación (e.g., reintentos simples).
* **RFG-1.5:** Configuración Estática. La configuración de los dispositivos y puntos de datos se realizará directamente en el código o en archivos de configuración planos.
  + *Detalle TRL4:* No existirá una interfaz de usuario para la configuración dinámica.
* **RFE-1.1: Lectura de Holdings Registers Modbus.** El script cliente en Python debe ser capaz de solicitar y recibir el valor de Holdings Registers específicos de un esclavo Modbus TCP.
* **RFE-1.2: Escritura de Coils Modbus.** El script cliente en Python debe ser capaz de cambiar el estado (ON/OFF) de Coils específicos de un esclavo Modbus TCP.
* **RFE-1.3: Interfaz Gráfica Básica Tkinter.** La aplicación SCADA de Tkinter debe mostrar al menos 5 puntos de datos (ej. 3 valores analógicos y 2 estados digitales) y permitir el control de 2 salidas digitales.
* **RFE-1.4: Conexión Única Modbus TCP.** La aplicación SCADA de Tkinter se conectará a una única dirección IP/Puerto Modbus TCP de un dispositivo simulado o real.
* **RFE-1.5: Actualización de Datos Periódica Simple.** Los datos en la interfaz de Tkinter se actualizarán con una frecuencia fija y predefinida (e.g., cada 5 segundos).

**Para la Fase 2 (TRL7 - Plataforma Completa):**

Todos los requisitos de la Fase 1 se mantendrán y se expandirán significativamente, además de los siguientes:

* **RFG-2.1:** Adquisición de Datos Avanzada. El sistema debe ser capaz de adquirir datos de una amplia variedad de dispositivos de campo, soportando múltiples protocolos industriales y configuraciones dinámicas.
  + *Detalle TRL7:* Soporte para Modbus TCP, PROFINET y OPC-UA. Configuración dinámica de dispositivos, tags (puntos de datos), frecuencias de muestreo, escalado y unidades a través de una interfaz de administración web.
* **RFG-2.2:** Visualización de Datos Completa (HMI/SCADA). El sistema debe proporcionar una interfaz de usuario gráfica completa y personalizable para la visualización en tiempo real de los procesos.
  + *Detalle TRL7:* Paneles de control (dashboards) personalizables, diagramas sinópticos con representación gráfica de equipos y procesos, tendencias en tiempo real e históricas, tablas de datos, visualización de estados y alarmas. Interfaz web responsiva desarrollada con frameworks modernos.
* **RFG-2.3:** Control Bidireccional Seguro. El sistema debe permitir el control seguro y auditado de equipos y procesos de campo.
  + *Detalle TRL7:* Envío de comandos con confirmación, validación de rangos, control de permisos de usuario (RBAC), registro de acciones de control para auditoría, y mecanismos de seguridad para evitar operaciones no autorizadas o erróneas.
* **RFG-2.4:** Gestión de Alarmas y Eventos. El sistema debe detectar, gestionar, notificar y registrar alarmas y eventos.
  + *Detalle TRL7:* Definición de umbrales para alarmas (altas, bajas, por cambio de estado), priorización de alarmas, historización de alarmas, reconocimiento de alarmas, filtros y búsqueda, y notificaciones configurables (e.g., email, SMS, push notifications).
* **RFG-2.5:** Gestión de Usuarios y Permisos. El sistema debe permitir la creación, modificación y gestión de usuarios y roles con permisos detallados sobre las funcionalidades y áreas del sistema.
  + *Detalle TRL7:* Autenticación basada en usuarios/contraseñas (con soporte para 2FA opcional), asignación de roles y permisos granulares (lectura, escritura, configuración) para diferentes partes de la interfaz, reportes o áreas geográficas.
* **RFG-2.6:** Gestión de Datos Históricos y Tendencias. El sistema debe almacenar eficientemente datos históricos y permitir su visualización y análisis.
  + *Detalle TRL7:* Almacenamiento de series temporales en InfluxDB, capacidad para visualizar tendencias históricas con rangos de tiempo configurables, zoom, comparación de variables y exportación de datos.
* **RFG-2.7:** Generación de Informes. El sistema debe ser capaz de generar informes personalizables sobre datos históricos, alarmas, eventos y rendimiento del sistema.
  + *Detalle TRL7:* Informes programados o bajo demanda, con opciones de filtrado, agregación y exportación a formatos comunes (PDF, CSV).
* **RFG-2.8:** Interoperabilidad a través de API RESTful. El sistema debe exponer una API RESTful para permitir la integración con otros sistemas de gestión empresarial (ERP, GMAO, etc.).
  + *Detalle TRL7:* API bien documentada (OpenAPI/Swagger), con mecanismos de autenticación y autorización para el acceso seguro a datos y funcionalidades del SCADA.
* **RFG-2.9:** Configuración Dinámica y Persistente. Toda la configuración del sistema (dispositivos, tags, alarmas, usuarios, vistas) debe ser persistente y configurable a través de la interfaz de usuario web.
  + *Detalle TRL7:* Almacenamiento en MySQL para la configuración y metadatos.
* **RFE-2.1: Soporte Multihilo/Asíncrono para Adquisición.** El backend (FastAPI/Django) debe ser capaz de gestionar múltiples conexiones y adquisiciones de datos de forma concurrente para maximizar el rendimiento.
  + *Detalle TRL7:* Uso de mecanismos asíncronos en FastAPI y/o Celery para tareas de adquisición de datos en segundo plano.
* **RFE-2.2: Conectividad Robusta con PROFINET.** Implementación de un conector robusto para la adquisición y control de datos a través de PROFINET, incluyendo el manejo de diferentes tipos de datos y configuraciones de dispositivos.
  + *Detalle TRL7:* Consideración de librerías Python específicas para PROFINET o integración con gateways.
* **RFE-2.3: Conectividad Robusta con OPC-UA.** Implementación de un cliente OPC-UA para la suscripción a datos, lectura de nodos y escritura de valores, con soporte para seguridad OPC-UA (cifrado y autenticación).
  + *Detalle TRL7:* Uso de librerías como asyncua para Python, con gestión de certificados y sesiones seguras.
* **RFE-2.4: Mapeo de Variables Personalizable.** Capacidad para mapear tags de dispositivos de campo a nombres lógicos dentro del SCADA, con escalado, unidades y tipos de datos configurables.
  + *Detalle TRL7:* Interfaz web para definir y modificar estos mapeos, almacenados en MySQL.
* **RFE-2.5: Histórico de Eventos y Auditoría.** Registro detallado de todas las acciones de los usuarios, cambios de configuración, inicios de sesión, y operaciones de control.
  + *Detalle TRL7:* Almacenamiento en MySQL con marcas de tiempo, usuario y descripción de la acción.
* **RFE-2.6: Gestión de Recetas y Lotes (Opcional).** El sistema podría soportar la creación y ejecución de recetas para automatizar secuencias de control complejas, aplicable a procesos específicos como dosificación en plantas de tratamiento.
  + *Detalle TRL7:* Interfaz para definir y ejecutar recetas, con seguimiento del progreso y registro de resultados.
* **RFE-2.7: Cálculo de KPIs y Variables Derivadas.** Capacidad para realizar cálculos sobre los datos adquiridos (ej. eficiencia, consumo, caudal medio) y generar nuevas variables.
  + *Detalle TRL7:* Módulo de cálculo configurable, con fórmulas matemáticas y lógicas basadas en tags existentes. Los resultados se almacenarán en InfluxDB.
* **RFE-2.8: Módulo de Mantenimiento Predictivo Básico.** A partir del análisis de datos históricos, el sistema podría ofrecer alertas o recomendaciones básicas para el mantenimiento de equipos.
  + *Detalle TRL7:* Detección de anomalías simples (e.g., valores fuera de rango sostenidos, cambios en patrones de consumo) a través de reglas configurables.
* **RFE-2.9: Notificaciones Personalizadas.** Los usuarios deben poder configurar qué tipo de alarmas o eventos desean recibir y por qué medios (e.g., email, SMS, interfaz de usuario).
  + *Detalle TRL7:* Módulo de notificación con plantillas y opciones de suscripción/desuscripción.

**Categoría: Adquisición y Procesamiento de Datos**

**ID: RF-001**

* **Descripción:** El sistema DEBE adquirir datos en tiempo real de los dispositivos de campo (sensores, actuadores, PLCs) utilizando los protocolos de comunicación configurados (ej., OPC UA, Modbus/TCP).
* **Actor(es):** Servidor de Adquisición de Datos, Dispositivos de Campo.
* **Precondiciones:** Los dispositivos de campo DEBEN estar conectados y operativos; la configuración de comunicación DEBE estar establecida en el sistema.
* **Postcondiciones:** Los valores de los tags configurados DEBEN ser leídos y procesados por el sistema a la frecuencia especificada.
* **Criterios de Aceptación:**
  + Para 1000 tags configurados, el sistema DEBE actualizar los valores en la base de datos de series temporales con una latencia máxima de 500 ms el 99% de las veces.
  + El sistema DEBE soportar la adición dinámica de nuevos tags sin requerir reinicio del Servidor de Adquisición.
  + El sistema DEBE registrar errores de comunicación (ej., timeout, conexión perdida) con un nivel de severidad ERROR.

**ID: RF-002**

* **Descripción:** El sistema DEBE historizar los datos de los tags configurados en la base de datos de series temporales.
* **Actor(es):** Servidor de Adquisición de Datos, Base de Datos de Series Temporales.
* **Precondiciones:** RF-001 cumplido; la base de datos de series temporales DEBE estar operativa.
* **Postcondiciones:** Los datos adquiridos DEBEN ser almacenados persistentemente, incluyendo el valor del tag y un timestamp de alta precisión.
* **Criterios de Aceptación:**
  + Los datos de los tags DEBEN ser almacenados con una resolución de milisegundos (ms).
  + El sistema DEBE retener los datos históricos por un mínimo de 12 meses, configurable por el administrador del sistema.
  + La tasa de ingesta DEBE ser de al menos 10.000 puntos/segundo sin degradación del rendimiento.

**Categoría: Control y Actuación**

**ID: RF-003**

* **Descripción:** El sistema DEBE permitir a un operador autorizado iniciar/parar remotamente bombas, abrir/cerrar válvulas y ejecutar otros comandos definidos sobre dispositivos de campo a través de la interfaz de usuario.
* **Actor(es):** Operador, Servidor de Control, Dispositivos de Campo.
* **Precondiciones:** El operador DEBE estar autenticado y autorizado para el dispositivo específico; el dispositivo DEBE estar en un estado que permita el comando (ej., bomba en Parada para Iniciar).
* **Postcondiciones:** El comando DEBE ser enviado al dispositivo; el estado del dispositivo DEBE actualizarse en el sistema; un evento de auditoría DEBE ser registrado.
* **Criterios de Aceptación:**
  + El tiempo desde que el operador pulsa el botón en la UI hasta que el comando es enviado al dispositivo NO EXCEDERÁ de 2 segundos.
  + El sistema DEBE validar las precondiciones del dispositivo antes de enviar el comando y mostrar un mensaje de error claro al operador si la validación falla.
  + El sistema DEBE requerir una confirmación explícita del operador antes de ejecutar cualquier comando de control.
* **ID:** RF-004
* **Descripción:** El sistema DEBE gestionar y mantener el estado de cada dispositivo de campo (ej., 'Parada', 'Arrancando', 'Funcionando', 'Fallida', 'Mantenimiento').
* **Actor(es):** Servidor de Adquisición de Datos, Servidor de Control, Lógica del Sistema.
* **Precondiciones:** RF-001 cumplido; los dispositivos DEBEN proporcionar su estado actual o permitir inferirlo de otros tags.
* **Postcondiciones:** El sistema DEBE reflejar con precisión el estado actual y los estados transitorios de cada dispositivo; las transiciones de estado DEBEN ser gestionadas según reglas predefinidas.
* **Criterios de Aceptación:**
  + Para una bomba, el sistema DEBE transicionar de Arrancando a Funcionando en menos de 5 segundos una vez que se detecta un caudal superior a 0.5 L/s.
  + Si una bomba pasa a estado Fallida, el sistema DEBE generar una alarma de alta prioridad.
  + La lógica de estado DEBE ser configurable para diferentes tipos de dispositivos sin requerir cambios en el código base.

**Categoría: Alarmas y Eventos**

**ID: RF-005**

* **Descripción:** El sistema DEBE generar alarmas automáticamente cuando los valores de los tags excedan los umbrales preconfigurados (alto/bajo, muy alto/muy bajo).
* **Actor(es):** Motor de Alarmas, Tags de Dispositivo.
* **Precondiciones:** Los umbrales de alarma DEBEN estar configurados para el tag; el tag DEBE estar adquiriendo datos.
* **Postcondiciones:** Se DEBE generar una alarma con su ID único, descripción, timestamp, valor actual, umbral violado y nivel de severidad; la alarma DEBE ser visible en la interfaz de usuario.
* **Criterios de Aceptación:**
  + El tiempo desde que un valor excede un umbral hasta que la alarma es generada NO EXCEDERÁ de 1 segundo.
  + El sistema DEBE soportar al menos 4 niveles de severidad de alarma (informativa, baja, media, alta, crítica).
  + Los umbrales de alarma DEBEN ser configurables por el administrador del sistema.

**ID: RF-006**

* **Descripción:** El sistema DEBE permitir a los operadores reconocer y silenciar alarmas activas a través de la interfaz de usuario.
* **Actor(es):** Operador, Motor de Alarmas.
* **Precondiciones:** El operador DEBE estar autenticado y autorizado; la alarma DEBE estar activa.
* **Postcondiciones:** El estado de la alarma DEBE cambiar a "Reconocida" o "Silenciada"; el registro de la alarma DEBE actualizarse con el operador y timestamp de la acción.
* **Criterios de Aceptación:**
  + La interfaz de usuario DEBE mostrar claramente el estado de reconocimiento de cada alarma.
  + El sistema DEBE prevenir que un operador reconozca una alarma si no tiene los permisos adecuados.
  + Las alarmas silenciadas DEBEN dejar de emitir notificaciones sonoras/visuales pero seguir siendo visibles en la lista de alarmas.

**Categoría: Visualización e Interfaz de Usuario**

**ID: RF-007**

* **Descripción:** El sistema DEBE mostrar sinópticos dinámicos que representen gráficamente el proceso del agua potable, actualizando los valores de los tags y los estados de los dispositivos en tiempo real.
* **Actor(es):** Operador, Interfaz de Usuario.
* **Precondiciones:** El operador DEBE estar logueado; los datos de los tags DEBEN estar disponibles.
* **Postcondiciones:** El sinóptico correspondiente DEBE cargar y mostrar los datos actualizados de forma continua.
* **Criterios de Aceptación:**
  + Los sinópticos DEBEN actualizarse con una frecuencia NO INFERIOR a 1 segundo para los datos en tiempo real.
  + La carga inicial de un sinóptico promedio (ej., 50 elementos dinámicos) NO EXCEDERÁ de 3 segundos.
  + Los elementos gráficos (bombas, válvulas, niveles de tanques) DEBEN cambiar su apariencia visual según el estado o valor del tag asociado (ej., color rojo para bomba Fallida).

**ID: RF-008**

* **Descripción:** El sistema DEBE proporcionar herramientas para visualizar datos históricos en gráficos de tendencias configurables.
* **Actor(es):** Operador.
* **Precondiciones:** Datos históricos DEBEN estar disponibles para los tags seleccionados.
* **Postcondiciones:** Se DEBE generar un gráfico de tendencia para los tags y el rango de tiempo seleccionado, con opciones de zoom y navegación.
* **Criterios de Aceptación:**
  + El sistema DEBE permitir comparar simultáneamente datos de hasta 5 tags en un mismo gráfico.
  + La carga de un gráfico con 1000 puntos de datos para cada uno de 3 tags NO EXCEDERÁ de 5 segundos.
  + El usuario DEBE poder exportar los datos mostrados en el gráfico a un formato CSV.

**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). El usuario SCADA podrá *controlar* (leer y escribir valores) y también *configurar* (crear/modificar/eliminar) elementos como tags, alarmas, usuarios, etc.
* **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 "Tratamiento de 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.).

**RFUNC006: Detalle de Registro de Datos (Historización):**

* **Descripción:** se define la **frecuencia de muestreo** para cada parámetro (Presión, Temperatura, Flujo, Nivel) en segundos para cada segundo que cada minuto. Se especifica el **periodo de almacenamiento** de datos históricos. Se utiliza para la historización una base de datos MySQL, planteando para el futuro InfluxDB.

**RFUNC007: Gestión de Alarmas:**

* **Descripción:** Para la gestión de las alarmas se necesita acuse de recibo por parte del operador. Se registran en un log de eventos. Hay prioridades de alarma. Las acciones automáticas se disparan con cada alarma (aparte de las ya definidas para el nivel (ejemplo: un paro de emergencia general ante ciertos fallos críticos)

**RFUNC008: Control de Válvulas desde HMI:**

* **Descripción:** existe alguna lógica de interbloqueo o seguridad para el control de válvulas (Ej. no permitir cerrar una válvula de salida si el nivel es bajo y se está llenando). Se necesita confirmación del operador para acciones de control críticas.

**RFUNC009: Simulación de fallos:**

* **Descripción:** la simulación permite la **introducción de fallos** para probar la robustez del sistema SCADA (ej. simular un sensor que da lecturas erróneas, una válvula que no responde, etc.)

## Requisitos No Funcionales

Los requisitos no funcionales definen los atributos de calidad del sistema, como rendimiento, disponibilidad, escalabilidad, seguridad y usabilidad. Deben ser cuantificables y medibles. Describen las cualidades del sistema, cómo debe comportarse.

**Para la Fase 1 (TRL4 - Demostración de Concepto):**

* **RNFG-1.1: Rendimiento Básico.** El sistema debe ser capaz de adquirir y mostrar datos sin retrasos significativos para un número limitado de puntos.
  + *Detalle TRL4:* Actualización de la interfaz en Tkinter cada pocos segundos sin bloqueos evidentes.
* **RNFG-1.2: Fiabilidad Básico.** La comunicación Modbus TCP debe ser razonablemente estable durante periodos cortos de demostración.
  + *Detalle TRL4:* Manejo de desconexiones temporales con reintentos simples.
* **RNFG-1.3: Facilidad de Uso (Desarrollo).** El código debe ser comprensible para futuros desarrolladores.
  + *Detalle TRL4:* Comentarios claros y estructura simple de los scripts Python y la aplicación Tkinter.
* **RNFG-1.4: Seguridad (Consideración Inicial).** Reconocimiento de la necesidad de seguridad, pero con implementación limitada.
  + *Detalle TRL4:* No se implementará autenticación de usuario ni cifrado de comunicaciones en esta fase. Se asumirá un entorno de laboratorio controlado.
* **RNFE-1.1: Uso de Python Standard Library.** La aplicación debe depender principalmente de la librería estándar de Python y tkinter para la interfaz.
* **RNFE-1.2: Entorno de Ejecución Sencillo.** Los scripts y la aplicación Tkinter deben ejecutarse en un entorno Python básico sin necesidad de configuraciones complejas o bases de datos externas.
* **RNFE-1.3: Bajo Consumo de Recursos (Demostración).** Para la demostración, el sistema debe funcionar en hardware limitado (e.g., un portátil estándar).

**Para la Fase 2 (TRL7 - Plataforma Completa):**

Todos los requisitos de la Fase 1 se mantendrán y se elevará su nivel de exigencia, además de los siguientes:

* **RNFG-2.1: Rendimiento Avanzado.** El sistema debe ser capaz de procesar y visualizar grandes volúmenes de datos en tiempo real con una latencia mínima.
  + *Detalle TRL7:* Tiempo de respuesta de las APIs para la adquisición de datos inferior a 100ms. Capacidad de manejar miles de puntos de datos con frecuencias de muestreo de hasta 1 segundo. Interfaz de usuario fluida y reactiva.
* **RNFG-2.2: Escalabilidad Horizontal y Vertical.** El sistema debe poder escalar para soportar un número creciente de dispositivos, puntos de datos, usuarios y carga de trabajo.
  + *Detalle TRL7:* Arquitectura basada en microservicios (Django, FastAPI) que permita la distribución de componentes en múltiples servidores. Bases de datos optimizadas para grandes volúmenes de datos (InfluxDB) y concurrencia (MySQL).
* **RNFG-2.3: Fiabilidad y Alta Disponibilidad.** El sistema debe ser resiliente a fallos y ofrecer un alto grado de disponibilidad.
  + *Detalle TRL7:* Implementación de mecanismos de redundancia (ej. balanceo de carga, clúster de bases de datos), manejo robusto de errores de comunicación, reintentos inteligentes, logs detallados y mecanismos de recuperación automática. Monitorización de la salud del sistema.
* **RNFG-2.4: Mantenibilidad.** El sistema debe ser fácil de mantener, actualizar y depurar.
  + *Detalle TRL7:* Código bien documentado, arquitectura modular, uso de contenedores (Docker) para despliegue consistente, pruebas unitarias e integración, y un pipeline CI/CD.
* **RNFG-2.5: Usabilidad.** La interfaz de usuario debe ser intuitiva, fácil de aprender y eficiente para los operadores.
  + *Detalle TRL7:* Diseño de Interfaz de Usuario (UI) y Experiencia de Usuario (UX) profesional, coherente y ergonómico. Minimización de clics y pasos para tareas comunes.
* **RNFG-2.6: Portabilidad.** El sistema debe ser desplegable en diferentes entornos operativos.
  + *Detalle TRL7:* Uso de tecnologías multiplataforma (Python, web), contenedorización (Docker) para aislar dependencias y asegurar la consistencia del entorno.
* **RNFG-2.7: Seguridad.** El sistema debe proteger los datos y el control de accesos no autorizados, ataques cibernéticos y vulnerabilidades.
  + *Detalle TRL7:* Implementación de ciberseguridad desde el diseño, autenticación robusta, autorización granular, cifrado de datos en tránsito y en reposo, auditoría de seguridad y conformidad con estándares de seguridad industrial.
* **RNFG-2.8: Documentación.** El sistema debe estar acompañado de documentación técnica y de usuario completa y actualizada.
  + *Detalle TRL7:* Manual de instalación, manual de usuario, manual de administración, documentación de la API, diagramas de arquitectura.
* **RNFE-2.1: Soporte de Navegadores Modernos.** La interfaz web debe ser compatible y funcionar correctamente en las últimas versiones de los navegadores web más populares (Chrome, Firefox, Edge, Safari).
* **RNFE-2.2: Tiempo de Actividad del 99.9% (24/7).** El sistema debe estar operativo el 99.9% del tiempo, excluyendo ventanas de mantenimiento planificadas.
  + *Detalle TRL7:* Implica estrategias de alta disponibilidad y recuperación ante desastres.
* **RNFE-2.3: Replicación de Datos.** Las bases de datos deben soportar replicación para garantizar la persistencia de datos y la continuidad del servicio.
  + *Detalle TRL7:* Configuración de MySQL en modo de replicación (ej. master-slave o clúster Galera) e InfluxDB en modo de alta disponibilidad.
* **RNFE-2.4: Tiempo de Carga de Páginas Inferior a 2 Segundos.** La mayoría de las páginas de la interfaz de usuario web deben cargar en menos de 2 segundos bajo condiciones de carga normales.
* **RNFE-2.5: Resiliencia ante Fallos de Conexión Industrial.** El sistema debe ser capaz de manejar la pérdida temporal de comunicación con dispositivos industriales sin perder datos acumulados y con reconexión automática.
  + *Detalle TRL7:* Buffering de datos en memoria o en disco local, colas de mensajes para datos en tiempo real.
* **RNFE-2.6: Soporte para Múltiples Idiomas (i18n).** La interfaz de usuario debe permitir la selección de diferentes idiomas (al menos español e inglés).
  + *Detalle TRL7:* Implementación de internacionalización en Django y en el frontend.
* **RNFE-2.7: Rendimiento de Consulta de Series Temporales.** Las consultas a InfluxDB para tendencias históricas de un día deben ser respondidas en menos de 1 segundo para un máximo de 10 variables.
* **RNFE-2.8: Optimización de Recursos.** El sistema debe hacer un uso eficiente de los recursos de CPU, memoria y disco, especialmente en el almacenamiento de datos de series temporales.
  + *Detalle TRL7:* Configuración de políticas de retención de datos en InfluxDB, compresión de datos y optimización de consultas SQL.

**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.
  + Requerimientos de control de acceso y autenticación (ej. usuario/contraseña, roles).
  + Requisitos de cifrado para las comunicaciones
  + Niveles de acceso requeridos (ej. operador, supervisor, administrador). Cada rol tendrá permisos diferentes sobre qué puede ver y controlar.
* **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.

**Categoría: Rendimiento**

**ID: RNF-001 (Latencia de Adquisición)**

* **Descripción:** La latencia máxima entre la actualización de un valor en un dispositivo de campo y su reflejo en la base de datos de series temporales NO DEBE EXCEDER 500 milisegundos para el 99% de los tags, bajo carga operativa normal.
* **Métrica:** Tiempo promedio de actualización de tags.
* **Prueba:** Monitoreo del tiempo de ciclo de adquisición de datos y comparación con timestamps de dispositivo/base de datos.

**ID: RNF-002 (Tiempo de Respuesta UI)**

* **Descripción:** El tiempo de carga y renderizado de la interfaz de usuario para sinópticos con hasta 100 elementos dinámicos NO DEBE EXCEDER 3 segundos. El tiempo de respuesta para comandos de control críticos (ej., iniciar/parar bomba) en la interfaz de usuario NO DEBE EXCEDER 2 segundos.
* **Métrica:** Tiempo de carga de página, tiempo de respuesta a eventos de UI.
* **Prueba:** Pruebas de carga con usuarios concurrentes, medición de tiempos de respuesta de UI.
* **ID:** RNF-003 (Capacidad de Procesamiento de Alarmas)
* **Descripción:** El motor de alarmas DEBE ser capaz de procesar la generación y actualización de hasta 1000 alarmas por minuto sin impactar negativamente el rendimiento de la adquisición de datos o la UI.
* **Métrica:** Tasa de procesamiento de alarmas, consumo de CPU/memoria del motor de alarmas.
* **Prueba:** Simulación de generación masiva de alarmas.

**Categoría: Disponibilidad**

**ID: RNF-004 (Uptime del Sistema)**

* **Descripción:** La plataforma SCADA23 DEBE garantizar un tiempo de actividad (uptime) anual mínimo del 99.95% para los componentes críticos (Servidor de Adquisición, Servidor de Control, Base de Datos).
* **Métrica:** Porcentaje de tiempo operativo.
* **Prueba:** Monitoreo continuo de servicios, pruebas de failover.
* **ID:** RNF-005 (Recuperación ante Fallos)
* **Descripción:** En caso de fallo de un componente primario (ej., Servidor de Adquisición), el sistema DEBE realizar un failover automático a un componente secundario (hot-standby) en un máximo de 60 segundos, sin pérdida de datos.
* **Métrica:** Tiempo de recuperación, pérdida de datos.
* **Prueba:** Inyección de fallos en componentes primarios.

**Categoría: Escalabilidad**

**ID: RNF-006 (Escalabilidad de Tags)**

* **Descripción:** El sistema DEBE ser capaz de soportar la gestión y adquisición de datos de hasta 50.000 tags distribuidos en múltiples instalaciones, con la posibilidad de expandirse a 100.000 tags mediante la adición de recursos de hardware sin reestructuración mayor de la arquitectura.
* **Métrica:** Número máximo de tags soportados, consumo de recursos por tag.
* **Prueba:** Pruebas de carga y estrés con un número creciente de tags.
* **ID:** RNF-007 (Escalabilidad de Usuarios)
* **Descripción:** El sistema DEBE soportar simultáneamente a 50 usuarios concurrentes en la interfaz de usuario sin degradación notable del rendimiento (tiempo de respuesta de UI < 5 segundos).
* **Métrica:** Número de usuarios concurrentes, tiempo de respuesta.
* **Prueba:** Pruebas de carga multi-usuario.

**Categoría: Mantenibilidad y Operabilidad**

**ID: RNF-008 (Registro de Eventos)**

* **Descripción:** El sistema DEBE registrar todos los eventos operativos, de seguridad y de sistema con información detallada (timestamp, nivel de severidad, componente, mensaje, ID de usuario si aplica) en un sistema de logs centralizado, con retención mínima de 6 meses.
* **Métrica:** Cobertura y detalle de logs, capacidad de almacenamiento.
* **Prueba:** Revisión de logs, pruebas de estrés para asegurar la persistencia de logs bajo carga.
* **ID:** RNF-009 (Facilidad de Configuración)
* **Descripción:** La configuración de nuevos tags, dispositivos, alarmas y usuarios DEBE poder realizarse a través de una interfaz gráfica de usuario por parte de un administrador, sin necesidad de modificación del código.
* **Métrica:** Tiempo para configurar un nuevo elemento, número de pasos.
* **Prueba:** Pruebas de usabilidad para el rol de administrador.

## Requisitos de Ciberseguridad

**Introducción**

Los requisitos de ciberseguridad son fundamentales para proteger la infraestructura crítica del agua potable contra amenazas cibernéticas, siguiendo los principios de "Security by Design".

De forma general se utilizarán los controles definidos en los siguientes estándares:

* ISA IEC 62443-3-3 y/o ISA IEC 62443-4-2 para los controles técnicos.
* Especificaciones técnicas para el HMI según norma ISA 101, incluyendo:
  + a) inicio de sesión temporal con privilegios elevados (capacidad de aumentar los derechos de usuario sin cerrar sesión completamente) para tareas específicas,
  + b) concepto de múltiples roles y privilegios dentro de una aplicación (restricción basada en roles),
  + c) restricción del acceso a la información basada en el ámbito de autoridad del usuario,
  + d) restricción de contenido basada en la ubicación,
  + e) uso de firmas electrónicas,
  + f) notas de autenticación (requisito de que un usuario añada una razón para una acción de control),
  + g) uso de biometría.

**Buenas prácticas**

La seguridad es un requisito no funcional crítico para la plataforma SCADA23. Además de los requisitos generales, se adoptarán las siguientes prácticas específicas durante la codificación y el despliegue en un entorno Windows para mitigar riesgos:

**Para la Fase 1 (TRL4 - Demostración de Concepto):**

* **RGC-1.1: Conciencia de Seguridad.** Aunque la implementación será limitada, se debe tener en cuenta la importancia de la ciberseguridad para futuras fases.
  + *Detalle TRL4:* No se expondrán servicios Modbus TCP a redes no confiables. Se operará en una red de laboratorio aislada.
* **RGC-1.2: Prácticas de Codificación Segura Básicas.** Aplicar buenas prácticas de codificación para evitar vulnerabilidades obvias (ej. inyección de código, desbordamiento de búfer).
  + *Detalle TRL4:* Validación básica de entradas en los scripts.
* **RCE-1.1: Uso de Puertos Estándar Modbus TCP.** La comunicación se realizará a través del puerto 502 estándar para Modbus TCP.
* **RCE-1.2: Dependencia de Entorno Aislado.** La seguridad en esta fase dependerá en gran medida de que el entorno de demostración sea completamente aislado de redes externas.

**Para la Fase 2 (TRL7 - Plataforma Completa):**

* **RGC-2.1: Autenticación Robusta.** El sistema debe garantizar que solo usuarios autorizados puedan acceder al sistema.
  + *Detalle TRL7:* Soporte para contraseñas complejas, hashes de contraseñas seguros (BCrypt, Argon2), bloqueo de cuentas tras múltiples intentos fallidos, y opcionalmente autenticación de dos factores (2FA).
* **RGC-2.2: Autorización Basada en Roles (RBAC).** El sistema debe aplicar políticas de autorización que limiten el acceso a funciones y datos específicos según el rol del usuario.
  + *Detalle TRL7:* Definición de roles (Administrador, Operador, Supervisor, Solo Lectura) con permisos detallados sobre vistas, acciones de control y configuraciones.
* **RGC-2.3: Cifrado de Comunicaciones.** Todas las comunicaciones sensibles (interfaz de usuario, APIs, comunicación con dispositivos si el protocolo lo soporta) deben ser cifradas.
  + *Detalle TRL7:* Uso de HTTPS/TLS para la interfaz web y APIs. Consideración de TLS para Modbus TCP (Modbus TCP Security) y OPC-UA security profiles.
* **RGC-2.4: Integridad de Datos.** El sistema debe proteger los datos contra modificaciones no autorizadas, garantizando su fiabilidad.
  + *Detalle TRL7:* Uso de checksums, hashes o firmas digitales donde sea aplicable (ej. para actualizaciones de firmware, archivos de configuración).
* **RGC-2.5: Registro de Auditoría y Trazabilidad.** Todas las acciones relevantes del sistema y de los usuarios deben ser registradas para fines de auditoría y análisis forense.
  + *Detalle TRL7:* Logs inmutables que registren quién hizo qué, cuándo y dónde. Protección de los logs contra modificaciones.
* **RGC-2.6: Gestión de Vulnerabilidades.** El proceso de desarrollo debe incluir un ciclo de vida de desarrollo seguro (SDLC) con pruebas de seguridad.
  + *Detalle TRL7:* Escaneo de vulnerabilidades, pruebas de penetración (externas o internas), gestión de parches para librerías y dependencias.
* **RGC-2.7: Respaldo y Recuperación de Datos.** El sistema debe soportar copias de seguridad regulares y un plan de recuperación de desastres.
  + *Detalle TRL7:* Copias de seguridad automáticas de bases de datos y configuraciones, con almacenamiento seguro y validación de la recuperabilidad.
* **RGC-2.8: Endurecimiento del Sistema.** El sistema operativo y los componentes de software (servidores web, bases de datos) deben ser configurados de forma segura.
  + *Detalle TRL7:* Eliminación de servicios innecesarios, uso de firewalls, políticas de contraseñas robustas, monitorización de la seguridad.
* **RGC-2.9: Segmentación de Red (Recomendación).** Aunque es una preocupación de infraestructura, el diseño del SCADA debe facilitar la implementación de segmentación de red.
  + *Detalle TRL7:* Arquitectura que permita el aislamiento de las redes OT y IT, y la segregación de los componentes del SCADA.
* **RCE-2.1: Gestión Segura de Credenciales.** Las credenciales (contraseñas de base de datos, claves API) no deben estar codificadas en el código fuente ni almacenadas en texto plano.
  + *Detalle TRL7:* Uso de variables de entorno, servicios de gestión de secretos (ej. HashiCorp Vault, Kubernetes Secrets) o gestores de configuración seguros.
* **RCE-2.2: Protección contra Ataques Comunes de Aplicaciones Web.** El sistema debe protegerse contra vulnerabilidades como inyección SQL, Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF).
  + *Detalle TRL7:* Frameworks como Django incorporan protecciones para muchos de estos ataques. Pruebas de seguridad estáticas y dinámicas (SAST/DAST).
* **RCE-2.3: Control de Sesiones Seguro.** Las sesiones de usuario deben ser gestionadas de forma segura.
  + *Detalle TRL7:* Uso de cookies seguras (HttpOnly, Secure), timeouts de sesión configurables, invalidación de sesiones al cierre de sesión y regeneración de IDs de sesión.
* **RCE-2.4: Auditoría de Acceso a Datos.** El sistema debe registrar el acceso a datos sensibles, incluyendo quién accedió, cuándo y desde dónde.
  + *Detalle TRL7:* Logs específicos de acceso a datos que puedan ser revisados para identificar patrones anómalos.
* **RCE-2.5: Actualizaciones de Seguridad.** Se debe establecer un proceso para aplicar regularmente parches de seguridad al sistema operativo, librerías y frameworks utilizados.
  + *Detalle TRL7:* Uso de herramientas de gestión de dependencias y alertas de vulnerabilidades.
* **RCE-2.6: Monitoreo de Seguridad.** Implementación de herramientas de monitoreo (SIEM) para detectar actividades sospechosas o ataques.
  + *Detalle TRL7:* Envío de logs del SCADA a un sistema centralizado de gestión de eventos e información de seguridad.
* **RCE-2.7: Hardening de Protocolos Industriales.** Configuración segura de los protocolos de comunicación industrial cuando sea posible.
  + *Detalle TRL7:* Uso de TLS para OPC-UA, y evaluación de Modbus TCP Security si el hardware lo soporta. Restricción de acceso a puertos industriales a IPs autorizadas.
* **RCE-2.8: Protección contra Denial of Service (DoS).** Implementación de medidas para mitigar ataques de denegación de servicio.
  + *Detalle TRL7:* Limitación de tasas (rate limiting) en APIs, configuración de firewalls, uso de proxies inversos.
* **RCE-2.9: Gestión de Certificados (para OPC-UA/TLS).** El sistema debe gestionar de forma segura los certificados digitales para las comunicaciones cifradas (especialmente para OPC-UA).
  + *Detalle TRL7:* Generación, almacenamiento y renovación seguros de certificados de cliente y servidor.

**Otros requerimientos**

* **Manejo Seguro de Credenciales:**
  + **Nunca codificar credenciales directamente en el código.** Todas las credenciales (bases de datos, APIs externas, accesos a dispositivos) se gestionarán mediante variables de entorno de Windows o archivos de configuración cifrados (ej., utilizando bibliotecas de cifrado de Python como cryptography para claves AES).
  + Se limitará el acceso a estos secretos a solo los usuarios de sistema y servicios que los necesiten, bajo el principio de privilegio mínimo.
  + Para información extremadamente sensible, se podría considerar el uso del Gestor de Credenciales de Windows o herramientas de terceros.
* **Validación Estricta de Entradas (Input Validation):**
  + Todas las entradas de datos de usuarios (a través de la HMI) y de dispositivos (lecturas de sensores, comandos) serán validadas rigurosamente.
  + Se evitarán inyecciones de código (SQL Injection, Command Injection) mediante el uso de parámetros parametrizados en las consultas a bases de datos (SQLAlchemy ORM) y la sanitización de todas las entradas antes de su procesamiento o ejecución.
  + Las validaciones se realizarán tanto en el frontend como, y especialmente, en el backend.
* **Gestión de Sesiones y Autenticación/Autorización:**
  + Se implementarán mecanismos robustos de autenticación de usuarios (ej., basados en tokens JWT si la HMI es web).
  + Se aplicará un control de acceso basado en roles (RBAC) para la autorización, asegurando que los usuarios solo puedan realizar las acciones para las que tienen permisos (ej., un operador solo puede ver ciertos datos, un administrador puede modificar configuraciones).
  + Las sesiones de usuario deben tener tiempos de vida limitados y gestionarse de forma segura.
* **Principios de Mínimo Privilegio (Windows):**
  + Los servicios de Windows que ejecutan los componentes de SCADA23 deben configurarse para ejecutarse con las cuentas de usuario de sistema dedicadas y con los permisos mínimos necesarios. Evitar ejecutar servicios como LocalSystem a menos que sea estrictamente necesario y se comprendan los riesgos.
  + Los directorios de la aplicación y de datos deben tener permisos NTFS restringidos para que solo el usuario del servicio y los administradores autorizados puedan acceder a ellos.
* **Actualizaciones y Parches de Seguridad:**
  + Se establecerá un proceso para la monitorización de vulnerabilidades conocidas en las bibliotecas de Python y frameworks utilizados (pip-audit u otras herramientas de SCA - Software Composition Analysis).
  + Se aplicarán rápidamente los parches de seguridad del sistema operativo Windows y de Python.
* **Logging y Auditoría de Seguridad:**
  + Se registrarán todos los eventos de seguridad relevantes (intentos de inicio de sesión fallidos, cambios de configuración, comandos críticos ejecutados) con marcas de tiempo y detalles del usuario/origen.
  + Estos logs se escribirán en archivos que pueden ser configurados para rotación y, si es necesario, integrados con el Visor de Eventos de Windows o un sistema de gestión de eventos e información de seguridad (SIEM).
* **Comunicación Segura:**
  + Toda la comunicación entre componentes del sistema (backend HMI a base de datos, módulos internos) se protegerá utilizando cifrado (ej., TLS/SSL) cuando sea posible, especialmente si atraviesan redes no confiables.
  + Las interfaces de red expuestas al exterior (ej., API de HMI) deberán estar protegidas con TLS/HTTPS.
  + Se configurará el Firewall de Windows para permitir solo el tráfico necesario hacia y desde la aplicación SCADA23.
* **Manejo de Errores Detallado (Seguridad):**
  + Los mensajes de error expuestos al usuario final no deben revelar información sensible sobre la arquitectura interna del sistema, las rutas de archivos o los detalles de la base de datos. Se usarán mensajes genéricos y se registrarán los detalles técnicos internamente.

La integración de estas prácticas de seguridad en cada etapa del ciclo de vida del desarrollo contribuirá significativamente a la resiliencia de la plataforma SCADA23 frente a las ciberamenazas en un entorno Windows.

**Categoría: Autenticación y Autorización**

**ID: RCS-001 (Autenticación Robusta de Usuarios)**

* **Descripción:** El sistema DEBE implementar un mecanismo de autenticación robusto para todos los usuarios, soportando políticas de contraseñas complejas (longitud mínima 12 caracteres, combinación de mayúsculas, minúsculas, números, símbolos) y la posibilidad de autenticación multifactor (MFA).
* **Estándar:** Cumplimiento con NIST SP 800-63B.
* **Criterios de Aceptación:**
  + El sistema DEBE forzar el cambio de contraseñas iniciales y requerir el cambio periódico de contraseñas (cada 90 días).
  + El sistema DEBE bloquear cuentas después de 5 intentos fallidos de login durante un período de 15 minutos.
  + El sistema DEBE soportar integración con directorios externos (ej., LDAP/Active Directory) para la gestión de usuarios.

**ID: RCS-002 (Control de Acceso Basado en Roles - RBAC)**

* **Descripción:** El sistema DEBE implementar un control de acceso basado en roles (RBAC) que restrinja las funcionalidades y los datos accesibles para cada usuario en función de su rol (ej., Administrador, Operador, Visualizador).
* **Modelo:** Basado en el principio de mínimo privilegio.
* **Criterios de Aceptación:**
  + Un usuario con rol "Visualizador" NO DEBE poder ejecutar comandos de control ni modificar configuraciones.
  + Un usuario con rol "Operador" DEBE poder ejecutar comandos de control pero NO DEBE poder añadir/eliminar usuarios o dispositivos.
  + Los roles y permisos asociados DEBEN ser configurables por un "Administrador de Seguridad".

**Categoría: Seguridad de la Comunicación**

**ID: RCS-003 (Cifrado de Comunicaciones Externas)**

* **Descripción:** Todas las comunicaciones entre los clientes de la interfaz de usuario (navegadores web, aplicaciones de escritorio) y el servidor SCADA23 DEBEN estar cifradas utilizando protocolos seguros (ej., TLS v1.2 o superior).
* **Estándar:** RFC 5246 (TLS 1.2), RFC 8446 (TLS 1.3).
* **Criterios de Aceptación:**
  + El sistema DEBE utilizar certificados digitales válidos y emitidos por una autoridad de confianza.
  + El acceso HTTP plano DEBE ser deshabilitado.
  + Las conexiones OPC UA DEBEN utilizar seguridad de mensajes y autenticación de usuarios.

**ID: RCS-004 (Cifrado de Comunicaciones con Dispositivos Críticos)**

* **Descripción:** Las comunicaciones entre el Servidor de Adquisición y los dispositivos de campo que soporten cifrado (ej., OPC UA con seguridad) DEBEN ser cifradas para proteger la integridad y confidencialidad de los datos de control.
* **Estándar:** IEC 62443.
* **Criterios de Aceptación:**
  + Si el dispositivo de campo soporta OPC UA, se DEBE configurar el nivel de seguridad "Sign & Encrypt".
  + Para protocolos no cifrados, se DEBE evaluar e implementar soluciones de segmentación de red y VPNs si la arquitectura lo permite.

**Categoría: Integridad y Resiliencia**

**ID: RCS-005 (Validación de Datos de Entrada)**

* **Descripción:** El sistema DEBE validar rigurosamente todas las entradas de datos (desde usuarios, dispositivos de campo o APIs) para prevenir ataques de inyección (SQL Injection, Command Injection) y garantizar la integridad de los datos.
* **Metodología:** White-listing, saneamiento de entradas.
* **Criterios de Aceptación:**
  + Cualquier intento de inyección DEBE ser detectado y bloqueado, y el evento DEBE ser registrado.
  + Los campos de entrada en la UI DEBEN aplicar validaciones de formato y longitud.

**ID: RCS-006 (Registro de Auditoría de Seguridad)**

* **Descripción:** El sistema DEBE registrar todos los eventos de seguridad relevantes (intentos de login exitosos/fallidos, cambios de permisos, ejecución de comandos de control, modificaciones de configuración, accesos a datos sensibles) con un timestamp preciso y el ID del usuario/sistema.
* **Criterios de Aceptación:**
  + Los logs de auditoría DEBEN ser inmutables y protegidos contra modificaciones no autorizadas.
  + Los logs DEBEN ser accesibles solo para roles autorizados y DEBEN poder ser exportados para análisis.
  + La retención de logs de seguridad DEBE ser de al menos 12 meses.

**ID: RCS-007 (Gestión de Vulnerabilidades)**

* **Descripción:** El desarrollo de SCADA23 DEBE seguir un proceso de gestión de vulnerabilidades que incluya escaneos de seguridad periódicos, análisis de código estático/dinámico, y la aplicación oportuna de parches de seguridad a las librerías y componentes de terceros.
* **Metodología:** OWASP Top 10, CWE.
* **Criterios de Aceptación:**
  + Todas las dependencias de terceros DEBEN ser escaneadas regularmente para vulnerabilidades conocidas (CVEs).
  + Las vulnerabilidades de severidad "Alta" o "Crítica" identificadas en el código base o dependencias DEBEN ser corregidas en un plazo máximo de 7 días.

# PROCESO DEL SECTOR AGUA POTABLE

Este capítulo describe en detalle el proceso de gestión del agua potable que la plataforma SCADA23 debe supervisar y controlar. Comprender este proceso es esencial para el diseño y la implementación de la lógica de control, la visualización de datos y la gestión de alarmas específicas del dominio.

## Introducción al proceso ETAP

El ciclo del agua potable, en el contexto de una empresa de gestión de recursos hídricos, abarca una serie de etapas interconectadas, cada una con sus propios desafíos operativos y requisitos de control. Estas etapas incluyen:

* **Captación**

Recolección y extracción de agua de fuentes naturales (ríos, acuíferos, embalses, pozos).

* + Equipos: Bombas de captación, válvulas de compuerta, filtros de entrada.
  + Puntos de Medida: Nivel de embalse/pozo, caudal de captación, calidad del agua cruda (turbidez, pH).
  + Puntos de Control: Arranque/parada de bombas de captación, apertura/cierre de válvulas.
* **Pretratamiento**

Eliminación de sólidos gruesos y ajuste inicial de la calidad.

* + Equipos: Rejas de desbaste, desarenadores, estaciones de dosificación de pre-oxidantes.
  + Puntos de Medida: Caudal de entrada, niveles en depósitos, concentración de químicos.
  + Puntos de Control: Control de velocidad de rejas, dosificación de químicos.
* **Tratamiento Principal**

Proceso físico-químicos para adecuar el agua cruda a los estándares de potabilidad, eliminando impurezas físicas, químicas y biológicas. (Coagulación-Floculación-Decantación-Filtración), en una (ETAP - Estación de Tratamiento de Agua Potable):

* + Equipos: Agitadores, tanques de floculación, decantadores, filtros de arena/carbón.
  + Puntos de Medida: Caudal de entrada/salida, niveles en tanques, presión en filtros, turbidez post-filtración.
  + Puntos de Control: Velocidad de agitadores, dosificación de coagulantes/floculantes, control de retrolavado de filtros.
* **Desinfección: Eliminación de microorganismos patógenos.**
  + Equipos: Sistemas de dosificación de cloro/hipoclorito, lámparas UV.
  + Puntos de Medida: Concentración de desinfectante residual, caudal.
  + Puntos de Control: Control de dosificación de desinfectante.
* **Almacenamiento y Distribución:**

Almacenamiento de agua tratada en depósitos y tanques para acumular agua tratada, asegurando el suministro y la presión para su posterior bombeo a la red mediante la red de tuberías y estaciones de bombeo para transportar el agua desde los depósitos hasta los puntos de consumo.

* + Equipos: Depósitos de agua tratada, bombas de impulsión a red, válvulas de red, estaciones de bombeo intermedias.
  + Puntos de Medida: Nivel de depósitos, caudal de salida, presión en la red de distribución (en puntos clave), calidad del agua en red (cloro residual).
  + Puntos de Control: Arranque/parada de bombas de impulsión, apertura/cierre de válvulas de sectorización, control de presión en red.
* **Consumo**

Uso del agua por parte de usuarios residenciales, comerciales e industriales.

* **Saneamiento**

En las EDAR - Estación Depuradora de Aguas Residuales, Recolección y tratamiento de aguas residuales antes de su vertido al medio natural.

## Subprocesos y Puntos de Control en el Sector del Agua Potable

A continuación, se describen los subprocesos más relevantes del sector del agua potable y los puntos de control y supervisión típicos que SCADA23 gestionará.

**Captación y Abastecimiento**

* **Descripción:** Implica la extracción de agua de sus fuentes naturales y su transporte inicial.
* **Puntos de Control/Medida Típicos:**
  + **Caudalímetros:** Medición del volumen de agua captada.
  + **Nivel de Agua:** En embalses o pozos.
  + **Bombas de Captación:** Estado (marcha/paro), horas de funcionamiento, consumo energético.
  + **Válvulas:** Estado (abierta/cerrada, posición).
  + **Calidad del Agua Cruda:** Parámetros básicos como pH, turbidez (en Fase 2).
* **Rol de SCADA23 (Fase 1 TRL4):**
  + **Adquisición:** Lectura de caudal (Modbus TCP en caudalímetros compatibles), estado de bombas (ON/OFF) y válvulas (abierta/cerrada) a través de Modbus TCP.
  + **Control:** Envío de comandos Modbus TCP para arrancar/parar bombas o abrir/cerrar válvulas de forma manual desde la interfaz Tkinter.
  + **Visualización:** Mostrar valores numéricos básicos en la interfaz de usuario.
* **Rol de SCADA23 (Fase 2 TRL7):**
  + **Adquisición:** Ampliación a más parámetros de calidad (pH, conductividad, oxígeno disuelto, etc.) a través de Modbus TCP, PROFINET o OPC-UA. Monitoreo de presión en líneas de impulsión.
  + **Control:** Control automático de bombas basado en niveles de embalse o demanda, regulación PID de válvulas para mantener caudales o presiones objetivo. Implementación de secuencias operacionales.
  + **Visualización:** Diagramas sinópticos detallados con representación gráfica de bombas, válvulas, caudalímetros, niveles. Tendencias históricas y en tiempo real de todos los parámetros.
  + **Alarmas:** Detección de caudales anómalos, bajos niveles críticos, fallos de bomba.
  + **Optimización:** Integración con sistemas de pronóstico de demanda para optimizar la captación.

**Estación de Tratamiento de Agua Potable (ETAP)**

* **Descripción:** Conjunto de procesos físicos, químicos y biológicos para purificar el agua.
* **Puntos de Control/Medida Típicos:**
  + **Caudalímetros:** A la entrada y salida de las diferentes etapas (floculación, decantación, filtración).
  + **Nivel de Tanques:** En floculadores, decantadores, filtros y tanques de agua tratada.
  + **Dosificación de Productos Químicos:** Caudal y concentración de coagulantes, desinfectantes (cloro, ozono), correctores de pH.
  + **Bombas dosificadoras:** Estado, velocidad/frecuencia, alarmas.
  + **Filtros:** Presión diferencial, ciclos de lavado.
  + **Parámetros de Calidad del Agua:** Turbidez, pH, cloro residual, conductividad, temperatura, etc., en diferentes puntos del proceso.
  + **Válvulas y Actuadores:** Para control de flujo, mezcla, y procesos de lavado.
* **Rol de SCADA23 (Fase 1 TRL4):**
  + **Adquisición:** Lectura de caudal a la entrada/salida (Modbus TCP), nivel de un tanque de agua tratada, estado de una bomba dosificadora, turbidez (si es un sensor Modbus TCP).
  + **Control:** Arranque/Paro de una bomba dosificadora, apertura/cierre de una válvula Modbus TCP.
  + **Visualización:** Mostrar los valores básicos y el estado de los componentes.
* **Rol de SCADA23 (Fase 2 TRL7):**
  + **Adquisición:** Monitoreo exhaustivo de todos los parámetros de proceso y calidad del agua en tiempo real. Integración de analizadores en línea vía Modbus TCP, PROFINET, OPC-UA.
  + **Control:** Control PID de dosificación de productos químicos basado en calidad del agua y caudal. Gestión automática de ciclos de lavado de filtros. Optimización del proceso de mezcla.
  + **Visualización:** Sinópticos detallados de cada unidad de proceso (pretratamiento, coagulación-floculación, decantación, filtración, desinfección, almacenamiento), con indicadores gráficos, animaciones y alarmas contextuales.
  + **Alarmas:** Múltiples niveles de alarma para parámetros de calidad (alto/bajo pH, alto/bajo cloro residual, alta turbidez), fallos de equipos, interrupciones de proceso.
  + **Trazabilidad:** Registro histórico de todos los parámetros de calidad, dosis de productos, horas de funcionamiento de equipos para cumplimiento normativo y optimización.
  + **Gestión de Recetas:** Posiblemente para secuencias de arranque/parada o lavado de filtros.

**Almacenamiento y Distribución**

* **Descripción:** Gestión de depósitos intermedios y finales, y la red de tuberías que lleva el agua a los consumidores.
* **Puntos de Control/Medida Típicos:**
  + **Nivel de Depósitos:** Crucial para asegurar el suministro y la presión en la red.
  + **Presión en la Red:** En puntos clave para detectar fugas, roturas o anomalías.
  + **Caudal en la Red:** Medición de caudales de entrada y salida de zonas, consumo.
  + **Bombas de Impulsión:** Estado, velocidad/frecuencia, consumo energético, horas de funcionamiento.
  + **Válvulas Reguladoras de Presión/Caudal:** Posición, estado.
  + **Calidad del Agua:** Monitoreo de cloro residual en la red para garantizar la potabilidad.
* **Rol de SCADA23 (Fase 1 TRL4):**
  + **Adquisición:** Lectura de nivel de un depósito (Modbus TCP), estado de una bomba de impulsión, presión en un punto de la red.
  + **Control:** Arrancar/parar una bomba de impulsión, abrir/cerrar una válvula de corte.
  + **Visualización:** Mostrar los niveles, presiones y estados.
* **Rol de SCADA23 (Fase 2 TRL7):**
  + **Adquisición:** Monitoreo de niveles en múltiples depósitos, presiones en zonas de la red, caudales en entradas y salidas de sectores. Integración con sensores de presión y caudal remotos.
  + **Control:** Control automático de bombas para mantener niveles óptimos en depósitos y presiones en la red. Regulación de válvulas para sectorización y gestión de fugas.
  + **Visualización:** Mapas geográficos de la red con visualización de niveles de depósitos, presión en nodos, estado de bombas. Perfiles de nivel históricos y en tiempo real.
  + **Alarmas:** Bajos niveles críticos en depósitos, altas/bajas presiones en la red, fallos de bomba.
  + **Detección de Fugas:** Posibilidad de integrar algoritmos sencillos de balance hídrico basados en datos de caudal y presión para identificar potenciales fugas.
  + **Optimización Energética:** Programación de bombas en base a tarifas eléctricas y niveles de depósito.

**Monitorización de Puntos Remotos (Telecontrol)**

* **Descripción:** Superar las barreras geográficas para supervisar y controlar activos distribuidos.
* **Puntos de Control/Medida Típicos:** Cualquier punto de los subprocesos anteriores ubicado en zonas remotas, donde la comunicación se realiza a través de redes WAN (3G/4G/5G, radio, fibra).
* **Rol de SCADA23 (Fase 1 TRL4):**
  + **Comunicación:** Establecimiento de conexiones Modbus TCP a través de red IP (asumiendo que los dispositivos remotos tienen conectividad IP). No se abordará la complejidad de las redes WAN ni la resiliencia en esta fase.
* **Rol de SCADA23 (Fase 2 TRL7):**
  + **Comunicación Robusta:** Gestión de la latencia y fiabilidad de las comunicaciones en entornos WAN. Implementación de mecanismos de reintento, buffering y compresión de datos.
  + **Seguridad:** Cifrado de comunicaciones (VPNs o TLS si el protocolo lo permite) para proteger la transmisión de datos a y desde ubicaciones remotas.
  + **Redundancia:** Opciones de rutas de comunicación alternativas para asegurar la continuidad en caso de fallos de red.
  + **Gestión de Dispositivos Remotos:** Capacidad para monitorizar el estado de la comunicación con cada dispositivo remoto y generar alarmas por desconexión.

**Flujo de Datos y Operaciones Típicas**

El flujo de datos en SCADA23 seguirá un patrón claro, evolucionando en complejidad entre las dos fases:

**Flujo de Datos (Fase 1 TRL4):**

1. **Adquisición:** Scripts Python en línea de comandos (simulando clientes de campo o actuando como un Gateway Modbus básico) envían datos Modbus TCP al programa SCADA (Python/Tkinter). El SCADA también realiza lecturas activas.
2. **Procesamiento:** El programa SCADA recibe los paquetes Modbus TCP, extrae los valores de los registros y los convierte a un formato numérico simple.
3. **Visualización:** Los valores numéricos procesados se muestran directamente en la interfaz de Tkinter.
4. **Control:** El operador introduce un comando en la interfaz de Tkinter, que el programa SCADA traduce a un comando Modbus TCP (write\_coil, write\_register) y lo envía al dispositivo.

**Flujo de Datos (Fase 2 TRL7):**

1. **Adquisición:** Módulos de adquisición (posiblemente microservicios FastAPI) se conectan a dispositivos de campo vía Modbus TCP, PROFINET y OPC-UA. Estos módulos obtienen datos en tiempo real.
2. **Procesamiento y Normalización:** Los datos crudos de los dispositivos son procesados, escalados, convertidos a unidades de ingeniería y asociados a sus respectivos tags lógicos.
3. **Almacenamiento:**
   * **InfluxDB:** Datos de series temporales (mediciones de sensores, actuadores, estados digitales) se almacenan en InfluxDB con su marca de tiempo.
   * **MySQL:** Metadatos (configuración de tags, dispositivos, usuarios, alarmas), eventos y datos de auditoría se almacenan en MySQL.
4. **Lógica de Negocio/Alarmas:** El backend (Django/FastAPI) monitorea los datos en tiempo real, compara con umbrales definidos en MySQL para activar alarmas, evalúa lógicas de control y gestiona eventos.
5. **API RESTful:** El backend expone APIs RESTful (Django REST Framework) para que los clientes web y otros sistemas puedan acceder a datos en tiempo real, históricos, configuración y funcionalidades de control.
6. **Visualización (HMI Web):** La interfaz de usuario web interactúa con las APIs RESTful para obtener los datos y representarlos en paneles de control, sinópticos, tendencias, etc. Los operadores envían comandos de control a través de la interfaz, que viajan por las APIs al backend.
7. **Notificaciones:** El sistema de alarmas envía notificaciones a los usuarios autorizados a través de los canales configurados.

## Diagramas de flujo del proceso

Se pretende incluirán en este apartado los Diagramas de Flujo del Proceso (PFD - Process Flow Diagrams) detallados para cada estación (ETAP, EBAP) y la red de distribución, incluyendo:

* Identificación de todos los equipos principales (bombas, válvulas, tanques, filtros).¡
* Sensores y actuadores (con etiquetas claras).
* Flujo del agua (indicado con flechas).
* Puntos de interconexión.

Se recomienda que los Diagramas de tuberías e instrumentación (P&ID) muestren las interconexiones detalladas de la instrumentación, incluyendo los diagramas PFD/P&ID pertinentes, referenciando un anexo o un repositorio de diagramas si son muy extensos. Cada elemento en el diagrama tiene un identificador único que se corresponde con la instrumentación y los tags SCADA.)

**Pendiente**

## Estados operaciones del Sistema y transiciones

El sistema SCADA23 DEBE ser capaz de operar y gestionar los siguientes estados generales del proceso y sus transiciones:

**Estado: INICIALIZACIÓN**

* **Descripción:** El sistema se está arrancando o recuperando de un fallo. Los equipos están en un estado conocido de seguridad (ej., parados, válvulas cerradas).
* **Transiciones permitidas:** A OPERACIÓN NORMAL (si todos los subsistemas están operativos y el proceso es seguro), a PARADA DE EMERGENCIA (si se detecta una condición crítica durante el arranque).

**Estado: OPERACIÓN NORMAL**

* **Descripción:** El proceso de agua potable está funcionando de acuerdo con los parámetros preestablecidos. El SCADA está supervisando y permitiendo el control habitual.
* **Transiciones permitidas:** A MODO MANTENIMIENTO (para intervenciones programadas), a PARADA DE EMERGENCIA (ante una condición de fallo grave o alarma crítica).

**Estado: MODO MANTENIMIENTO**

* **Descripción:** Un segmento o la totalidad del proceso está en un estado de mantenimiento programado. Ciertas funcionalidades de control pueden estar bloqueadas o ser manuales.
* **Transiciones permitidas:** A OPERACIÓN NORMAL (una vez finalizado el mantenimiento y verificada la seguridad), a INICIALIZACIÓN (si se requiere un reinicio completo).

**Estado: PARADA DE EMERGENCIA**

* **Descripción:** El proceso ha sido detenido de forma abrupta debido a una condición crítica (ej., alarma de seguridad, fallo de equipo principal). El sistema debe llevar el proceso a un estado seguro.
* **Transiciones permitidas:** A INICIALIZACIÓN (una vez resuelta la causa de la emergencia y el sistema esté listo para un reinicio controlado).

## Reglas de negocio generales y validaciones del sistema

Además de la descripción de los casos de uso, es fundamental definir explícitamente las **Reglas de Negocio** y las **Validaciones** inherentes a la lógica operativa del sistema SCADA23. Estas reglas dictan el comportamiento del sistema bajo diversas condiciones, aseguran la integridad de los datos y previenen operaciones incorrectas o peligrosas. Serán implementadas como parte integral del código de control, la lógica de la HMI y la gestión de la base de datos.

Las reglas de negocio y validaciones clave incluyen, pero no se limitan a:

* **Interbloqueos y Secuencias Operativas:**
  + **Bloqueo de Arranque de Bomba por Válvula Cerrada:** Una bomba (ej., bomba de impulsión, bomba de llenado) **no podrá ser arrancada** si su válvula de descarga o de entrada asociada (ej., Válvula Z, Válvula de Salida de Tanque) se encuentra en estado 'Cerrada'. El sistema deberá verificar el estado de la válvula antes de permitir la orden de arranque y, en caso de estar cerrada, deberá negar la acción y notificar al operador.
  + **Secuencia de Arranque/Parada de Bombeo:** Para el llenado o vaciado de tanques, se establecerá una secuencia lógica para la activación de bombas y la apertura/cierre de válvulas. Por ejemplo, una válvula puede requerir estar abierta un tiempo mínimo antes de que la bomba asociada pueda arrancar, o la bomba debe pararse antes de que una válvula se cierre completamente.
  + **Protección por Bajo Nivel de Tanque de Aspiración:** Una bomba de impulsión **no podrá arrancar** si el nivel del tanque del cual aspira (ej., Tanque de Almacenamiento) se encuentra por debajo de un umbral crítico de nivel mínimo (Nivel\_Min\_Aspiración) para evitar el funcionamiento en seco y daños a la bomba.
* **Validación de Rangos y Umbrales:**
  + **Niveles de Tanque:** El nivel del agua en los tanques (ej., Tanque de Almacenamiento, Tanque Elevado) **no podrá superar el 95%** de su capacidad total (Nivel\_Max\_Seguro) para evitar desbordamientos. Si se detecta un nivel por encima de este umbral, el sistema deberá generar una alarma de "Nivel Alto" y, si está en modo automático, iniciar acciones correctivas (ej., parar bomba de llenado, abrir válvula de alivio).
  + Así mismo, el nivel del agua **no podrá descender por debajo del 5%** de su capacidad total (Nivel\_Min\_Seguro) para asegurar un suministro continuo o evitar el funcionamiento en seco de las bombas. Si se detecta un nivel por debajo de este umbral, el sistema deberá generar una alarma de "Nivel Bajo" y, si está en modo automático, iniciar acciones correctivas (ej., arrancar bomba de llenado).
  + **Valores de Sensores:** Todas las lecturas de los sensores (presión, caudal, temperatura, etc.) deben ser validadas contra rangos lógicos predefinidos (ej., presión no puede ser negativa, caudal no puede exceder la capacidad máxima de la tubería). Las lecturas fuera de rango se marcarán como "inválidas" y podrán generar alarmas de fallo de sensor.
* **Modos de Operación (Automático/Manual):**
  + Las acciones de control sobre actuadores (ej., arrancar/parar bombas, abrir/cerrar válvulas) solo podrán ser ejecutadas por el operador si el equipo o el proceso en cuestión está en modo 'Manual'.
  + Si el sistema está en modo 'Automático', las órdenes manuales sobre equipos bajo control automático serán ignoradas o requerirán una confirmación específica para anular temporalmente el modo automático.
  + El cambio de modo de operación (de automático a manual o viceversa) debe ser una acción explícita del operador y, en algunos casos, puede requerir una confirmación o credenciales de un nivel superior.
* **Gestión de Alarmas y Eventos:**
  + Las alarmas críticas (ej., fallo de bomba, nivel extremo) deben requerir acuse de recibo explícito por parte del operador antes de que puedan ser silenciadas o borradas.
  + El registro de eventos debe ser inmutable y capturar la marca de tiempo, el usuario o el origen del sistema, la acción realizada o el evento detectado, y cualquier valor relevante.

Estas reglas de negocio y validaciones serán la base para el desarrollo de la lógica de control y la validación de entrada de datos en la implementación Python, asegurando la fiabilidad y seguridad del sistema SCADA23.

## Hojas de Datos de Instrumentos

Aunque las hojas de datos físicas de los instrumentos son documentos externos, la plataforma SCADA23 DEBE ser capaz de modelar y procesar la información relevante que estas hojas contienen. Esta sección define la estructura de datos esperada para los diferentes tipos de instrumentos que el sistema supervisará y controlará.

Esta sección es un borrador para el modelado de datos de dispositivos, con idea de que esto se traduzca directamente en el esquema de la base de datos y la lógica de la capa de acceso a datos. Es crucial que esta estructura sea estable para minimizar futuros cambios en el backend y el frontend.

**Estructura de Datos General para Instrumentos**

Cada instrumento (sensor o actuador) que se integra en SCADA23 DEBE tener un conjunto de atributos base:

* **ID\_Instrumento (String):** Identificador único del instrumento (ej., "BOMBA\_ETAP\_001", "VALVULA\_DIST\_005").
* **Nombre (String):** Nombre descriptivo y legible (ej., "Bomba de Impulsión P-1", "Válvula de Sectorización V-3").
* **Tipo\_Instrumento (Enum):** Clasificación del equipo (ej., BOMBA, VALVULA, SENSOR\_NIVEL, CAUDALIMETRO, SENSOR\_PRESION, CLORADOR, etc.).
* **Fabricante (String):** Nombre del fabricante del equipo.
* **Modelo (String):** Modelo específico del equipo.
* **Ubicacion (String):** Descripción de la ubicación física (ej., "ETAP - Sala de Bombas", "Depósito Principal - Entrada").
* **Estado\_Operacional (Enum):** Estado actual del instrumento (OPERATIVO, FALLO, MANTENIMIENTO, FUERA\_DE\_SERVICIO).
* **Ultima\_Actualizacion (Timestamp):** Último momento en que se recibió un dato o se actualizó el estado.
* **Protocolo\_Comunicacion (Enum):** Protocolo utilizado para comunicarse con este dispositivo (ej., OPC\_UA, MODBUS\_TCP, PROFINET).
* **Direccion\_Comunicacion (String):** Dirección IP, puerto, ID de esclavo, o endpoint del servidor OPC UA.

**Atributos Específicos por Tipo de Instrumento**

Además de los atributos generales, cada tipo de instrumento tendrá atributos específicos:

* **Tipo: BOMBA**
  + Estado\_Bomba (Enum): PARADA, ARRANCANDO, FUNCIONANDO, FALLIDA, MANTENIMIENTO.
  + Comando\_Operacion (Boolean/Enum): INICIAR, PARAR.
  + Caudal\_Actual (Float, unidades: L/s o m³/h): Caudal de bombeo actual.
  + Presion\_Salida (Float, unidades: Bar): Presión de salida de la bomba.
  + Consumo\_Energetico (Float, unidades: kW): Consumo eléctrico actual.
  + Tiempo\_Funcionamiento\_Acumulado (Float, unidades: Horas): Tiempo total que la bomba ha estado en operación.
  + Numero\_Arranques\_Acumulado (Integer): Número de veces que la bomba ha sido arrancada.
* **Tipo: VALVULA**
  + Estado\_Valvula (Enum): ABIERTA, CERRADA, ABRIENDO, CERRANDO, POSICION\_INTERMEDIA, FALLIDA.
  + Comando\_Operacion (Enum): ABRIR, CERRAR, POSICIONAR\_A\_X (para válvulas modulantes).
  + Posicion\_Actual\_Porcentaje (Float, rango 0-100): Para válvulas modulantes.
* **Tipo: SENSOR\_NIVEL**
  + Nivel\_Actual (Float, unidades: metros o porcentaje): Nivel de líquido en tanque/depósito.
  + Nivel\_Min\_Alarma (Float): Umbral mínimo para alarma.
  + Nivel\_Max\_Alarma (Float): Umbral máximo para alarma.
* **Tipo: CAUDALIMETRO**
  + Caudal\_Actual (Float, unidades: L/s o m³/h): Caudal instantáneo.
  + Caudal\_Acumulado (Float, unidades: m³): Caudal totalizado.
  + Caudal\_Min\_Alarma (Float): Umbral mínimo para alarma.
  + Caudal\_Max\_Alarma (Float): Umbral máximo para alarma.
* **Tipo: SENSOR\_PRESION**
  + Presion\_Actual (Float, unidades: Bar o kPa): Presión instantánea.
  + Presion\_Min\_Alarma (Float): Umbral mínimo para alarma.
  + Presion\_Max\_Alarma (Float): Umbral máximo para alarma.
* **Tipo: SENSOR\_CALIDAD\_AGUA (ej., turbidez, pH, cloro residual)**
  + Valor\_Actual (Float, unidades específicas).
  + Unidad\_Medida (String).
  + Min\_Alarma, Max\_Alarma (Float).

**Identificadores únicos para sensores y actuadores**

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.

Se asignan identificadores únicos a cada sensor y actuador (válvulas) en los tres tanques.

**a. Sensores de Presión (PS)**

* **Identificadores:** PS-101 (Sensor de Presión Tanque 1); PS-102 (Sensor de Presión Tanque 2); PS-103 (Sensor de Presión 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 (Sensor de Temperatura Tanque 1); TS-102 (Sensor de Temperatura Tanque 2); TS-103 (Sensor de Temperatura 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 (Sensor de Flujo Tanque 1); FS-102 (Sensor de Flujo Tanque 2); FS-103 (Sensor de Flujo 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 (Sensor de Nivel Tanque 1); LS-102 (Sensor de Nivel Tanque 2); LS-103 (Sensor de Nivel 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 (XVI)**

* **Identificadores:** XVI-101 (Válvula de entrada Tanque 1); XVI-102 (Válvula de entrada Tanque 2); XVI-103 (Válvula de entrada 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 (XVO)**

* **Identificadores:** XVO-101 (Válvula de salida Tanque 1); XVO-102 (Válvula de salida Tanque 2); XVO-103 (Válvula de salida 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).

**Nota:** respecto los estados de las válvulas, son on/off (digitales) con estados de abrir y cerrar, si bien en el futuro tendrán un control proporcional (analógico).

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

El sistema de la **Estación de Tratamiento de Agua Potable (ETAP)** simulará de forma integral el proceso de **purificación y distribución de agua**, centrándose en el control y monitorización de tres fases clave: **Oxidación, Clarificación y Almacenamiento**. Cada fase se representa mediante un tanque específico (T1, T2, T3), permitiendo la supervisión detallada de sus parámetros operativos y el control de las válvulas asociadas. Este sistema simulará tanto el llenado como el vaciado de los tanques, reflejando el flujo del proceso de tratamiento.

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**

Para asegurar la **calidad del agua** y la eficiencia del proceso, cada tanque mantendrá los siguientes rangos operativos normales para sus parámetros clave:

* **Presión:** De **2 a 6 bar**. La presión se monitoriza para garantizar un flujo adecuado y evitar sobrecargas en el sistema.
* **Temperatura:** De **15 a 30 °C**. La temperatura es crucial para la efectividad de los procesos químicos y biológicos de tratamiento.
* **Flujo:** De **30 a 70 L/min**. El flujo indica la velocidad a la que el agua se mueve a través de cada etapa del tratamiento.
* **Nivel:** De **20% a 80%**. El nivel óptimo garantiza la capacidad de proceso y evita desbordamientos o vaciados excesivos.

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

* El sistema estará equipado con un robusto esquema de **alarmas y puntos de control** para reaccionar ante desviaciones y garantizar la continuidad y seguridad del proceso:
* **Alarma de Presión Alta:** Se activará si la presión **supera los 8 bar**, indicando una posible obstrucción o sobrepresión crítica.
* **Alarma de Presión Baja:** Se activará si la presión **es inferior a 1 bar**, señalando una posible fuga o falta de suministro.
* **Alarma de Temperatura Alta:** Se activará si la temperatura **supera los 90 °C**, lo que podría indicar un mal funcionamiento en los equipos de calentamiento o una reacción exotérmica anómala.
* **Alarma de Temperatura Baja:** Se activará si la temperatura **es inferior a 5 °C**, lo que podría afectar la eficiencia de los tratamientos químicos o biológicos.
* **Alarma de Flujo Alto:** Se activará si el flujo **supera los 95 L/min**, sugiriendo una demanda excesiva o un fallo en el control de caudal.
* **Alarma de Flujo Bajo:** Se activará si el flujo **es inferior a 10 L/min**, indicando una posible obstrucción o un problema en el suministro.
* **Alarma de Nivel Alto:** Se activará si el nivel del tanque **supera el 95%**. Ante esta alarma, el sistema automáticamente **detendrá el llenado** y, si es necesario, **abrirá la válvula de salida** para evitar un desbordamiento.
* **Alarma de Nivel Bajo:** Se activará si el nivel del tanque **es inferior al 5%**. En respuesta, el sistema automáticamente **iniciará el llenado** y, si es necesario, **cerrará la válvula de salida** para evitar que el tanque se vacíe por completo.
* **Control de Válvulas:** La interfaz hombre-máquina (HMI) permitirá al operador tener un control manual completo sobre las **válvulas de entrada y salida** de cada tanque, facilitando la intervención directa en el proceso.

**c. Descripción de los Tanques por Etapa del Proceso**

El proceso de tratamiento se simulará a través de tres tanques interconectados, cada uno representando una etapa fundamental:

* **Tanque T1: Oxidación.** En esta etapa inicial, el agua cruda se somete a un proceso de oxidación. Aquí se monitorizará y controlará la dosificación de oxidantes (no simulada directamente, pero implícita en el proceso) para precipitar impurezas y preparar el agua para la clarificación.
* **Tanque T2: Clarificación.** El agua del Tanque T1 fluye hacia el Tanque T2, donde se simulará el proceso de sedimentación o floculación. Aquí, las partículas suspendidas se agrupan y se asientan, separándose del agua. El control de nivel y flujo es crítico para la eficiencia de este proceso.
* **Tanque T3: Almacenamiento.** Una vez clarificada, el agua tratada se almacena en el Tanque T3, el **tanque de agua potable**. Este tanque representa la reserva de agua lista para ser distribuida, y su nivel es fundamental para asegurar el suministro continuo a la red. El control de su válvula de salida es clave para la distribución.

Este diseño funcional sienta las bases para el desarrollo del sistema SCADA descrito a continuación, permitiendo una simulación realista y un control efectivo de un proceso de tratamiento de agua.

## Lógica de control específica

Las siguientes reglas de negocio y lógica de control deben ser implementada para el comportamiento del sistema. Se recomienda el uso de tablas para cada regla de negocio si la complejidad lo permite, o diagramas de estado/flujo para lógicas más complejas.

**Regla RB-001 (Control de Nivel de Depósito):**

* + **Descripción:** La bomba de impulsión Bomba\_Imp\_X DEBE iniciar automáticamente si el nivel del depósito Dep\_Principal desciende por debajo de 2.5 metros.
  + **Condición de Activación:** Dep\_Principal.Nivel < 2.5 metros
  + **Acción:** Bomba\_Imp\_X.Iniciar()
  + **Condición de Desactivación:** Bomba\_Imp\_X DEBE parar automáticamente si el nivel del depósito Dep\_Principal supera los 4.8 metros.
  + **Acción:** Bomba\_Imp\_X.Parar()

**Regla RB-002 (Protección por Baja Presión):**

* + **Descripción:** Si la presión en la tubería de salida Tubería\_Salida\_Y cae por debajo de 1.2 Bar durante más de 15 segundos, el sistema DEBE:
    1. Generar una alarma de Presión Baja Crítica (Severidad: CRÍTICA).
    2. Parar automáticamente todas las bombas de impulsión asociadas a esa tubería (ej., Bomba\_Imp\_X, Bomba\_Imp\_Z).
  + **Condición de Activación:** Presion\_Tubería\_Salida\_Y < 1.2 Bar durante 15 segundos
  + **Acción:** Alarma.Generar('Presión Baja Crítica', Tubería\_Salida\_Y), Bomba\_Imp\_X.Parar(), Bomba\_Imp\_Z.Parar().

**Regla RB-003 (Manejo de Caudal Máximo):**

* + **Descripción:** Si el caudal total de entrada a la ETAP Caudal\_ETAP\_Entrada excede 250 m³/h durante más de 5 minutos, el sistema DEBE generar una alarma de Sobrecarga de Planta (Severidad: ALTA) y sugerir la reducción del caudal de captación.
  + **Condición de Activación:** Caudal\_ETAP\_Entrada > 250 m³/h durante 5 minutos
  + **Acción:** Alarma.Generar('Sobrecarga de Planta', Caudal\_ETAP\_Entrada)

**Regla RB-004 (Lógica de Alternancia de Bombas):**

* + **Descripción:** En estaciones con bombas gemelas (ej., Bomba\_Principal\_1, Bomba\_Principal\_2), el sistema DEBE gestionar su operación de forma alternada cada 24 horas de funcionamiento acumulado para equilibrar el desgaste.
  + **Acción:** Si Bomba\_Principal\_1.Tiempo\_Funcionamiento\_Acumulado >= 24h, entonces Bomba\_Principal\_1.Parar() y Bomba\_Principal\_2.Iniciar(). (Asumiendo que solo una bomba opera a la vez).
  + **Excepción:** Si una de las bombas entra en estado Fallida, la otra DEBE intentar iniciar inmediatamente si no está funcionando.

# CASOS DE USO

Para asegurar una implementación robusta y completa de la plataforma SCADA23, se procede a desglosar los requisitos funcionales a un nivel de detalle que permita una comprensión clara de las interacciones del sistema y sus comportamientos esperados. Esto se logra a través de la definición de **Casos de Uso/Escenarios Detallados**, que especifican las funcionalidades principales del sistema y sus interacciones con los actores.

## Diagrama de Casos de Uso (UML)

*NOTA: Se incluiría un diagrama UML de casos de uso que visualice las relaciones entre los actores y los casos de uso principales. Para este ejercicio de texto, se omitirá el diagrama, pero es crucial en el documento final.*

**Ejemplo de actores comunes en SCADA23:**

* **Operador:** Usuario principal del sistema que monitorea el proceso y realiza acciones de control.
* **Administrador:** Usuario con privilegios para configurar el sistema, gestionar usuarios y dispositivos.
* **Ingeniero de Mantenimiento:** Usuario que accede a datos históricos y diagnósticos.
* **Sistema Externo / PLC:** Representa la interacción del SCADA con otros sistemas o los dispositivos de campo.

## Formato de Casos de Uso

Cada caso de uso se presentará siguiendo el siguiente formato para garantizar la exhaustividad:

* **Identificador del Caso de Uso (UC-XXX):** Código único para el caso de uso.
* **Nombre del Caso de Uso:** Descripción concisa de la función.
* **Actor(es) Principal(es):** Entidad o usuario que inicia el caso de uso.
* **Actores Secundarios:** Otras entidades que interactúan con el caso de uso.
* **Descripción:** Breve explicación del propósito del caso de uso.
* **Precondiciones:** Condiciones que deben cumplirse antes de que el caso de uso pueda iniciarse.
* **Flujo Principal:** Secuencia de pasos que describen el comportamiento normal del sistema para lograr el objetivo del caso de uso.
* **Flujos Alternativos:** Secuencias de pasos que describen comportamientos diferentes al flujo principal (ej. entradas erróneas, condiciones excepcionales que no son errores críticos).
* **Manejo de Errores:** Descripción de cómo el sistema responde a situaciones de error inesperadas, validaciones fallidas o fallos del sistema.
* **Postcondiciones:** Estado del sistema después de que el caso de uso se ha completado con éxito.
* A continuación, se listan los casos de uso iniciales basados en la "Descripción Funcional del Proceso (FDS)" del sistema de agua potable. Esta sección se expandirá con el desarrollo de la ingeniería de software:

## Categoría: Gestión de Bombas

* **UC1: Arrancar Bomba Manualmente**
  + **Actor Principal:** Operador HMI.
  + **Descripción:** El operador inicia manualmente el funcionamiento de una bomba específica a través de la interfaz HMI.
  + **Precondiciones:** Bomba en estado 'Parada', alimentación eléctrica disponible, válvula de descarga asociada abierta.
  + **Flujo Principal:**
    1. El Operador selecciona la bomba en la interfaz HMI.
    2. El Operador presiona el botón "Arrancar" para la bomba seleccionada.
    3. El sistema envía la señal de arranque a la bomba.
    4. El sistema verifica el estado de la bomba.
    5. Si la bomba arranca correctamente, su estado en la HMI se actualiza a 'Funcionando'.
    6. El sistema registra el evento de arranque.
  + **Flujos Alternativos:**
    1. **FA1.1: Válvula de descarga cerrada:** El sistema no permite el arranque y muestra un mensaje de advertencia al operador.
    2. **FA1.2: Fallo de arranque de la bomba:** Si la bomba no reporta el estado 'Funcionando' en un tiempo definido (ej. 5 segundos), el sistema intenta un reintento (máx. 2) y si falla, notifica al operador y registra la alarma.
  + **Manejo de Errores:** En caso de fallos de comunicación con la bomba, se mostrará una alarma y se registrará el error en el log del sistema.
  + **Postcondiciones:** La bomba está funcionando y su estado se refleja correctamente en la HMI y en la base de datos.
* **UC2: Parar Bomba Manualmente**
  + **Actor Principal:** Operador HMI.
  + **Descripción:** El operador detiene manualmente el funcionamiento de una bomba específica a través de la interfaz HMI.
  + **Precondiciones:** Bomba en estado 'Funcionando'.
  + **Flujo Principal:**
    1. El Operador selecciona la bomba en la interfaz HMI.
    2. El Operador presiona el botón "Parar" para la bomba seleccionada.
    3. El sistema envía la señal de parada a la bomba.
    4. El sistema verifica el estado de la bomba.
    5. Si la bomba se detiene correctamente, su estado en la HMI se actualiza a 'Parada'.
    6. El sistema registra el evento de parada.
  + **Flujos Alternativos:**
    1. **FA2.1: Fallo de parada de la bomba:** Si la bomba no reporta el estado 'Parada' en un tiempo definido (ej. 5 segundos), el sistema notifica al operador y registra la alarma.
  + **Manejo de Errores:** En caso de fallos de comunicación con la bomba, se mostrará una alarma y se registrará el error en el log del sistema.
  + **Postcondiciones:** La bomba está parada y su estado se refleja correctamente en la HMI y en la base de datos.
* **UC3: Arrancar Bomba por Nivel de Tanque Bajo (Automático)**
  + **Actor Principal:** Sistema (Automático).
  + **Actor Secundario:** Sensor de Nivel del Tanque.
  + **Descripción:** El sistema inicia automáticamente el funcionamiento de una bomba de llenado cuando el nivel del agua en el tanque principal desciende por debajo de un umbral predefinido.
  + **Precondiciones:** Modo de control automático activado, bomba en estado 'Parada', nivel del tanque por debajo del umbral mínimo configurado, válvula de descarga asociada abierta.
  + **Flujo Principal:**
    1. El Sensor de Nivel del Tanque reporta un valor por debajo del umbral mínimo (Nivel\_Min\_Tanque).
    2. El sistema detecta la condición de nivel bajo.
    3. El sistema verifica las precondiciones (ej. modo automático activado).
    4. El sistema envía la señal de arranque a la bomba de llenado.
    5. El sistema verifica el estado de la bomba.
    6. Si la bomba arranca correctamente, su estado en la HMI se actualiza a 'Funcionando (Auto)'.
    7. El sistema registra el evento de arranque automático.
  + **Flujos Alternativos:**
    1. **FA3.1: Fallo de arranque de la bomba:** Si la bomba no reporta el estado 'Funcionando' en un tiempo definido, el sistema intenta un reintento (máx. 2), si falla notifica una alarma y registra el error.
    2. **FA3.2: Nivel ya recuperado:** Si el nivel sube por encima del umbral mínimo antes de que la bomba arranque, el sistema cancela el arranque.
  + **Manejo de Errores:** Fallos de comunicación con el sensor o la bomba generarán alarmas y entradas en el log.
  + **Postcondiciones:** La bomba de llenado está funcionando y el nivel del tanque comienza a subir.

## Categoría: Visualización y Monitorización

* **UC4: Visualizar Estado del Sistema**
  + **Actor Principal:** Operador HMI.
  + **Descripción:** El operador visualiza el estado actual de todos los componentes del sistema (bombas, válvulas, niveles, alarmas) a través de la interfaz HMI.
  + **Precondiciones:** Sesión de operador activa.
  + **Flujo Principal:**
    1. El Operador accede a la interfaz HMI.
    2. El sistema carga y muestra el diagrama de proceso con los estados en tiempo real de los equipos.
    3. El sistema actualiza periódicamente (ej. cada segundo) los valores de los sensores y el estado de los actuadores.
  + **Postcondiciones:** El operador tiene una vista actualizada del sistema.

## Categoría: Gestión de Alarmas

* **UC5: Generar Alarma por Nivel de Tanque Alto**
  + **Actor Principal:** Sistema (Automático).
  + **Actor Secundario:** Sensor de Nivel del Tanque.
  + **Descripción:** El sistema genera una alarma cuando el nivel del agua en el tanque principal supera un umbral máximo predefinido.
  + **Precondiciones:** Nivel del tanque por encima del umbral máximo configurado.
  + **Flujo Principal:**
    1. El Sensor de Nivel del Tanque reporta un valor por encima del umbral máximo (Nivel\_Max\_Tanque).
    2. El sistema detecta la condición de nivel alto.
    3. El sistema genera una alarma de "Nivel de Tanque Alto".
    4. La alarma se muestra en la HMI (intermitente, color rojo).
    5. La alarma se registra en el historial de eventos.
  + **Flujos Alternativos:**
    1. **FA5.1: Nivel vuelve a la normalidad antes de la acción:** Si el nivel desciende por debajo del umbral antes de la confirmación, la alarma se clasifica como 'desactivada' pero no se borra.
  + **Manejo de Errores:** Fallo del sensor de nivel resultará en una alarma de 'Fallo de sensor' y se registrará.
  + **Postcondiciones:** Alarma de nivel alto activa y visible en la HMI, registrada en el sistema.

## Categoría: Gestión de Usuarios y Seguridad

* **UC6: Autenticar Usuario**
  + **Actor Principal:** Usuario (Operador, Administrador).
  + **Descripción:** Un usuario intenta acceder al sistema SCADA23 mediante sus credenciales.
  + **Precondiciones:** El sistema SCADA23 está en funcionamiento.
  + **Flujo Principal:**
    1. El Usuario accede a la interfaz de inicio de sesión.
    2. El Usuario introduce su nombre de usuario y contraseña.
    3. El sistema valida las credenciales.
    4. Si las credenciales son válidas, el sistema concede acceso al Usuario con su nivel de permisos asociado.
    5. El sistema registra el evento de inicio de sesión.
  + **Flujos Alternativos:**
    1. **FA6.1: Credenciales inválidas:** El sistema muestra un mensaje de error ("Usuario o contraseña incorrectos") y no concede acceso.
    2. **FA6.2: Bloqueo por intentos fallidos:** Después de N intentos fallidos, el sistema bloquea temporalmente la cuenta.
  + **Manejo de Errores:** Errores de conexión con la base de datos de usuarios resultarán en un mensaje de error genérico y un registro de error.
  + **Postcondiciones:** El usuario está autenticado y tiene acceso al sistema, o el intento de inicio de sesión ha fallado y se ha registrado.

## Otros casos de uso

Este apartado describe las interacciones entre los actores y el sistema SCADA23 para lograr objetivos específicos. Los Casos de Uso (CU) son una herramienta fundamental para comprender los requisitos del usuario desde una perspectiva de alto nivel, sirviendo de puente entre las necesidades operacionales y los requisitos funcionales detallados. Cada caso de uso se presenta con un identificador único, una descripción general, los actores involucrados, precondiciones, el flujo básico de eventos, flujos alternativos, flujos de excepción y postcondiciones.

**Formato de Caso de Uso (CU-XXX):**

* **ID:** CU-YYY (Ej. CU-001)
* **Nombre:** Título conciso del caso de uso.
* **Descripción:** Un resumen del objetivo del caso de uso.
* **Actor(es) Primario:** El actor principal que inicia el caso de uso y se beneficia de él.
* **Actores Secundarios:** Otros actores que interactúan con el caso de uso.
* **Precondiciones:** Condiciones que deben ser verdaderas antes de que el caso de uso pueda comenzar.
* **Flujo Básico de Eventos:** Los pasos normales y esperados de la interacción entre el actor y el sistema.
* **Flujos Alternativos:** Variaciones del flujo básico que también logran el objetivo principal, pero de una manera diferente.
* **Flujos de Excepción:** Pasos o condiciones que resultan en un fallo o un resultado no deseado del caso de uso.
* **Postcondiciones:** El estado del sistema después de que el caso de uso se ha completado exitosamente.
* **Referencias a RFs:** Lista de Requisitos Funcionales (RF-XXX) relacionados que soportan este caso de uso.

**Caso de Uso 1: Monitorizar Estado de Planta**

* **ID:** CU-001
* **Nombre:** Monitorear Estado Operacional de la Planta
* **Descripción:** El Operador visualiza en tiempo real el estado general de las instalaciones (ETAP, EBAP, red de distribución) y los valores actuales de los dispositivos y sensores clave.
* **Actor(es) Primario:** Operador
* **Actores Secundarios:** N/A
* **Precondiciones:** El Operador DEBE estar autenticado en el sistema. El sistema SCADA DEBE estar recibiendo datos de los dispositivos de campo.
* **Flujo Básico de Eventos:**
  1. El Operador accede al sistema SCADA23.
  2. El sistema presenta la pantalla de inicio con un resumen del estado de la planta.
  3. El Operador selecciona un sinóptico específico (ej., "ETAP Principal", "Red de Distribución Sur") desde el menú de navegación.
  4. El sistema carga y muestra el sinóptico seleccionado, actualizando dinámicamente los valores de los tags, los estados de los equipos (ej., bombas, válvulas) y las alarmas activas.
  5. El Operador puede navegar entre diferentes sinópticos o vistas de la planta.
* **Flujos Alternativos:**
  1. **FA-1.1:** El Operador puede usar la barra de búsqueda para encontrar un dispositivo o vista específica.
  2. **FA-1.2:** El Operador puede personalizar la disposición de ciertos widgets o paneles si se le otorgan permisos para ello.
* **Flujos de Excepción:**
  1. **FE-1.1: Fallo de Conexión:** Si el sistema pierde la conexión con un dispositivo o un servidor de datos, los valores asociados DEBEN mostrarse como "Sin Datos" o "Error de Conexión", y DEBE generarse una alarma de comunicación.
  2. **FE-1.2: Sinóptico no Carga:** Si un sinóptico no puede ser cargado (ej., por error de configuración), el sistema DEBE mostrar un mensaje de error claro al Operador y registrar el incidente.
* **Postcondiciones:** El Operador tiene una comprensión actualizada del estado operativo de la planta.
* **Referencias a RFs:** RF-001, RF-007, RNF-001, RNF-002, RCS-001, RCS-002.

**Caso de Uso 2: Ejecutar Comando de Control Remoto**

* **ID:** CU-002
* **Nombre:** Ejecutar un Comando de Control Remoto sobre un Dispositivo
* **Descripción:** El Operador inicia o detiene una bomba, o abre/cierra una válvula, u otro actuador de forma remota a través de la interfaz SCADA.
* **Actor(es) Primario:** Operador
* **Actores Secundarios:** Servidor de Control, Dispositivo de Campo.
* **Precondiciones:** El Operador DEBE estar autenticado y tener permisos de control para el dispositivo específico. El dispositivo DEBE estar online y en un estado que permita la ejecución del comando.
* **Flujo Básico de Eventos:**
  1. El Operador navega al sinóptico o vista donde se encuentra el dispositivo a controlar (ej., Bomba P-1).
  2. El Operador selecciona el dispositivo (ej., hace clic en el ícono de la Bomba P-1).
  3. El sistema muestra una ventana de control con las acciones disponibles para el dispositivo (ej., "Iniciar", "Parar", "Alternar").
  4. El Operador selecciona la acción deseada (ej., "Iniciar").
  5. El sistema muestra una ventana de confirmación pidiendo al Operador que confirme la acción.
  6. El Operador confirma la acción.
  7. El sistema envía el comando de control al dispositivo de campo.
  8. El sistema actualiza el estado del dispositivo en la interfaz de usuario (ej., de "Parada" a "Arrancando").
  9. El dispositivo de campo ejecuta el comando y su estado real se refleja en el SCADA.
  10. El sistema registra el comando ejecutado en el log de eventos.
* **Flujos Alternativos:**
  1. **FA-2.1: Comando Modulante:** Si el dispositivo es modulante (ej., válvula con posición), el Operador introduce un valor numérico (ej., 0-100%) en la ventana de control antes de confirmar.
* **Flujos de Excepción:**
  1. **FE-2.1: Permisos Insuficientes:** Si el Operador no tiene permisos, el sistema DEBE deshabilitar la opción de control y/o mostrar un mensaje de "Acceso Denegado".
  2. **FE-2.2: Dispositivo Fuera de Servicio/No Disponible:** Si el dispositivo no está en línea o en un estado de fallo, el sistema DEBE indicar que el comando no puede ser enviado y mostrar un mensaje de error.
  3. **FE-2.3: Error en el Comando:** Si el dispositivo de campo no ejecuta el comando (ej., por bloqueo mecánico, fallo de comunicación), el sistema DEBE mostrar un mensaje de error, generar una alarma de "Fallo de Comando" y revertir el estado del dispositivo en la UI si es necesario.
  4. **FE-2.4: Cancelación de Operador:** El Operador cancela la operación en el paso de confirmación. El sistema DEBE volver al estado anterior sin cambios.
* **Postcondiciones:** El comando de control ha sido enviado al dispositivo y su ejecución, o el fallo de ejecución, ha sido registrado y reflejado en el sistema.
* **Referencias a RFs:** RF-003, RF-004, RNF-002, RCS-002, RCS-006.

**Caso de Uso 3: Gestionar Alarmas del Sistema**

* **ID:** CU-003
* **Nombre:** Gestionar Alarmas del Sistema
* **Descripción:** El Operador visualiza, reconoce y gestiona las alarmas activas y el historial de alarmas generadas por el sistema.
* **Actor(es) Primario:** Operador
* **Actores Secundarios:** Motor de Alarmas.
* **Precondiciones:** El Operador DEBE estar autenticado. El sistema DEBE haber generado alarmas.
* **Flujo Básico de Eventos:**
  1. El Operador accede a la pantalla de "Gestión de Alarmas" (o el widget de alarmas en el sinóptico principal).
  2. El sistema muestra una lista de alarmas activas, ordenadas por severidad y timestamp, con indicadores visuales de estado (activo/reconocido).
  3. El Operador selecciona una o varias alarmas activas.
  4. El Operador hace clic en el botón "Reconocer" o "Silenciar".
  5. El sistema actualiza el estado de la alarma a "Reconocida" o "Silenciada" y registra la acción con el ID del operador y timestamp.
  6. El Operador puede filtrar o buscar alarmas por diferentes criterios (severidad, dispositivo, rango de tiempo).
* **Flujos Alternativos:**
  1. **FA-3.1: Acceder a Historial:** El Operador puede seleccionar ver el "Historial de Alarmas" para visualizar alarmas pasadas que ya han vuelto a su estado normal.
  2. **FA-3.2: Exportar Alarmas:** El Operador puede exportar la lista de alarmas filtradas a un archivo (ej., CSV).
* **Flujos de Excepción:**
  1. **FE-3.1: Permisos Insuficientes:** Si el Operador intenta reconocer una alarma sin los permisos adecuados, el sistema DEBE denegar la acción y mostrar un mensaje.
  2. **FE-3.2: Alarma Ya Reconocida/Inactiva:** Si el Operador intenta reconocer una alarma que ya está reconocida o que ya no está activa, el sistema DEBE informar al Operador.
* **Postcondiciones:** Las alarmas activas han sido gestionadas (reconocidas/silenciadas) y su estado se ha registrado. El Operador tiene acceso al historial de alarmas.
* **Referencias a RFs:** RF-005, RF-006, RNF-003, RNF-008, RCS-002, RCS-006.

**Caso de Uso 4: Visualizar Datos Históricos y Tendencias**

* **ID:** CU-004
* **Nombre:** Visualizar Datos Históricos y Tendencias
* **Descripción:** El Operador o Ingeniero de Mantenimiento consulta el rendimiento pasado de los equipos o las variables del proceso a través de gráficos de tendencias.
* **Actor(es) Primario:** Operador, Ingeniero de Mantenimiento.
* **Actores Secundarios:** Base de Datos de Series Temporales.
* **Precondiciones:** El actor DEBE estar autenticado. Datos históricos DEBEN estar disponibles para los tags seleccionados.
* **Flujo Básico de Eventos:**
  1. El actor accede a la sección de "Tendencias Históricas".
  2. El sistema presenta una interfaz para seleccionar tags y un rango de tiempo.
  3. El actor selecciona uno o más tags (ej., "Nivel Depósito Principal", "Caudal Bomba P-1", "Presión Salida Tubería B").
  4. El actor selecciona un rango de tiempo (ej., "Últimas 24h", "Semana Pasada", "Personalizado").
  5. El sistema recupera los datos históricos y los muestra en un gráfico de tendencias interactivo.
  6. El actor puede aplicar zoom, desplazarse por el tiempo o cambiar la resolución del gráfico.
* **Flujos Alternativos:**
  1. **FA-4.1: Comparación de Períodos:** El actor puede seleccionar dos rangos de tiempo diferentes para comparar tendencias en el mismo gráfico.
  2. **FA-4.2: Exportar Datos:** El actor puede exportar los datos mostrados en el gráfico a un formato de hoja de cálculo (ej., CSV).
* **Flujos de Excepción:**
  1. **FE-4.1: Datos no Disponibles:** Si no hay datos históricos para el rango de tiempo o los tags seleccionados, el sistema DEBE mostrar un mensaje indicando que no hay datos.
  2. **FE-4.2: Error de Carga:** Si la carga de datos es excesivamente lenta o falla, el sistema DEBE notificar al actor y registrar el error.
* **Postcondiciones:** El actor tiene acceso a la información histórica de los tags seleccionados, permitiendo el análisis de tendencias.
* **Referencias a RFs:** RF-002, RF-008, RNF-001, RNF-002.

**Caso de Uso 5: Gestionar Usuarios y Roles (Administrador)**

* **ID:** CU-005
* **Nombre:** Gestionar Usuarios y Roles
* **Descripción:** El Administrador crea, modifica, elimina y gestiona los roles de los usuarios del sistema SCADA23.
* **Actor(es) Primario:** Administrador
* **Actores Secundarios:** N/A
* **Precondiciones:** El Administrador DEBE estar autenticado y tener el rol de "Administrador de Seguridad".
* **Flujo Básico de Eventos:**
  1. El Administrador accede a la sección de "Gestión de Usuarios".
  2. El sistema muestra una lista de usuarios existentes.
  3. **Para Crear Usuario:**
     + El Administrador hace clic en "Crear Nuevo Usuario".
     + Introduce los datos del nuevo usuario (nombre de usuario, contraseña inicial, email) y le asigna uno o más roles (ej., Operador, Visualizador).
     + El sistema valida los datos y crea el nuevo usuario, registrando la acción.
  4. **Para Modificar Usuario:**
     + El Administrador selecciona un usuario existente.
     + Modifica sus datos (ej., roles asignados, fuerza a cambio de contraseña).
     + El sistema actualiza el usuario, registrando la acción.
  5. **Para Eliminar Usuario:**
     + El Administrador selecciona un usuario existente.
     + Confirma la eliminación.
     + El sistema elimina el usuario y sus asociaciones, registrando la acción.
* **Flujos Alternativos:**
  1. **FA-5.1: Resetear Contraseña:** El Administrador puede forzar el reseteo de la contraseña de un usuario.
  2. **FA-5.2: Gestionar Roles:** El Administrador puede acceder a una pantalla de gestión de roles para definir nuevos roles y sus permisos asociados, aunque se priorizará un conjunto fijo de roles predefinidos en la versión inicial.
* **Flujos de Excepción:**
  1. **FE-5.1: Nombre de Usuario Existente:** Si el Administrador intenta crear un usuario con un nombre ya existente, el sistema DEBE mostrar un error.
  2. **FE-5.2: Datos Inválidos:** Si los datos introducidos no cumplen con las políticas de contraseña o formato, el sistema DEBE mostrar errores de validación.
  3. **FE-5.3: Eliminación de Último Administrador:** El sistema DEBE prevenir la eliminación o degradación del último usuario con rol de "Administrador de Seguridad" para evitar un bloqueo total.
* **Postcondiciones:** La base de datos de usuarios se ha actualizado de acuerdo con las acciones del Administrador, y las acciones han sido registradas en la auditoría.
* **Referencias a RFs:** RCS-001, RCS-002, RCS-006, RNF-009.

# DISEÑO DE LA ARQUITECTURA DE LA INFRAESTRUCTURA

## Arquitectura general

Este capítulo detalla la arquitectura de la Plataforma SCADA23, describiendo sus componentes principales, cómo interactúan entre sí y las tecnologías subyacentes. Se presentarán dos arquitecturas distintas, reflejando la evolución del proyecto a través de sus fases de desarrollo: la arquitectura de la Fase 1 (TRL4), enfocada en la demostración de concepto, y la arquitectura de la Fase 2 (TRL7),

**Principios Arquitectónicos**

Independientemente de la fase, la concepción arquitectónica de SCADA23 se rige por los siguientes principios:

* **Modularidad:** División del sistema en componentes lógicos y autocontenidos para facilitar el desarrollo, mantenimiento y escalabilidad.
* **Escalabilidad:** Capacidad de la arquitectura para crecer y adaptarse a un número creciente de dispositivos, puntos de datos, usuarios y carga de trabajo.
* **Flexibilidad/Extensibilidad:** Facilidad para incorporar nuevas funcionalidades, protocolos de comunicación o tecnologías en el futuro.
* **Fiabilidad y Resiliencia:** Diseño que minimiza el impacto de fallos individuales y asegura la continuidad del servicio.
* **Ciberseguridad desde el Diseño:** Integración de medidas de seguridad en cada capa y componente de la arquitectura.
* **Orientación a Datos:** Procesamiento y almacenamiento eficiente de grandes volúmenes de datos en tiempo real e históricos.
* **Desacoplamiento:** Reducción de las dependencias entre componentes para permitir su desarrollo y despliegue independiente.

## Arquitectura de la Fase 1 (TRL4)

La arquitectura de la Fase 1 está diseñada para ser minimalista y funcional, centrada en validar la capacidad de la plataforma para adquirir y controlar datos utilizando el protocolo Modbus TCP en un entorno controlado.

**Diagrama de la arquitectura**

Interfaz de usuario gráfica, Diagrama

El contenido generado por IA puede ser incorrecto.

Fuente: <https://websim.com/@stieber71/diagrama-de-arquitectura-modbus-tcp/1>

**Componentes Lógicos (Fase 1)**

Los componentes lógicos principales de la Fase 1 son:

* **Cliente de Campo (Simulado/Básico):** Representa los dispositivos industriales o un pequeño conjunto de scripts de línea de comandos que actúan como un cliente Modbus TCP.
* **Módulo de Comunicación Modbus TCP:** La parte del SCADA que maneja la conexión y el intercambio de datos con los dispositivos Modbus TCP.
* **Lógica de Procesamiento y Control Básica:** El núcleo de la aplicación SCADA que recibe datos, los procesa y envía comandos.
* **Interfaz de Usuario (Tkinter):** La capa de presentación que muestra los datos y permite la interacción manual.

**Componentes de Software (Fase 1)**

* **Scripts Python en Línea de Comandos (Cliente):**
  + Implementación de clientes Modbus TCP para enviar y recibir datos.
  + Ejecución manual para simular lectura de sensores o activación de actuadores.
  + Posiblemente utilizando la librería pymodbus o similar.
* **Programa SCADA (Python + Tkinter):**
  + **Core Lógico:** Un único script Python que integra la lógica de comunicación Modbus TCP (como cliente) y el procesamiento de datos.
  + **Interfaz Gráfica (Tkinter):** Módulo de Tkinter para construir la interfaz de usuario. Controles básicos (botones, etiquetas de texto, campos de entrada) para visualizar valores y enviar comandos.
  + **Almacenamiento:** No hay base de datos dedicada. Los datos son efímeros y se mantienen en memoria para su visualización inmediata. La configuración de los puntos de datos es estática, posiblemente codificada directamente en el script o en un archivo de configuración .ini simple.
  + **Comunicación:** Utiliza una librería Modbus TCP de Python para interactuar con los dispositivos.

**Consideraciones de Hardware (Fase 1)**

* Servidor SCADA: Un ordenador personal estándar (portátil o de escritorio) con sistema operativo Windows, Linux o macOS compatible con Python y Tkinter.
* Dispositivos de Campo: Pueden ser PLCs reales con Modbus TCP o simuladores de Modbus TCP ejecutándose en otro PC o incluso en el mismo para una demostración sencilla.
* Red: Una red LAN básica (Ethernet) para la comunicación Modbus TCP.

## Arquitectura de la Fase 2 (TRL7)

La arquitectura de la Fase 2 es una evolución robusta y escalable de la Fase 1, diseñada para soportar entornos operacionales complejos, grandes volúmenes de datos, múltiples protocolos y una alta disponibilidad. Se basa en una arquitectura de microservicios o servicios desacoplados.

**Diagrama de la arquitectura**

Diagrama

El contenido generado por IA puede ser incorrecto.

Fuente: <https://websim.com/@stieber71/diagrama-de-arquitectura-modbus-tcp>

**Componentes Lógicos (Fase 2)**

* **Capa de Adquisición de Datos (Data Acquisition Layer):** Responsable de la comunicación con los dispositivos de campo y la recolección de datos.
* **Capa de Procesamiento y Lógica de Negocio (Business Logic Layer):** Donde se gestionan las alarmas, los cálculos, la lógica de control, la autenticación y autorización.
* **Capa de Persistencia de Datos (Data Persistence Layer):** Bases de datos optimizadas para diferentes tipos de datos (series temporales y relacionales).
* **Capa de API (API Layer):** Proporciona interfaces RESTful para la comunicación con los clientes y otros sistemas.
* **Capa de Presentación (Presentation Layer / HMI):** La interfaz de usuario web para la supervisión y el control.
* **Capa de Seguridad (Security Layer):** Integrada en todas las capas para garantizar la protección de datos y el control.

**Componentes de Software (Fase 2)**

* **Módulos de Adquisición de Datos (Data Collectors / Drivers):**
  + Aplicaciones o servicios desarrollados con **FastAPI** para la recolección de datos en tiempo real de alta eficiencia.
  + Librerías Python específicas para **Modbus TCP**, **PROFINET** y **OPC-UA** (ej., pymodbus, librerías de terceros para PROFINET, asyncua para OPC-UA).
  + Diseñados para ser resilientes a fallos de comunicación (reconexiones automáticas, buffering local).
* **Backend / Lógica de Negocio (Core SCADA Server):**
  + Desarrollado principalmente con **Django**. Manejará:
    - **Gestión de Modelos de Datos:** Dispositivos, tags, alarmas, usuarios, roles, configuraciones (ORM de Django).
    - **Lógica de Alarmas y Eventos:** Procesamiento de umbrales, envío de notificaciones.
    - **Autenticación y Autorización:** Integrado con el sistema de usuarios de Django.
    - **Servicios RESTful:** Expuestos a través de **Django REST Framework (DRF)** para la interfaz de usuario y otras integraciones.
    - **Cálculos y Agregación de Datos:** Procesamiento de datos para KPIs, variables derivadas.
  + Posibles tareas asíncronas con **Celery** si se requiere procesamiento en segundo plano (ej. generación de informes pesados, cálculos complejos).
* **Bases de Datos:**
  + **MySQL:**
    - Para datos relacionales estructurados: Configuración de dispositivos, tags, alarmas, usuarios, roles, permisos, registros de eventos, historial de auditoría, metadatos.
    - Proporciona integridad referencial y transacciones ACID.
  + **InfluxDB:**
    - Para datos de series temporales de alta velocidad: Medidas de sensores (temperatura, presión, caudal, nivel), estados de actuadores (ON/OFF), valores de control.
    - Optimizado para la ingesta rápida de datos de series temporales y consultas eficientes por rangos de tiempo.
* **Frontend / Interfaz de Usuario (HMI Web):**
  + Aplicación web SPA (Single Page Application) o renderizada por el servidor con Django Templates/HTMX, pero con un enfoque en la interactividad y la visualización rica.
  + Se comunicará exclusivamente con el backend a través de las **APIs RESTful** proporcionadas por DRF.
  + Componentes de visualización para sinópticos, gráficos de tendencias (ej., Chart.js, Plotly.js), tablas de alarmas, gestión de usuarios.
* **Servidor Web (Gateway/Proxy Inverso):**
  + **Nginx** o **Apache:** Para servir la aplicación web estática, actuar como proxy inverso para las APIs de Django/FastAPI, gestionar el balanceo de carga y aplicar políticas de seguridad (HTTPS/TLS).
* **Contenedorización (Opcional pero muy recomendado):**
  + **Docker:** Para empaquetar cada componente (FastAPI collector, Django backend, MySQL, InfluxDB, Nginx) en contenedores aislados. Esto facilita el despliegue, la gestión de dependencias y la escalabilidad.
  + **Docker Compose:** Para orquestar el despliegue de todos los servicios en un único host.
  + **Kubernetes (para despliegues muy grandes):** Para orquestación de contenedores a escala, alta disponibilidad y autoescalado en entornos de producción distribuidos.

**Consideraciones de Hardware (Fase 2)**

* **Servidores SCADA:** Se recomienda el despliegue en servidores dedicados o máquinas virtuales potentes, con recursos suficientes de CPU, RAM y almacenamiento SSD.
  + Puede ser un único servidor potente o varios servidores distribuidos, especialmente si se usa Docker/Kubernetes.
* **Almacenamiento:** Alta capacidad y rendimiento (SSD NVMe) para las bases de datos, especialmente InfluxDB, dado el volumen de datos de series temporales.
* **Red:** Infraestructura de red robusta y segura, con segmentación de red (OT/IT) y firewalls para proteger los componentes críticos. Se requerirá conectividad de red confiable (Ethernet, Fibra, 4G/5G con VPNs) para los dispositivos remotos.
* **Equipos de Campo:** PLCs, RTUs, sensores y actuadores compatibles con Modbus TCP, PROFINET y/o OPC-UA.

**Diagrama del modelo C4**

La arquitectura final de SCADA23 se basará en un modelo distribuido por capas, diseñado para desacoplar las responsabilidades y facilitar la escalabilidad horizontal. Se distinguen principalmente las siguientes capas y sus respectivos componentes:

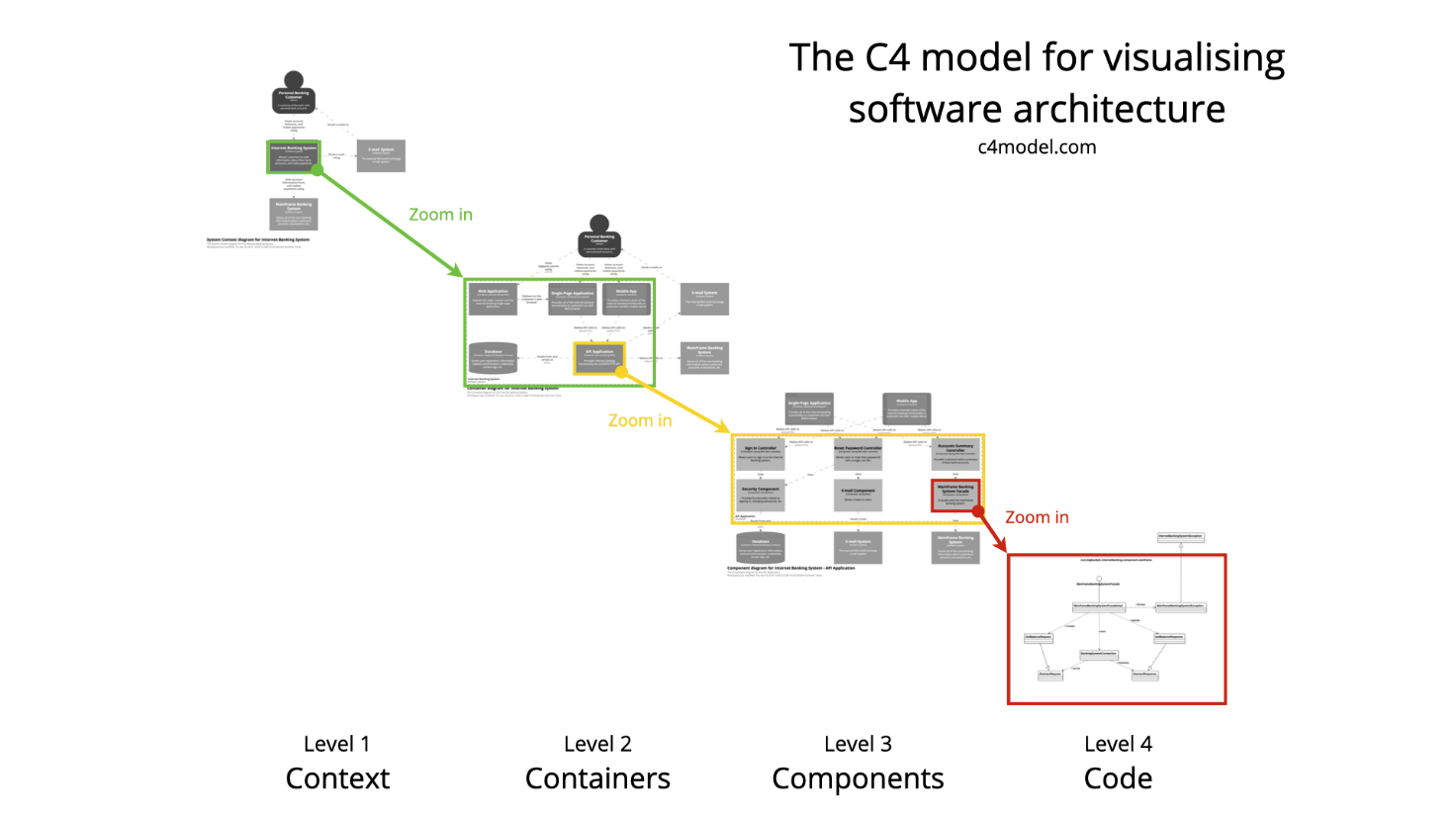
* **Capa de Presentación (Frontend):** Interfaz de usuario para la interacción con los operadores y administradores.
* **Capa de Lógica de Negocio (Backend/Servicios):** Contiene la lógica central del sistema, procesamiento de datos, gestión de comandos y alarmas.
* **Capa de Adquisición y Conectividad (OT/IT Gateway):** Responsable de la comunicación con los dispositivos de campo y la normalización de datos.
* **Capa de Persistencia de Datos:** Almacenamiento de datos en tiempo real, históricos y configuraciones.
* **Capa de Infraestructura (Despliegue):** Plataforma subyacente para el despliegue y la orquestación de los componentes.

El modelo C4 consiste en un conjunto jerárquico de diagramas de arquitectura de software para contexto, contenedores, componentes y código. La jerarquía de los diagramas C4 proporciona diferentes niveles de abstracción, cada uno de los cuales es relevante para una audiencia diferente. El diagrama de alto nivel de la arquitectura muestra un diagrama de componentes del Modelo C4, que muestra las capas y las interacciones principales:

* C4 Nivel 1: Context/Contexto
* C4 Nivel 2: Container/Contenedor
* C4 Nivel 3: Component/Componente
* C4 Nivel 4: Code/Código

El diagrama de componentes en el modelo C4 de arquitectura de software es una representación visual que muestra cómo un contenedor (como una aplicación o una base de datos) está compuesto por componentes individuales y cómo estos componentes interactúan entre sí. Es como una ampliación de un diagrama de contenedores, mostrando los detalles internos de cada uno de ellos. El Modelo C4 es un formato de documentación creado por el ingeniero Simon Brown entre 2006 y 2011, y basado en los modelos 4+1 y UML y descrito en su libro ““Software Architecture for Developers”.

Fuente: <https://github.com/simonbrowndotje/c4model/>



Fuente: <https://c4model.com/>

**Diagrama de red**

Diagrama

El contenido generado por IA puede ser incorrecto.

Fuente: <https://rodrigocantera.com/laboratorio-virtual-de-ciberseguridad-industrial-parte-1-instalacion-de-router-pfsense/>

**Descripción de la arquitectura**

La conexión de OpenPLC, Factory I/O y ScadaBR a la plataforma SCADA23 se realizara todos en la misma subred, permitiendo acceso mediante OpenVPN.

## Componentes generales

**Tecnologías fundamentales por capa**

|  |  |  |
| --- | --- | --- |
| **Capa / Componente** | **Fase 1 (TRL4)** | **Fase 2 (TRL7)** |
| **Lenguaje de Programación** | Python | Python |
| **Framework Backend** | N/A (scripts directos) | Django, FastAPI, Django REST Framework |
| **Interfaz de Usuario** | Tkinter (Python) | Aplicación Web (HTML5, CSS3, JavaScript - vía Django Templates/Framework Frontend) |
| **Bases de Datos** | N/A (memoria/archivos planos) | MySQL (relacional), InfluxDB (series temporales) |
| **Comunicación Industrial** | Modbus TCP (librería Python) | Modbus TCP, PROFINET, OPC-UA (librerías Python especializadas) |
| **Servidor Web** | N/A (aplicación standalone) | Nginx / Apache (como proxy inverso) |
| **Contenedorización** | N/A | Docker, Docker Compose (opcionalmente Kubernetes) |
| **Gestión de Tareas Asíncronas** | N/A | Celery (opcional) |
| **Sistema Operativo** | Windows / Linux / macOS | Linux (preferiblemente para producción) |

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

El módulo de Agua Potable en SCADA23 se encarga de:

* **Adquisición de Datos:** Lee 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 muestran en tiempo real en el HMI (P&ID) de forma gráfica. Se utilizan indicadores visuales para el estado de las válvulas y el nivel de los tanques.
* **Control:** Envía 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:** Detecta y muestra alarmas cuando los valores de los sensores superen o caigan por debajo de los puntos de ajuste predefinidos.
* **Configuración:** Los rangos de medición, unidades y puntos de ajuste se cargan desde el archivo de configuración del sector, permitiendo una fácil adaptación.

**Historian**

* **Registro de Datos (Historian):** La plataforma tiene capacidad básica de registrar los datos de los sensores a lo largo del tiempo, como medida futura.

**PLCs**

* El sistema está pensado para simulación por software, pero se plantea la compatibilidad específica con modelos específico de Raspberry Pi o Arduino para OpenPLC
* Igualmente, se plantean varios tipos de sensores y actuadores simulados en Factory I/O

**Software Open Source**

* Versiones específicas de OpenPLC Editor y Runtime
* Versión de ScadaBR
* Versión de Factory I/O

**Sistema Operativo**

* Soporte Windows 11 y Linux Debian 12 “Bookworm” LTS para el servidor SCADA y los clientes

**Bibliotecas**

Dependencias de Software Python

## Componentes principales y sus responsabilidades

**Componentes de la Capa de Adquisición y Conectividad**

* **Servicio de Comunicación (Data Collector / Driver Service)**
  + **Descripción:** Componente responsable de establecer y mantener conexiones con los dispositivos de campo (PLCs, RTUs, sensores inteligentes). Actúa como un gateway entre la red de operaciones (OT) y la red de tecnología de la información (IT).
  + **Responsabilidades:**
    - Lectura periódica de tags (puntos de datos) de dispositivos.
    - Escritura de comandos a dispositivos.
    - Manejo de errores de comunicación y reintentos.
    - Normalización inicial de los datos recibidos (ej., conversión de tipos de datos, escalado).
    - Reporte de estado de conexión de dispositivos.
  + **Protocolos Soportados:** OPC UA (Cliente), Modbus/TCP (Maestro), MQTT (Cliente). Se priorizará OPC UA por sus capacidades de seguridad y metadatos.
  + **Tecnologías:** Librerías específicas para cada protocolo (ej., python-opcua, pymodbus).
  + **Patrones de Diseño Aplicados:**
    - **Adapter:** Para adaptar diferentes protocolos de comunicación a una interfaz de datos interna unificada.
    - **Observer:** El Servicio de Comunicación notificará a los servicios de la capa de lógica de negocio (ej., Historian, Motor de Alarmas) sobre las actualizaciones de datos.

**Componentes de la Capa de Lógica de Negocio (Backend)**

* **Servicio Historian (Data Archiver)**
  + **Descripción:** Responsable de recibir los datos normalizados del Servicio de Comunicación y persistirlos eficientemente en la base de datos de series temporales.
  + **Responsabilidades:**
    - Ingesta de datos de tags en tiempo real.
    - Indexación y almacenamiento de datos con alta precisión de timestamp.
    - Gestión de políticas de retención de datos.
    - Proporcionar API para la consulta de datos históricos por parte de otros servicios.
  + **Tecnologías:** **InfluxDB** (Base de Datos de Series Temporales), Python (para el servicio de ingesta).
* **Motor de Alarmas y Eventos (Alarm & Event Engine)**
  + **Descripción:** Procesa los datos en tiempo real para detectar condiciones de alarma y genera eventos del sistema.
  + **Responsabilidades:**
    - Evaluación continua de los valores de los tags contra umbrales y reglas de alarma configuradas.
    - Generación de alarmas con su severidad, descripción, timestamp, y valor asociado.
    - Gestión del ciclo de vida de las alarmas (activa, reconocida, inactiva).
    - Generación de notificaciones (internas del sistema, email, SMS).
    - Registro de todos los eventos del sistema (login, logout, cambios de configuración, comandos).
  + **Tecnologías:** Python, motor de reglas ligero (ej., basado en expresiones JSON o un DSL simple).
  + **Patrones de Diseño Aplicados:**
    - **Observer:** Observa los cambios en los tags y notifica a la interfaz de usuario y otros servicios sobre nuevas alarmas.
    - **State:** Las alarmas en sí mismas pueden tener un objeto de estado (Activa, Reconocida, Inactiva) que controla su comportamiento.
* **Servicio de Control y Lógica de Proceso (Control & Process Logic Service)**
  + **Descripción:** Ejecuta comandos de control recibidos de la UI o de lógicas automáticas y gestiona la lógica de estado de los equipos.
  + **Responsabilidades:**
    - Validación de comandos antes de su envío a los dispositivos.
    - Enrutamiento de comandos al Servicio de Comunicación adecuado.
    - Implementación de reglas de negocio complejas (ej., alternancia de bombas, secuencias de arranque/parada, lógica de protección).
    - Manejo de los estados de los dispositivos (ej., Parada, Arrancando, Funcionando, Fallida) y sus transiciones.
    - Registro de todos los comandos y sus resultados.
  + **Tecnologías:** Python (lógica de negocio).
  + **Patrones de Diseño Aplicados:**
    - **Command:** Cada operación de control (iniciar bomba, abrir válvula) se encapsula como un objeto comando. Esto permite la historización, reintentos y deshacer operaciones si es necesario.
    - **State:** Fundamental para el manejo de estados de equipos (bombas, válvulas) y estados globales del proceso. Cada estado define el comportamiento de los métodos de control y las transiciones válidas.
    - **Strategy:** Para la implementación de diferentes algoritmos de control o reglas de negocio que pueden ser intercambiables.
* **Servicio de Autenticación y Autorización (Auth Service)**
  + **Descripción:** Gestiona la autenticación de usuarios y la autorización basada en roles (RBAC).
  + **Responsabilidades:**
    - Validación de credenciales de usuario.
    - Generación y validación de tokens de sesión (ej., JWT).
    - Mapeo de usuarios a roles y permisos.
    - Gestión de políticas de contraseñas.
  + **Tecnologías:** Python/Flask o Django, JWT, posible integración LDAP/Active Directory.
* **Servicio de Configuración (Configuration Service)**
  + **Descripción:** Gestiona la configuración persistente del sistema, incluyendo tags, dispositivos, umbrales de alarma, usuarios, roles, y sinópticos.
  + **Responsabilidades:**
    - Proporcionar APIs para la creación, lectura, actualización y eliminación (CRUD) de elementos de configuración.
    - Validación de la integridad de la configuración.
    - Notificación a otros servicios sobre cambios en la configuración que requieran recarga.
  + **Tecnologías:** Python, PostgreSQL (Base de Datos Relacional para configuración).

**Componentes de la Capa de Presentación (Frontend)**

* **Aplicación Web SCADA (SCADA Web Application)**
  + **Descripción:** Interfaz gráfica de usuario basada en web para la interacción del operador.
  + **Responsabilidades:**
    - Visualización de sinópticos dinámicos con datos en tiempo real.
    - Presentación de alarmas y eventos.
    - Interfaz para el envío de comandos de control.
    - Visualización de tendencias históricas y generación de informes.
    - Gestión de usuarios y configuraciones (para administradores).
  + **Tecnologías:** React/Angular/Vue.js (Framework Frontend), WebSockets (para datos en tiempo real), RESTful APIs (para configuraciones, comandos, datos históricos).

**Capa de Persistencia de Datos**

* **Base de Datos Relacional (PostgreSQL)**
  + **Descripción:** Almacena información estructurada y relacional del sistema.
  + **Contenido:**
    - Configuración de tags y dispositivos.
    - Configuración de usuarios y roles.
    - Definiciones de sinópticos y plantillas.
    - Registros de auditoría (acciones de usuario, comandos ejecutados).
    - Logs de eventos del sistema (no datos de proceso).
  + **Tecnologías:** PostgreSQL.
* **Base de Datos de Series Temporales (InfluxDB)**
  + **Descripción:** Optimizado para el almacenamiento y consulta de grandes volúmenes de datos históricos con timestamps.
  + **Contenido:**
    - Valores de los tags de proceso en tiempo real e históricos.
    - Estados de dispositivos históricos.
    - Datos de alarmas (activación, reconocimiento, desactivación).
  + **Tecnologías:** InfluxDB.

**Interacciones entre Componentes y Protocolos**

* **Servicio de Comunicación <-> Dispositivos de Campo:** OPC UA, Modbus/TCP, MQTT.
* **Servicio de Comunicación -> Historian:** Mensajes asíncronos (ej., Kafka/RabbitMQ) para ingesta de datos en tiempo real.
* **Servicio de Comunicación -> Motor de Alarmas:** Mensajes asíncronos (ej., Kafka/RabbitMQ) para notificaciones de cambio de valor y eventos.
* **Aplicación Web SCADA <-> Servicios Backend:**
  + **Datos en Tiempo Real:** WebSockets (para sinópticos, alarmas activas).
  + **Comandos, Configuraciones, Datos Históricos:** RESTful APIs (HTTP/HTTPS).
* **Servicios Backend <-> Base de Datos Relacional:** Conexiones JDBC/ODBC o ORMs.
* **Servicios Backend <-> Base de Datos de Series Temporales:** Cliente nativo de InfluxDB.
* **Servicios de Lógica de Negocio <-> Servicio de Autenticación/Autorización:** Llamadas internas a API para validación de tokens y permisos.

**Patrones de Diseño Aplicados (Detalle ampliado)**

Los patrones de diseño son esenciales para la mantenibilidad, escalabilidad y claridad del código.

* **Observer (Observador)**
  + **Propósito:** Permite definir un mecanismo de suscripción para notificar a múltiples objetos sobre cualquier evento que le suceda al objeto que están observando.
  + **Aplicación en SCADA23:**
    - **Adquisición de Datos:** El Servicio de Comunicación (Sujeto) notifica al Servicio Historian y al Motor de Alarmas (Observadores) cada vez que un nuevo dato de tag es adquirido o un estado de dispositivo cambia. Esto desacopla la adquisición del procesamiento.
    - **Alarmas:** El Motor de Alarmas (Sujeto) notifica a la Interfaz de Usuario y a los servicios de notificación (Observadores) cuando una nueva alarma es generada, reconocida o inactiva.
  + **Implementación:** Uso de sistemas de mensajería (ej., Kafka o RabbitMQ) para publicación/suscripción asíncrona, o patrones de eventos internos para comunicación síncrona en un mismo servicio.
* **Command (Comando)**
  + **Propósito:** Encapsula una petición como un objeto, permitiendo parametrizar clientes con diferentes peticiones, encolar o loguear peticiones, y soportar operaciones des-hacibles.
  + **Aplicación en SCADA23:**
    - **Control Remoto:** Cada operación de control (ej., IniciarBomba, AbrirValvula) se encapsula como un objeto Command. Esto permite:
      * **Auditoría:** Los objetos comando pueden persistirse en una base de datos para registrar quién ejecutó qué comando, cuándo y con qué resultado (RCS-006).
      * **Reintentos/Cola:** Si un comando falla o debe ser ejecutado asíncronamente (ej., si el PLC está ocupado), puede ser reencolado y reintentado sin bloquear la UI.
      * **Deshacer (Potencial):** Aunque complejo en un SCADA en tiempo real, teóricamente permitiría "deshacer" un comando si la lógica de negocio lo permite y el estado lo hace seguro.
  + **Implementación:** Clases que implementan una interfaz Command común, con un método execute(). Uso de una cola de mensajes para la ejecución asíncrona y robusta.
* **State (Estado)**
  + **Propósito:** Permite que un objeto altere su comportamiento cuando su estado interno cambia. El objeto parecerá cambiar su clase.
  + **Aplicación en SCADA23:**
    - **Control de Bombas/Válvulas:** Un objeto Bomba o Valvula podría tener diferentes estados (ej., Parada, Arrancando, Funcionando, Fallida, Mantenimiento). Los métodos (ej., arrancar(), parar()) tendrían un comportamiento diferente dependiendo del estado actual del objeto, simplificando la lógica condicional (if/else anidados) y haciéndola más extensible.
    - **Estados de Proceso Globales:** El sistema global de gestión del proceso (ej., ETAP) podría tener estados como INICIALIZACIÓN, OPERACIÓN\_NORMAL, PARADA\_DE\_EMERGENCIA, MODO\_MANTENIMIENTO. Cada estado definiría qué comandos están permitidos, qué alarmas se suprimen, etc. (referencia a 3.1.3).
  + **Implementación:** Clases de estado que implementan una interfaz de estado común, con el objeto de contexto delegando el comportamiento al objeto de estado actual.
* **Strategy (Estrategia)**
  + **Propósito:** Define una familia de algoritmos, encapsula cada uno, y los hace intercambiables. La estrategia permite que el algoritmo varíe independientemente de los clientes que lo usan.
  + **Aplicación en SCADA23:**
    - **Cálculo de Consumo:** Diferentes algoritmos para calcular el consumo energético de una bomba dependiendo de su tipo (ej., trifásica, monofásica) o datos disponibles.
    - **Lógicas de Alarmas:** Diferentes estrategias para la evaluación de alarmas (ej., umbral fijo, umbral dinámico, tasa de cambio).
    - **Protocolos de Comunicación:** Aunque también puede usarse Adapter, Strategy podría usarse para la implementación de la lógica de envío/recepción específica de cada protocolo.

**Consideraciones de Despliegue y Escalabilidad**

* **Contenedorización:** Todos los componentes del backend (Servicio de Comunicación, Historian, Motor de Alarmas, etc.) DEBEN ser desplegados como contenedores Docker para asegurar la portabilidad y la consistencia en el entorno de producción.
* **Orquestación:** Se utilizará Kubernetes para la orquestación, gestión y escalado automático de los contenedores Docker, proporcionando alta disponibilidad y gestión de cargas de trabajo.
* **Redundancia:** Los componentes críticos (Servicio de Comunicación, Historian, Motor de Alarmas, Base de Datos) DEBEN desplegarse en configuraciones redundantes (ej., active-passive o active-active) para garantizar la alta disponibilidad (RNF-004, RNF-005). La base de datos de series temporales y relacional se configurará para replicación.
* **Balanceo de Carga:** Se utilizarán balanceadores de carga para distribuir el tráfico entre las instancias redundantes de los servicios.

**Tecnologías Clave Seleccionadas**

* **Backend / Lógica de Negocio:** Python (versión 3.x)
* **Framework Web (Frontend):** React (o Angular/Vue.js - A definir en fase de diseño detallado de UI/UX si no está ya fijado).
* **Base de Datos Relacional:** PostgreSQL
* **Base de Datos de Series Temporales:** InfluxDB (versión 2.x o superior)
* **Sistema de Mensajería/Colas:** Apache Kafka o RabbitMQ (para comunicación asíncrona entre servicios)
* **Orquestación de Contenedores:** Kubernetes
* **Contenerización:** Docker
* **Protocolos Industriales:** OPC UA, Modbus/TCP, MQTT
* **API Web:** RESTful APIs (JSON)
* **Comunicación en Tiempo Real (Frontend):** WebSockets

## Interfaces y drivers de conexión

Se muestran a continuación los drivers de comunicación vas a implementar para interactuar con los distintos subsistemas

* OpenPLC: Modbus/TCP
* Factory I/O (Modbus/TCP, OPC UA o el driver específico de Factory I/O para el PLC elegido)

## Protocolos utilizados

Para cada sensor y actuador, se define el protocolo específico y la dirección que usará:

* Por defecto, se utilizará Modbus TCP, dirección 40001 para la dirección de comienzo de los sensores.

A continuación, se muestra una tabla de asignación de E/S y tags de SCADA.

**Pendiente**

# DISEÑO DE LA BASE DE DATOS

Este capítulo detalla el diseño de las bases de datos utilizadas por la plataforma SCADA23, incluyendo el modelo de datos lógico y físico para la persistencia de la configuración del sistema, los usuarios, los registros de auditoría y, de forma separada, los datos históricos de series temporales. Un diseño de base de datos eficiente y robusto es crítico para el rendimiento, la integridad y la escalabilidad del sistema.

## Estrategia de Almacenamiento de Datos

1. SCADA23 empleará una estrategia de almacenamiento de datos híbrida, utilizando dos tipos de bases de datos especializadas para optimizar el rendimiento y la eficiencia en el manejo de diferentes tipos de información:
2. **Base de Datos Relacional (PostgreSQL):** Para datos estructurados, relacionales y transaccionales que requieren alta consistencia, como la configuración del sistema (dispositivos, tags, alarmas), la gestión de usuarios y roles, y los registros de auditoría/eventos de sistema.
3. **Base de Datos de Series Temporales (InfluxDB):** Para el almacenamiento de grandes volúmenes de datos de proceso (valores de tags de sensores/actuadores) en tiempo real e históricos, optimizada para la ingesta rápida y la consulta eficiente de series temporales.

## Diseño de la Base de Datos Relacional (PostgreSQL)

El modelo de base de datos relacional se centrará en la normalización para garantizar la integridad y minimizar la redundancia. A continuación, se presenta un modelo de datos lógico, que se traducirá a un esquema físico.

Se muestra un Diagrama de Entidad-Relación (ERD) detallado para PostgreSQL

**Pendiente**

Se describen a continuación las tablas principales y sus relaciones

**Entidades y Atributos Principales**

* **Usuarios (Tabla para la gestión de usuarios del sistema)**
  + id (PK, UUID): Identificador único del usuario.
  + nombre\_usuario (VARCHAR(50), UNIQUE, NOT NULL): Nombre de usuario para login.
  + password\_hash (VARCHAR(255), NOT NULL): Hash seguro de la contraseña.
  + email (VARCHAR(100), UNIQUE): Correo electrónico del usuario.
  + nombre\_completo (VARCHAR(100)): Nombre y apellidos del usuario.
  + activo (BOOLEAN, DEFAULT TRUE): Indica si la cuenta está activa.
  + ultimo\_login (TIMESTAMP): Fecha y hora del último inicio de sesión.
  + fecha\_creacion (TIMESTAMP, DEFAULT CURRENT\_TIMESTAMP).
  + fecha\_modificacion (TIMESTAMP).
  + intentos\_fallidos (INTEGER, DEFAULT 0): Contador de intentos de login fallidos.
  + bloqueado\_hasta (TIMESTAMP): Si la cuenta está temporalmente bloqueada.
* **Roles (Tabla para definir roles en el sistema)**
  + id (PK, UUID): Identificador único del rol.
  + nombre\_rol (VARCHAR(50), UNIQUE, NOT NULL): Nombre descriptivo del rol (ej., 'Administrador', 'Operador', 'Visualizador').
  + descripcion (TEXT): Descripción del rol.
* **Usuario\_Roles (Tabla de relación muchos a muchos entre Usuarios y Roles)**
  + usuario\_id (PK, FK a Usuarios.id)
  + rol\_id (PK, FK a Roles.id)
* **Permisos (Tabla para definir permisos específicos)**
  + id (PK, UUID)
  + nombre\_permiso (VARCHAR(100), UNIQUE, NOT NULL): Ej., 'control:bomba:iniciar', 'configuracion:tags:editar'.
  + descripcion (TEXT)
* **Rol\_Permisos (Tabla de relación muchos a muchos entre Roles y Permisos)**
  + rol\_id (PK, FK a Roles.id)
  + permiso\_id (PK, FK a Permisos.id)
* **Dispositivos (Tabla para la configuración de equipos de campo)**
  + id (PK, UUID): Identificador único del dispositivo.
  + nombre (VARCHAR(100), NOT NULL): Nombre legible del dispositivo (ej., 'Bomba P-1 ETAP').
  + tipo (VARCHAR(50), NOT NULL): Tipo de dispositivo (ej., 'BOMBA', 'VALVULA', 'SENSOR\_NIVEL').
  + ubicacion (VARCHAR(255)): Descripción de la ubicación física.
  + fabricante (VARCHAR(100)).
  + modelo (VARCHAR(100)).
  + protocolo\_comunicacion (VARCHAR(50), NOT NULL): Ej., 'OPC\_UA', 'MODBUS\_TCP'.
  + direccion\_comunicacion (VARCHAR(255), NOT NULL): Ej., 'opc.tcp://192.168.1.100:4840', '192.168.1.10:502:1'.
  + activo (BOOLEAN, DEFAULT TRUE): Si el dispositivo está habilitado en el sistema.
  + fecha\_instalacion (DATE).
* **Tags (Tabla para la configuración de puntos de datos - variables de proceso)**
  + id (PK, UUID): Identificador único del tag.
  + dispositivo\_id (FK a Dispositivos.id, NOT NULL): Dispositivo al que pertenece el tag.
  + nombre\_tag (VARCHAR(100), UNIQUE, NOT NULL): Nombre del tag en el sistema (ej., 'P\_ETAP\_001\_NIVEL\_AGUA').
  + descripcion (TEXT): Descripción del tag (ej., 'Nivel de agua en depósito principal').
  + tipo\_dato (VARCHAR(20), NOT NULL): Ej., 'FLOAT', 'BOOLEAN', 'INTEGER', 'STRING'.
  + unidad\_medida (VARCHAR(20)): Ej., 'm', '%', 'Bar', 'L/s'.
  + direccion\_protocolo (VARCHAR(255), NOT NULL): Dirección específica del tag en el protocolo (ej., 'ns=2;s=Channel1.Device1.Tag1').
  + frecuencia\_muestreo\_ms (INTEGER, DEFAULT 1000): Frecuencia de adquisición en milisegundos.
  + historizar (BOOLEAN, DEFAULT TRUE): Si el tag debe ser historizado.
  + es\_writable (BOOLEAN, DEFAULT FALSE): Si el tag permite escritura (es un punto de control).
* **Umbrales\_Alarma (Tabla para la configuración de umbrales de alarma)**
  + id (PK, UUID).
  + tag\_id (FK a Tags.id, NOT NULL, UNIQUE): El tag al que se aplica el umbral.
  + tipo\_alarma (VARCHAR(50), NOT NULL): Ej., 'ALTO', 'BAJO', 'MUY\_ALTO', 'MUY\_BAJO', 'CAMBIO\_ESTADO'.
  + umbral (NUMERIC): Valor numérico del umbral.
  + severidad (VARCHAR(20), NOT NULL): Ej., 'INFORMACION', 'BAJA', 'MEDIA', 'ALTA', 'CRITICA'.
  + retardo\_activacion\_s (INTEGER, DEFAULT 0): Tiempo en segundos que el valor debe estar fuera de umbral para activar la alarma.
  + mensaje\_alarma (TEXT): Mensaje descriptivo para la alarma.
  + activo (BOOLEAN, DEFAULT TRUE).
* **Logs\_Auditoria (Tabla para el registro de eventos de auditoría y comandos)**
  + id (PK, UUID).
  + timestamp (TIMESTAMP WITH TIME ZONE, NOT NULL, DEFAULT CURRENT\_TIMESTAMP).
  + usuario\_id (FK a Usuarios.id, NULLABLE): Usuario que realizó la acción (NULL si es acción de sistema).
  + tipo\_evento (VARCHAR(50), NOT NULL): Ej., 'LOGIN', 'LOGOUT', 'COMANDO\_CONTROL', 'CONFIG\_CAMBIADA', 'ALARMA\_RECONOCIDA'.
  + nivel\_severidad (VARCHAR(20)): Ej., 'INFO', 'ADVERTENCIA', 'ERROR', 'CRITICO'.
  + componente (VARCHAR(50)): Componente del sistema que generó el log (ej., 'Auth Service', 'Control Service').
  + descripcion (TEXT, NOT NULL): Descripción detallada del evento.
  + datos\_adicionales (JSONB): Cualquier información extra estructurada relevante para el evento (ej., {"dispositivo\_id": "...", "comando": "iniciar", "resultado": "EXITO"}).

**Consideraciones del Diseño Relacional:**

* **Índices:** Creación de índices adecuados en columnas frecuentemente consultadas (ej., nombre\_usuario en Usuarios, dispositivo\_id y nombre\_tag en Tags, timestamp en Logs\_Auditoria).
* **Restricciones:** Definición de restricciones NOT NULL, UNIQUE y claves foráneas (FOREIGN KEY) para garantizar la integridad de los datos.
* **Transacciones:** Uso de transacciones para asegurar la atomicidad de operaciones críticas (ej., crear usuario, modificar configuración).
* **Volumen de Datos:** Estimar que la tabla Logs\_Auditoria será la que más crezca. Planificar particionamiento o archivado de datos antiguos si el volumen anual supera los límites de rendimiento.

## Diseño de la Base de Datos de Series Temporales (InfluxDB)

InfluxDB es una base de datos NoSQL optimizada para datos de series temporales, donde el tiempo es un componente central de todos los datos. *InfluxDB organiza los datos en "Buckets" y "Measurements". Se describirá la estructura clave:*

**Estructura de Datos (Buckets y Measurements)**

* **Bucket Principal: scada23\_data**
  + **Propósito:** Almacenar todos los valores de tags de proceso y sus metadatos asociados.
  + **Retención:** Configurado para retener datos por un período mínimo de 12 meses (configurable).
* **Measurement: tag\_values**
  + **Propósito:** Almacenar los valores de los tags adquiridos de los dispositivos de campo.
  + **Tags (Metadatos indexados):**
    - tag\_id (String): Identificador único del tag (corresponde a Tags.id de PostgreSQL).
    - dispositivo\_id (String): Identificador único del dispositivo (corresponde a Dispositivos.id).
    - nombre\_tag (String): Nombre del tag (corresponde a Tags.nombre\_tag).
    - unidad\_medida (String): Unidad del valor (corresponde a Tags.unidad\_medida).
    - tipo\_dato (String): Tipo de dato (ej., 'float', 'boolean', 'integer').
  + **Fields (Valores de datos):**
    - value\_float (Float): Valor numérico para tags de tipo FLOAT/INTEGER (NULL si no aplica).
    - value\_bool (Boolean): Valor booleano para tags de tipo BOOLEAN (NULL si no aplica).
    - value\_string (String): Valor de texto para tags de tipo STRING (NULL si no aplica).
    - status\_calidad (Integer, opcional): Código de calidad del dato (ej., 0=bueno, 1=dudoso, 2=malo).
  + **Timestamp:** Automáticamente gestionado por InfluxDB, con precisión de nanosegundos (ns).
* **Measurement: device\_states**
  + **Propósito:** Registrar los cambios de estado de los dispositivos (ej., Bomba: 'Parada' a 'Funcionando').
  + **Tags:**
    - dispositivo\_id (String): Identificador único del dispositivo.
    - nombre\_dispositivo (String): Nombre legible del dispositivo.
  + **Fields:**
    - state (String): El nuevo estado (ej., 'FUNCIONANDO', 'FALLIDA').
    - previous\_state (String): El estado anterior.
  + **Timestamp:** Automático.
* **Measurement: alarm\_events**
  + **Propósito:** Registrar los eventos de ciclo de vida de las alarmas (activación, reconocimiento, inactivación).
  + **Tags:**
    - alarma\_id (String): Identificador único de la instancia de alarma.
    - tag\_id (String): Tag asociado a la alarma.
    - tipo\_alarma (String): Ej., 'ALTO', 'BAJO'.
    - severidad (String): 'CRITICA', 'ALTA', 'MEDIA', 'BAJA', 'INFO'.
  + **Fields:**
    - event\_type (String): 'ACTIVATED', 'ACKNOWLEDGED', 'CLEARED'.
    - value\_at\_event (Float/String/Boolean): Valor del tag en el momento del evento.
    - user\_id (String, opcional): ID del usuario que realizó la acción (ej., reconocimiento).
    - message (String): Mensaje de la alarma.
  + **Timestamp:** Automático.

**Consideraciones del Diseño de Series Temporales:**

* **Esquema de Nombramiento:** Convenciones claras y consistentes para buckets, measurements, tags y fields.
* **Cardinalidad de Tags:** Minimizar la cardinalidad excesiva de los tags de InfluxDB (campos indexados) para optimizar el rendimiento de escritura y consulta. Los tag\_id y dispositivo\_id son buenos candidatos.
* **Consultas:** Utilizar la sintaxis de Flux (o InfluxQL) para consultas eficientes de rangos de tiempo, agregaciones y downsampling.
* **Downsampling (Opcional):** Para datos a muy largo plazo, se podría implementar una estrategia de downsampling (ej., datos cada 1 minuto para el último mes, cada 10 minutos para el último año) para reducir el espacio de almacenamiento y acelerar las consultas históricas de muy largo plazo.

## Volúmenes de Datos y Crecimiento Esperado

* **Tags de Proceso:** Se estima un promedio de 10.000 tags iniciales, con una tasa de muestreo promedio de 5 segundos por tag.
  + Esto equivale a (10.000 tags \* 12 datos/minuto \* 60 minutos/hora \* 24 horas/día \* 365 días/año) = ~6.3 mil millones de puntos de datos al año.
  + Se anticipa un crecimiento del 10-20% anual en el número de tags.
* **Logs de Auditoría:** Se estima un volumen de 500.000 a 1 millón de registros de auditoría al año.
* **Planificación de Almacenamiento:** El diseño contempla que la base de datos de series temporales requerirá mayor capacidad de almacenamiento y IOPS, mientras que la base de datos relacional requerirá mayor énfasis en la consistencia transaccional y la capacidad de consulta compleja.

## Estrategias de Backup y Restauración

* Se establecerán rutinas de backup diarias para ambas bases de datos.
* Para PostgreSQL, se utilizarán backups completos y diferenciales/log shipping para Point-In-Time Recovery.
* Para InfluxDB, se utilizarán las herramientas de backup integradas que permiten copias de seguridad de datos incrementales.
* Se definirá un Objetivo de Punto de Recuperación (RPO) y un Objetivo de Tiempo de Recuperación (RTO) en el Plan de Recuperación ante Desastres, basados en los requisitos de disponibilidad (RNF-004, RNF-005).

# DISEÑO DEL INTERFAZ DE USUARIO Y HMI

## Introducción

Este capítulo define los principios de diseño y los componentes clave de la interfaz de usuario (UI) y la experiencia de usuario (UX) de la plataforma SCADA23. Un diseño UI/UX efectivo es fundamental para garantizar la usabilidad, la eficiencia operativa y la toma de decisiones rápida y precisa por parte de los operadores en entornos críticos.

## Principios de Diseño UI/UX para SCADA23

El diseño de la interfaz de SCADA23 se regirá por los siguientes principios clave, priorizando la claridad, la respuesta rápida y la reducción de la carga cognitiva del operador:

* **Claridad y Simplicidad:** La información se presentará de forma concisa, eliminando elementos superfluos. Los sinópticos y paneles serán fáciles de interpretar a primera vista.
* **Consistencia:** Elementos de UI, iconos, terminología y flujos de interacción serán uniformes en todo el sistema para minimizar la curva de aprendizaje.
* **Retroalimentación Inmediata:** El sistema DEBE proporcionar una respuesta visual o auditiva inmediata a las acciones del usuario y a los cambios en el proceso (ej., estados de dispositivos, confirmaciones de comandos).
* **Eficiencia Operativa:** Se minimizará el número de clics y pasos necesarios para ejecutar tareas críticas de monitoreo y control.
* **Priorización de la Información:** La información más crítica (ej., alarmas activas, estados de equipos críticos) se destacará visualmente para captar la atención del operador.
* **Robustez y Prevención de Errores:** El diseño incluirá mecanismos para prevenir errores humanos, como confirmaciones para comandos críticos y validación de entradas.
* **Adaptabilidad:** La interfaz DEBE ser adaptable a diferentes tamaños de pantalla y dispositivos, aunque priorizando el uso en monitores de control grandes.

## Elementos Clave de la Interfaz de Usuario

**Navegación Global**

* **Barra de Navegación Superior/Lateral:** Contendrá accesos directos a las secciones principales del sistema:
  + Inicio/Dashboard
  + Sinópticos (con submenús para áreas geográficas o estaciones)
  + Alarmas y Eventos
  + Tendencias Históricas
  + Informes
  + Administración (accesible solo para roles autorizados)
* **Migas de Pan (Breadcrumbs):** Para indicar la ubicación actual del usuario dentro de la jerarquía de navegación.
* **Búsqueda Global:** Campo de búsqueda para encontrar rápidamente tags, dispositivos, alarmas o sinópticos.

**Sinópticos Dinámicos (Vistas de Proceso)**

* Los sinópticos son la representación gráfica del proceso y son la interfaz principal para el monitoreo.
* **Representación Gráfica:**
  + **Iconografía Estandarizada:** Uso de iconos claros y reconocibles para bombas, válvulas, tanques, sensores, etc. (Ej., bombas circulares, válvulas con forma de rombo).
  + **Colores y Estados:** Uso consistente de colores para indicar el estado de los equipos:
    - **Verde:** Funcionando / Abierto / Normal.
    - **Rojo:** Parado / Cerrado / Alarma Activa / Fallo.
    - **Amarillo/Naranja:** Arrancando / Abriendo / Cerrando / Alarma Reconocida / Advertencia.
    - **Gris:** Fuera de servicio / Deshabilitado / Sin Conexión.
    - **Azul:** En modo de mantenimiento / Manual.
  + **Valores de Tags en Tiempo Real:** Los valores de los tags se mostrarán junto a sus respectivos iconos de equipo o puntos de medición, actualizándose en tiempo real.
  + **Animaciones:** Animaciones sutiles (ej., flujo de agua en tuberías, aspas de bomba girando) para indicar actividad.
* **Interacción:**
  + **Click para Detalle/Control:** Al hacer clic en un equipo (ej., una bomba), se abrirá una ventana contextual (modal/popup) con información detallada del equipo y opciones de control (si el usuario tiene permisos).
  + **Zoom y Panorámica:** Capacidad de hacer zoom y desplazar la vista para explorar sinópticos grandes.
  + **Navegación entre Sinópticos:** Puntos de enlace o botones para navegar rápidamente a sinópticos relacionados (ej., de una vista general de planta a un sinóptico detallado de la estación de tratamiento).
* **Múltiples Vistas:** El sistema permitirá la creación y gestión de múltiples sinópticos, organizados jerárquicamente.

**Gestión de Alarmas y Eventos**

* **Lista de Alarmas Activas:** Una tabla o panel dedicado que muestra solo las alarmas actualmente activas y no reconocidas, ordenada por severidad.
  + **Columnas:** ID Alarma, Timestamp, Dispositivo/Tag, Descripción, Severidad, Estado (Activa/Reconocida), Usuario Reconocedor.
  + **Filtros y Búsqueda:** Opciones para filtrar por severidad, dispositivo, tipo de alarma, etc.
  + **Sonidos de Alarma:** Diferentes sonidos para distintas severidades, con opciones de silenciar temporalmente.
* **Historial de Alarmas:** Una vista para consultar alarmas pasadas, con opciones de filtrado por rango de tiempo.
* **Interacción:**
  + **Reconocimiento Masivo/Individual:** Botones claros para reconocer alarmas seleccionadas o todas las alarmas.
  + **Navegación al Sinóptico:** Un clic en una alarma DEBE navegar al sinóptico donde se originó la alarma.
  + **Detalle de Alarma:** Abrir un modal con información extendida sobre la alarma (causa, valor al momento de la alarma, acciones sugeridas).

**Tendencias Históricas (Gráficos)**

* **Selector de Tags:** Interfaz intuitiva para buscar y seleccionar uno o más tags a visualizar.
* **Selector de Rango de Tiempo:** Opciones predefinidas (última hora, 24h, 7d, 30d) y un selector de fecha/hora personalizado.
* **Visualización de Gráficos:**
  + **Gráficos de Líneas Dinámicos:** Múltiples series de datos en un solo gráfico, con leyendas claras.
  + **Funcionalidades Interactivas:** Zoom, pan (arrastrar), selección de rango, tooltip al pasar el ratón para ver valores exactos y timestamps.
  + **Ejes Claros:** Ejes X (tiempo) e Y (valor con unidades) debidamente etiquetados.
  + **Exportación de Datos:** Opción para exportar los datos del gráfico a CSV.

**Gestión de Informes**

* **Generador de Informes Predefinidos:** Acceso a informes comunes (ej., consumo de energía mensual por bomba, calidad de agua diaria, tiempos de inactividad de equipos).
* **Configuración de Informes:** Interfaz para personalizar parámetros de informes (rango de fechas, selección de equipos).
* **Visualización y Exportación:** Previsualización de informes y opciones para exportar a PDF, Excel, o CSV.

**Módulo de Administración (Roles: Administrador, Ingeniero de Mantenimiento)**

* **Gestión de Usuarios y Roles:** Interfaz para CRUD (Crear, Leer, Actualizar, Eliminar) usuarios, asignar roles, y gestionar permisos.
* **Configuración de Dispositivos y Tags:** Interfaz para dar de alta, modificar y eliminar dispositivos y sus tags asociados.
* **Configuración de Alarmas:** Interfaz para definir y modificar umbrales de alarma, severidades y mensajes.
* **Gestión de Sinópticos:** Herramientas para la creación y edición de sinópticos (ej., editor gráfico drag-and-drop o importación de archivos).
* **Logs del Sistema y Auditoría:** Visores de logs con filtros y opciones de exportación para diagnósticos y auditoría.

## Wireframes y Maquetas

Se incluyen wireframes o maquetas de las pantallas principales del SCADA, usando sarían herramientas como Figma, Sketch o Balsamiq

* **Maqueta 1: Dashboard Principal:**
  + Panel superior con estado general (num. alarmas activas, resumen de energía).
  + Mapa conceptual de la planta con puntos clave (ETAP, EBAP, Depósitos) indicando estado de alto nivel.
  + Mini-panel de "Alarmas Recientes" con las 5 alarmas más críticas.
* **Maqueta 2: Sinóptico Detallado de ETAP:**
  + Diagrama P&ID simplificado de la ETAP con iconos de bombas, válvulas, tanques.
  + Lecturas de sensores superpuestas en el diagrama (ej., nivel en un tanque, caudal en una tubería).
  + Pop-up de control de bomba al hacer clic: muestra estado, opciones de control, y un mini-gráfico de tendencia de caudal/presión reciente.
* **Maqueta 3: Pantalla de Gestión de Alarmas:**
  + Tabla principal con columnas para ID, Fecha/Hora, Descripción, Dispositivo, Severidad, Estado, Reconocido por.
  + Botones "Reconocer Seleccionadas", "Silenciar", "Exportar".
  + Filtros en la parte superior para Severidad, Dispositivo, Rango de Fecha.

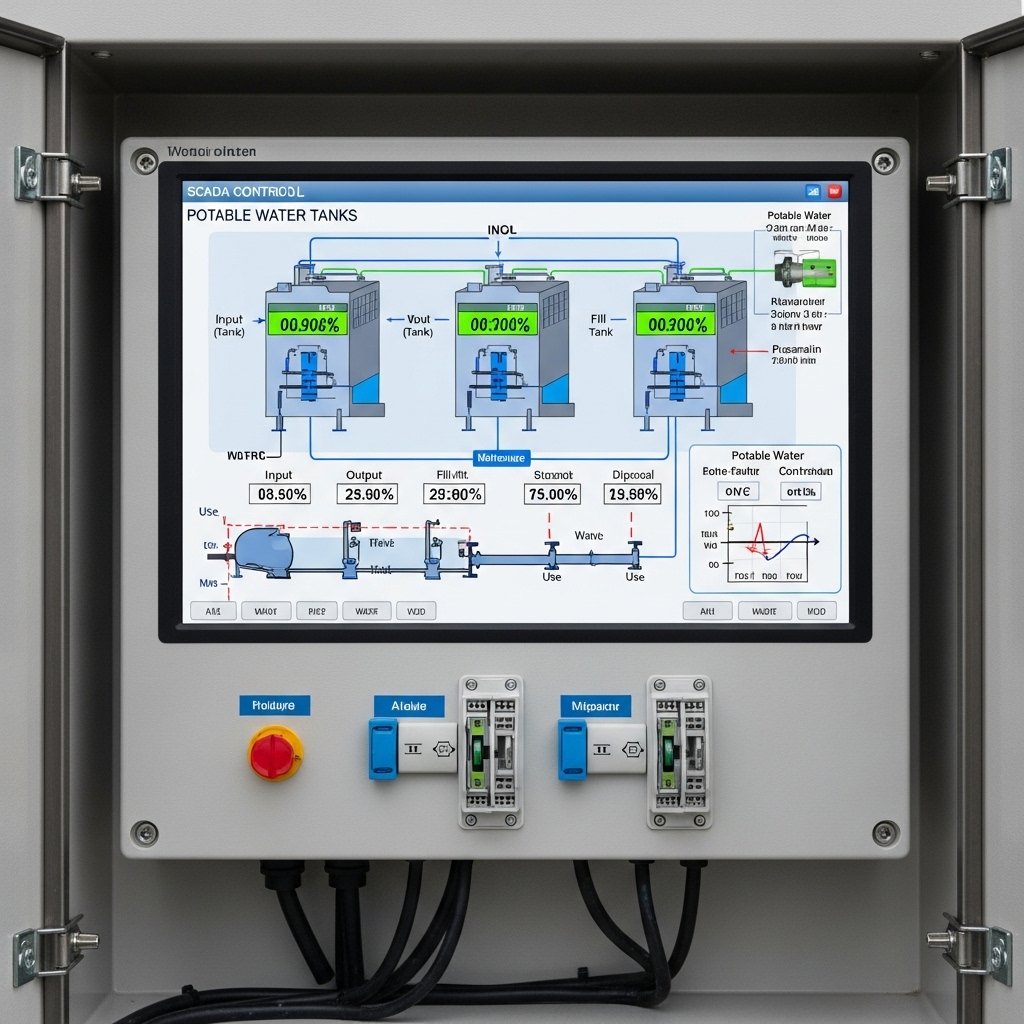
## Consideraciones de Usabilidad y Accesibilidad

* **Teclas de Acceso Rápido (Hotkeys):** Para operaciones frecuentes (ej., reconocimiento de alarmas, cambio de sinóptico).
* **Personalización:** Opción para que los usuarios puedan guardar sus configuraciones de pantalla o paneles favoritos.
* **Mensajes de Error Claros:** Mensajes informativos y útiles, con sugerencias para la resolución.
* **Internacionalización (i18n):** Diseño que permite la fácil traducción de textos para futuros mercados (aunque la primera versión será en español).
* **Diseño Responsivo (Responsive Design):** La interfaz DEBE ser accesible desde tabletas y pantallas más pequeñas, aunque la experiencia completa se optimizará para monitores de escritorio grandes.

## HMI (Human Management Interface)

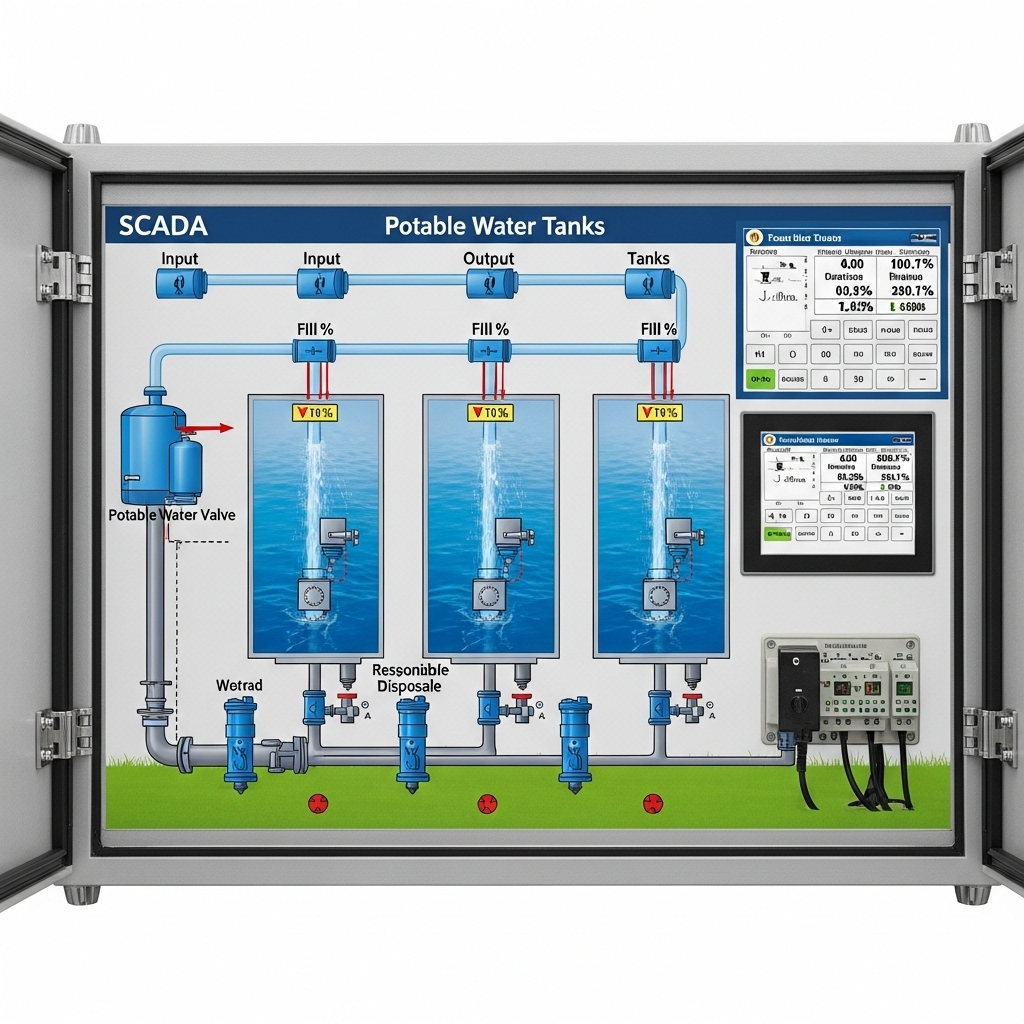
Se muestran ejemplos generados con Inteligencia artificial, del HMI para la supervisión y control de los tanques de tratamiento de agua potable en una ETAP (Estación de Tratamiento de Agua Potable).

**HMI 1**



Fuente: imagen generada con Inteligencia artificial

**HMI 2**



Fuente: imagen generada con Inteligencia artificial

**HMI 3**

Diagrama

El contenido generado por IA puede ser incorrecto.

Fuente: <https://www.youtube.com/watch?v=1Lwaig7oFW0>

En el futuro se ampliará el diseño del HMI con los siguientes elementos, de acuerdo a los estándares ANSI/ISA-101.01-2015 “Human Machine Interfaces forProcess Automation SystemsProcess Automation Systems”, aprobado el 9 de julio de 2015, que marca una serie de convenciones y normas a la hora del diseño y jerarquía de interfaces HMI utilizados en la automatización industrial de máquinas y procesos.

* Esquemas de las pantallas HMI, incluyendo la distribución de los elementos y la navegación entre pantallas):
  + Una pantalla por cada tanque
  + Una pantalla de resumen (sinóptico)
  + Una pantalla de alarmas

Se organizarán en displays Level 1, Level 2, Level 3 y Level 4 según el estándar ISA 101.

* El usuario interactuará con los elementos (botones para válvulas, campos de entrada para setpoints, etc.)
* Utilización de gráficos para los históricos.
* Se indicará el estado de las válvulas (abierta/cerrada) con colores verde/rojo)
* Se visualizarán las alarmas en la HMI con cambio de color, parpadeo y mensaje emergente).

**Navegación en el HMI**

Se utilizarán los siguientes métodos de navegación, según estándar ISA 101

1. hipervínculos incrustados
2. mostrar símbolos con hipervínculos
3. menús (desplegables o planos)
4. árboles
5. pestañas
6. barras de herramientas/cintas
7. paneles de control/paneles de tareas
8. botones en las pantallas
9. botones de teclado personalizados (botones de función y otros botones personalizados)
10. menús contextuales (por ejemplo, clics derechos del botón del ratón para un menú desplegable)
11. mecanismos de mostrar/ocultar para información detallada
12. enlaces a directorios de archivos
13. botones de transferencia de pantalla
14. comandos de voz
15. invocación.

**Operaciones en el HMI**

Las operaciones posibles en el HMI serán las indicadas en el estándar ISA 101. La capacitación en operaciones debe incluir el uso de la HMI para realizar las tareas operativas requeridas, estas pueden incluir:

a) interacción con el sistema de control en todos los modos de operación,

b) uso del sistema de alarmas,

c) reconocimiento de condiciones de control anormales como valores forzados,

d) respuesta a perturbaciones del proceso o del control,

e) comprensión del diseño de navegación de la pantalla,

f) recuperación de datos históricos,

g) ajuste de puntos de consigna **(setpoints**).

h) ajuste de parámetros como modos y salidas,

i) inicio de un lote o inicio de una secuencia preprogramada,

j) arranque o parada de un proceso continuo complejo,

k) reconocimiento de condiciones operativas especiales de equipos o dispositivos, como modos de mantenimiento,

l) familiaridad con la filosofía HMI.

El uso de simuladores con una HMI idéntica a la HMI real de la planta es un método aceptable y eficaz de capacitación para el personal de operaciones.

**Proceso de trabajo continuo para el desarrollo del HMI**

Interfaz de usuario gráfica, Diagrama

El contenido generado por IA puede ser incorrecto.

**Roles y perfiles para los usuarios**

Se muestran a continuación los tipos de usuarios para el HMI del sistema SCADA:

a) Usuarios de **operaciones**: usuarios que supervisan y realizan el control y la operación de la planta o instalación (esto puede incluir tanto a los operadores en la sala de control principal como al equipo de operaciones extendido, que puede incluir usuarios remotos y usuarios de dispositivos de interfaz portátiles);

b) Usuarios de **mantenimiento**: usuarios que realizan la resolución de problemas y/o el mantenimiento del proceso, la instrumentación y los elementos de control final y el equipo rotatorio;

c) Usuarios de **ingeniería**: usuarios que realizan modificaciones, adiciones o eliminaciones en la HMI o el sistema de control;

d) Usuarios **administradores**: usuarios que realizan actualizaciones en el propio sistema de control o asignan seguridad a otros usuarios;

e) Usuarios de **gestión**: usuarios que supervisan el funcionamiento de la planta o instalación;

f) Usuarios **analistas**: usuarios que supervisan el sistema para mejorar el rendimiento de la planta;

g) **Otros** usuarios: usuarios que usan o interactúan con el sistema para otros fines, por ejemplo, personal de gestión de calidad.

# INTEGRACIÓN CON OTRAS TECNOLOGÍAS

A continuación, se muestran otras tecnologías similares gratuitas disponibles en el mercado, con las que la idea es que SCADA23 sea compatible:

## OpenPLC

**Introducción a OpenPLC**

OpenPLC es una suite de software de PLC (Controlador Lógico Programable) de código abierto que cumple con el estándar internacional IEC 61131-3. Esto significa que permite a los usuarios diseñar y programar la lógica del PLC utilizando los cinco lenguajes de programación definidos en el estándar: Ladder Logic (LD), Structured Text (ST), Instruction List (IL), Function Block Diagram (FBD) y Sequential Function Chart (SFC). El proyecto OpenPLC tiene como objetivo proporcionar una solución de automatización de bajo costo y flexible, siendo ampliamente adoptado en entornos de investigación, educación y aplicaciones industriales a pequeña escala.

**Características y Principios de Funcionamiento**

El ecosistema de OpenPLC se compone principalmente de dos partes:

* **OpenPLC Editor:** Es el entorno de desarrollo integrado (IDE) donde los ingenieros y desarrolladores crean sus programas PLC. Está basado en el IDE Beremiz y proporciona una interfaz gráfica para la programación en los lenguajes IEC 61131-3. Una vez que el programa se crea, se compila a código C puro.
* **OpenPLC Runtime:** Es el componente multi-plataforma y portátil que ejecuta el código PLC compilado. Su núcleo, escrito en C, permite que se ejecute en una amplia variedad de hardware, desde microcontroladores de 8 bits como Arduino, hasta potentes servidores de 64 bits o dispositivos de computación de borde como Raspberry Pi y ESP32. El Runtime maneja la interacción con las entradas y salidas (I/O) físicas y lógicas del sistema.

**Principios de Funcionamiento:** OpenPLC sigue el ciclo de escaneo típico de un PLC:

1. **Lectura de Entradas:** El Runtime lee el estado actual de todas las entradas físicas.
2. **Ejecución del Programa:** La lógica de control programada por el usuario se ejecuta basándose en los estados de entrada y el estado interno del PLC.
3. **Actualización de Salidas:** El Runtime actualiza el estado de las salidas físicas según el resultado de la ejecución del programa.
4. **Mantenimiento y Comunicaciones:** Se realizan tareas internas y se gestionan las comunicaciones (por ejemplo, Modbus/TCP, DNP3) antes de repetir el ciclo.

**Ventajas de OpenPLC**

* **Código Abierto y Gratuito:** Elimina los costos de licencias de software y hardware propietario, lo que lo hace accesible para pequeñas empresas, educación e investigación.
* **Compatibilidad con IEC 61131-3:** Permite el uso de lenguajes de programación estandarizados, facilitando la curva de aprendizaje para ingenieros familiarizados con PLCs industriales.
* **Multi-plataforma:** La capacidad de su Runtime para ejecutarse en una gran variedad de hardware (Arduino, Raspberry Pi, ESP32, etc.) ofrece una flexibilidad sin precedentes para prototipos y soluciones personalizadas.
* **Flexibilidad y Personalización:** Al ser de código abierto, permite a los usuarios modificar, adaptar y extender su funcionalidad para satisfacer necesidades específicas del proyecto.
* **Comunidad Activa:** Aunque más pequeña que las comunidades de PLCs comerciales, existe una comunidad de desarrolladores y usuarios que comparten conocimientos y soluciones.
* **Ideal para Investigación:** Ha sido utilizado como plataforma para la investigación en ciberseguridad industrial y sistemas de control.

**Desventajas de OpenPLC**

* **Seguridad:** Históricamente, OpenPLC ha tenido vulnerabilidades de seguridad significativas (ej. buffer overflows, fallos de inyección, canales de comunicación inseguros sin cifrado, problemas de autenticación débil). Aunque se están realizando esfuerzos para abordarlas (como la integración de AES), la seguridad debe ser una consideración crítica y gestionada proactivamente en entornos de producción.
* **Fiabilidad en Entornos Industriales Críticos:** Aunque robusto para muchas aplicaciones, los dispositivos de hardware de bajo costo en los que suele ejecutarse (Raspberry Pi, Arduino) no están diseñados para la misma resistencia, tiempo de actividad y ciclo de vida que los PLCs industriales tradicionales, que operan continuamente en entornos hostiles.
* **Soporte y Documentación:** El soporte se basa en la comunidad, lo que puede ser menos inmediato y estructurado que el soporte de proveedores comerciales. La documentación, aunque existente, puede ser menos completa o dispersa en comparación con soluciones propietarias.
* **Tiempo de Desarrollo Adicional:** La implementación de soluciones con OpenPLC puede requerir más tiempo de programación y desarrollo para solucionar problemas inesperados o integrar componentes personalizados, lo que podría compensar los ahorros iniciales en hardware/software.
* **Depuración en Línea:** El soporte para la programación y depuración en línea (online programming and debugging) no está tan desarrollado o es tan fácil como en PLCs comerciales, lo que puede complicar la resolución de problemas en tiempo real.

**Interacción y Compatibilidad con un Sistema SCADA**

Un sistema SCADA puede interactuar con OpenPLC de diversas maneras, siendo la compatibilidad con protocolos industriales clave:

* **Protocolos Estándar:** OpenPLC Runtime soporta protocolos de comunicación industrial ampliamente utilizados como **Modbus/TCP**, **DNP3** y **Ethernet/IP**. Estos protocolos son esenciales para la comunicación entre PLCs y sistemas SCADA.
* **Intercambio de Datos:** El SCADA puede leer y escribir variables (puntos de datos) del OpenPLC a través de estos protocolos. Por ejemplo, utilizando Modbus/TCP, el SCADA puede solicitar el estado de entradas, salidas y registros internos del OpenPLC para visualización y control.
* **HMI Webserver:** OpenPLC incluye un servidor web integrado que puede servir como una interfaz HMI básica para monitorear el estado de las variables. Esto puede ser útil para una supervisión local o como una capa adicional para la integración SCADA.
* **Entorno de Pruebas y Simulación:** OpenPLC, combinado con simuladores como Factory I/O, es una excelente plataforma para el desarrollo y la prueba de la lógica de control PLC y su integración con el SCADA, permitiendo simular procesos industriales sin la necesidad de hardware físico.
* **Aplicaciones IoT y Edge Computing:** Gracias a su capacidad de ejecutarse en dispositivos de borde como el ALPON X4 (basado en Raspberry Pi CM4) con conectividad LTE, OpenPLC puede actuar como una puerta de enlace IoT para sistemas SCADA, permitiendo el procesamiento de datos en el borde y el monitoreo remoto en tiempo real.

Para una integración exitosa, tu sistema SCADA deberá implementar clientes para los protocolos compatibles con OpenPLC (principalmente Modbus/TCP) y manejar la asignación de direcciones de las variables del PLC a los puntos de datos del SCADA.

## ScadaBR

**Introducción a ScadaBR**

ScadaBR es un sistema SCADA (Supervisory Control and Data Acquisition) de código abierto, gratuito y basado en web. Fue desarrollado en Brasil y se ha convertido en una opción popular para la supervisión remota, la automatización industrial, la automatización residencial y aplicaciones de IoT. Al ser una aplicación Java, ScadaBR puede ejecutarse en diversas plataformas (Windows, Linux) mediante un servidor de aplicaciones como Apache Tomcat, y su interfaz de usuario se accede a través de un navegador web estándar.

**Características y Principios de Funcionamiento**

ScadaBR proporciona las funcionalidades esenciales de un sistema SCADA, permitiendo la adquisición de datos, visualización, control, alarmas y gestión de eventos. Sus principios de funcionamiento se centran en la recopilación centralizada de datos y la provisión de una interfaz para la interacción del operador.

**Características Clave:**

* **Motor de Adquisición de Datos:** Soporta más de 20 protocolos de comunicación, lo que le permite conectarse con una amplia gama de dispositivos y equipos industriales. Los protocolos comunes incluyen Modbus TCP/IP, Modbus Serial, DNP3, IEC 101, OPC DA 2.0, ASCII Serial y lectores de archivos, HTTP Listeners/Receivers, y conectores SQL.
* **Visualización Sinóptica (HMI Builder):** Permite a los usuarios crear pantallas gráficas personalizadas directamente en el navegador, representando el proceso industrial con elementos visuales dinámicos que reflejan el estado en tiempo real de los puntos de datos.
* **Gestión de Usuarios y Permisos:** Ofrece control de acceso basado en roles con permisos personalizables para el acceso a datos y el control de dispositivos.
* **Automatización de Procesos:** Permite la creación de eventos programados y scripts (mediante un motor de scripting que usa Javascript con la librería Rhino de Mozilla) para la automatización y el cálculo de valores en tiempo real, así como el establecimiento de puntos de ajuste y la emisión de comandos.
* **Históricos y Reporting:** Almacena datos históricos y genera informes en formatos como HTML y CSV, con capacidad para mostrar gráficos de tendencias.
* **Gestión de Alarmas y Eventos:** Configuración de alarmas sobre puntos de datos con notificaciones por email y SMS. Permite el registro de datos tan pronto como se activa una alarma.
* **Multi-idioma:** Soporte para múltiples idiomas, incluyendo inglés, portugués y español.
* **API REST y SOAP:** Ofrece APIs para integraciones personalizadas con otros sistemas.

**Ventajas de ScadaBR**

* **Solución SCADA Completa y Gratuita:** Es una alternativa de bajo costo a los sistemas SCADA comerciales, ideal para proyectos con presupuestos limitados.
* **Interfaz Intuitiva y Basada en Web:** Facilita el acceso y la operación desde cualquier navegador, sin necesidad de instalación de cliente.
* **Amplia Compatibilidad de Protocolos:** Su extenso soporte de protocolos permite la integración con una gran variedad de equipos existentes en la industria.
* **Flexibilidad:** Permite la creación de vistas gráficas personalizadas y la automatización mediante scripts.
* **Comunidad Activa:** Cuenta con una comunidad de usuarios y desarrolladores, lo que puede ser una fuente de soporte y recursos.
* **Escalabilidad:** Aunque el rendimiento puede depender del hardware, no tiene limitaciones de código en cuanto a número de puntos de datos o conexiones de usuario.
* **Ideal para Proyectos de Investigación y Prototipos:** Su naturaleza de código abierto y sus características lo hacen adecuado para la experimentación y el aprendizaje.

**Desventajas de ScadaBR**

* **Curva de Aprendizaje:** Aunque la interfaz es intuitiva para lo básico, la configuración y el despliegue de funcionalidades avanzadas pueden requerir una curva de aprendizaje pronunciada, especialmente para usuarios sin experiencia en comunicaciones remotas o I/O abstractos. La documentación no siempre está en inglés nativo.
* **Soporte:** El soporte se basa en la comunidad y puede no ser tan robusto o inmediato como el que ofrecen los proveedores comerciales.
* **Rendimiento y Optimización:** En algunos casos, los usuarios han reportado problemas con la precisión de los datos (ej. valores flotantes) o limitaciones en la resolución de trazado de datos en gráficos (máximo dos puntos por segundo), aunque la base de datos almacene todos los puntos.
* **Aspectos de Instalación y Configuración:** La instalación puede ser un poco compleja para usuarios inexpertos, involucrando la configuración de Java Runtime Environment (JRE) y Tomcat. Pueden surgir problemas durante la migración de bases de datos o al actualizar versiones.
* **Ausencia de "Unidades de Ingeniería" en HMI:** Algunos usuarios han señalado la falta de una propiedad directa para unidades de ingeniería (ej. %RH, °C) en la HMI, lo que requiere soluciones alternativas a través de scripts para mostrar la información correctamente.
* **Estabilidad en Entornos Críticos:** Aunque se utiliza en entornos industriales, para aplicaciones de misión crítica con requisitos de alta disponibilidad y tolerancia a fallos, podría necesitar una validación y robustecimiento adicionales.

**Interacción y Compatibilidad con un Sistema SCADA**

Como un sistema SCADA completo en sí mismo, la "interacción" con un sistema SCADA externo sería más bien la integración de ScadaBR como el *componente SCADA* de tu arquitectura, o su capacidad para interactuar con PLCs y otros sistemas.

* **Interacción con PLCs (como OpenPLC):** ScadaBR es totalmente compatible con OpenPLC. Se utiliza comúnmente para conectar con OpenPLC a través del protocolo **Modbus/TCP**. Se configura una "Fuente de Datos" Modbus IP en ScadaBR, apuntando a la dirección IP y puerto del dispositivo OpenPLC. Luego se definen "Puntos de Datos" en ScadaBR, cuyos offsets (direcciones Modbus) deben coincidir con las variables localizadas en el programa OpenPLC.
* **Adquisición de Datos:** Recopilará datos de tus PLCs (como OpenPLC) y otros dispositivos de campo mediante los diversos protocolos que soporta.
* **Visualización y Control:** Proporcionará la interfaz gráfica (HMI) para que los operadores visualicen el estado de los procesos y emitan comandos.
* **Integración con Otros Sistemas:**
  + **Bases de Datos:** Puede interactuar con bases de datos SQL para almacenar datos históricos y para la integración con sistemas MES (Manufacturing Execution System) o ERP (Enterprise Resource Planning), facilitando el flujo de datos a través de la organización.
  + **APIs (SOAP y REST):** Las APIs integradas permiten a otros sistemas o aplicaciones personalizadas acceder y manipular los datos de ScadaBR, o incluso redirigir la comunicación a una versión más moderna como Scada-LTS.
  + **Servicios de Notificación:** Envío de alarmas y eventos a través de email o SMS.

ScadaBR se posiciona como el cerebro de la capa de supervisión y control en tu arquitectura SCADA, conectando la capa de control (PLCs) con la capa de gestión (usuarios, bases de datos).

## Factory I/O

**Introducción a Factory I/O**

Factory I/O es un software de simulación industrial en 3D altamente realista diseñado para el entrenamiento y la validación de la lógica de control de PLCs. Proporciona un entorno virtual que replica escenarios industriales típicos con una gran variedad de piezas industriales (sensores, actuadores, transportadores, robots, etc.) y escenas prediseñadas. Su principal propósito es permitir a los usuarios practicar y probar sus programas de PLC sin necesidad de hardware físico real, reduciendo riesgos y costos.

**Características y Principios de Funcionamiento**

Factory I/O funciona como un "gemelo digital" o un entorno de simulación que se conecta directamente a un PLC (físico o virtual) o a un simulador de PLC.

**Características Clave:**

* **Escenas Industriales Realistas:** Incluye más de 20 escenas prediseñadas que representan aplicaciones industriales comunes (por ejemplo, clasificación de productos, almacenes automáticos, líneas de ensamblaje). Estas escenas pueden usarse tal cual o como punto de partida para proyectos personalizados.
* **Librería de Piezas Industriales:** Una extensa biblioteca de sensores (inductivos, capacitivos, ópticos, etc.), actuadores (motores, cilindros, válvulas), transportadores, emisores/removedores, robots, etc., todos con comportamiento realista.
* **I/O Digitales y Analógicos:** La mayoría de las piezas tienen entradas y salidas digitales y analógicas, lo que permite un control y una monitorización detallados. Por ejemplo, una entrada digital para arrancar/detener una cinta transportadora o una entrada analógica para controlar niveles de líquidos o pesar elementos.
* **Drivers de I/O:** Factory I/O utiliza drivers para interactuar con PLCs reales (Siemens, Allen-Bradley, Omron, etc.), SoftPLCs, simuladores de PLC (como el de Siemens TIA Portal, Rockwell Studio 5000), Modbus, OPC UA y otras tecnologías. Cada edición de Factory I/O (ej., Allen-Bradley Edition, Siemens Edition) viene con un conjunto de drivers específicos.
* **Herramientas de Edición 3D:** Permite a los usuarios construir sus propios escenarios 3D de forma intuitiva, arrastrando y soltando piezas de la biblioteca.
* **Modo Instructor:** Permite a los instructores bloquear opciones o introducir fallos en las escenas para desafiar a los alumnos a resolver problemas.

**Principios de Funcionamiento:** Factory I/O no ejecuta lógica de control por sí mismo; en su lugar, actúa como la "planta" virtual. Un PLC externo (real o simulado) ejecuta la lógica de control.

1. **Conexión:** Factory I/O se conecta al PLC/simulador a través de un driver específico (ej., Modbus TCP/IP, S7-PLCSIM).
2. **Lectura de Entradas (en el SCADA):** Los sensores en la escena virtual de Factory I/O (por ejemplo, un sensor detecta una caja) envían su estado (TRUE/FALSE o un valor analógico) al PLC/simulador a través del driver.
3. **Ejecución de Lógica PLC:** El PLC/simulador procesa esta información de entrada según su lógica de control.
4. **Escritura de Salidas (desde el SCADA):** El PLC/simulador envía comandos de salida (por ejemplo, activar un motor) a Factory I/O a través del driver.
5. **Simulación de Acción:** Factory I/O reacciona a estos comandos, animando la escena (la cinta transportadora se mueve, el cilindro se extiende), y los sensores virtuales se actualizan en consecuencia, cerrando el bucle.

**Ventajas de Factory I/O**

* **Entorno de Aprendizaje y Entrenamiento Realista:** Ofrece una experiencia inmersiva para aprender PLC y automatización, lo que lo hace ideal para estudiantes y profesionales.
* **Reducción de Riesgos y Costos:** Permite probar y validar la lógica de control sin el riesgo de dañar equipos físicos o interrumpir procesos de producción reales. No requiere hardware PLC costoso para las primeras etapas de desarrollo.
* **Validación de Lógica de Control:** Ayuda a identificar errores en la lógica del PLC antes de la implementación en hardware real, ahorrando tiempo y dinero en la puesta en marcha.
* **Amplia Compatibilidad:** Funciona con la mayoría de las marcas de PLC y tecnologías de automatización comunes gracias a sus drivers de I/O.
* **Escenarios Prediseñados y Personalizables:** Acelera el proceso de creación de entornos de prueba y permite la simulación de situaciones específicas.
* **Portabilidad:** Ocupa poco espacio en disco y es fácil de mover, lo que facilita su uso en diferentes máquinas.
* **Seguridad:** Permite a los estudiantes experimentar y modificar su "fábrica virtual" sin ningún riesgo de seguridad.

**Desventajas de Factory I/O**

* **No es un SCADA:** Factory I/O es un simulador de proceso, no un sistema SCADA. No gestiona bases de datos históricas, alarmas, informes ni comunicación con múltiples PLCs en una red SCADA compleja. Se enfoca en la simulación a nivel de campo.
* **Realismo Limitado:** Aunque es muy realista, una simulación nunca puede replicar completamente las complejidades y las condiciones del mundo real (por ejemplo, ruido eléctrico, fallos de hardware, problemas de red imprevistos).
* **Costo de Licencia:** A diferencia de OpenPLC y ScadaBR que son de código abierto y gratuitos, Factory I/O es un software comercial y requiere una licencia.
* **Requerimientos de Sistema:** Para una simulación fluida y visualmente atractiva, Factory I/O puede requerir un hardware de computadora con capacidades gráficas decentes.
* **Dependencia del PLC/Simulador Externo:** Necesita un PLC real o un simulador de PLC para ejecutar la lógica de control. No tiene un motor de lógica PLC integrado.

**Interacción y Compatibilidad con un Sistema SCADA**

La interacción de Factory I/O con un sistema SCADA no es directa en el sentido de que Factory I/O sea un componente de una red SCADA en tiempo real. En cambio, Factory I/O se utiliza en el contexto de un SCADA para:

* **Simulación y Validación de Lógica PLC:** Es una herramienta invaluable para desarrollar y probar la lógica de control del PLC que eventualmente será monitoreada y controlada por tu sistema SCADA. Puedes desarrollar el programa del PLC, probarlo en Factory I/O, y luego integrar el mismo PLC (o su simulador) con tu SCADA.
* **Entrenamiento de Operadores SCADA:** Puedes usar Factory I/O para crear un entorno virtual donde los operadores de tu SCADA puedan practicar el monitoreo y control de procesos simulados. Esto es crucial para la capacitación sin afectar la producción real.
* **Pruebas de Integración SCADA-PLC:** Una vez que la lógica del PLC está validada en Factory I/O, puedes conectar el PLC (real o simulado) a tu sistema SCADA. Factory I/O, al simular el proceso, te permite verificar que el SCADA está adquiriendo los datos correctos del PLC y que los comandos enviados desde el SCADA al PLC (que luego se reflejan en Factory I/O) tienen el efecto deseado en la planta virtual.
* **Visualización de Procesos Simulados:** Aunque Factory I/O tiene su propia representación 3D, los datos de I/O que genera pueden ser mostrados en la HMI de tu sistema SCADA, permitiendo que el SCADA actúe como la interfaz principal para la supervisión de la "planta" simulada de Factory I/O.
* **Protocolos Comunes para Interacción:** Factory I/O utiliza drivers de I/O que a menudo son los mismos protocolos que tu SCADA usaría para comunicarse con PLCs reales (ej., Modbus TCP/IP, OPC UA, drivers específicos de fabricantes como Siemens S7 Communication o Rockwell EtherNet/IP). Esto asegura que la integración entre Factory I/O y tu PLC (y por extensión, tu SCADA) sea realista.

Factory I/O complementa tu sistema SCADA al proporcionar un banco de pruebas seguro y eficiente para el desarrollo y la validación del nivel de control, así como una herramienta poderosa para el entrenamiento de los operadores.

# DISEÑO DE LA INTERFAZ DE PROGRAMACIÓN DE APLICACIONES (API)

Este capítulo describe el diseño de la Interfaz de Programación de Aplicaciones (API) para la plataforma SCADA23. La API será la columna vertebral de la comunicación entre los diferentes componentes del sistema (frontend, servicios de backend) y sentará las bases para futuras integraciones con sistemas de terceros. Se priorizará un diseño RESTful para la mayoría de las operaciones, complementado con WebSockets para la comunicación en tiempo real.

## Principios de Diseño de la API

El diseño de la API de SCADA23 se adhiere a los siguientes principios:

* **RESTful (Mayormente):** Utilización de los principios de REST para operaciones de recursos (CRUD), empleando verbos HTTP estándar (GET, POST, PUT, DELETE) y recursos con URIs significativos.
* **Sin Estado (Stateless):** Cada solicitud HTTP desde el cliente a la API debe contener toda la información necesaria para que el servidor la procese.
* **JSON como Formato de Datos:** Todos los datos de solicitud y respuesta se codificarán en formato JSON para facilitar la interoperabilidad.
* **Documentación Clara:** Cada endpoint, sus parámetros, tipos de datos y respuestas se documentarán exhaustivamente (ej., con OpenAPI/Swagger).
* **Control de Versiones:** La API se versionará (ej., /api/v1/) para permitir la evolución sin romper la compatibilidad con clientes existentes.
* **Seguridad:** Implementación de autenticación (JWT) y autorización (RBAC) en todos los endpoints protegidos.
* **Manejo de Errores Estructurado:** Las respuestas de error DEBEN ser coherentes y proporcionar códigos de estado HTTP apropiados y mensajes de error detallados.
* **Eficiencia:** Optimización para minimizar el tamaño de las respuestas y la latencia.

## Autenticación y Autorización de la API

* **Autenticación:**
  + **Método:** Basado en tokens JSON Web Tokens (JWT).
  + **Flujo:**
    1. El cliente (UI) envía credenciales (nombre de usuario, contraseña) al endpoint de login.
    2. El Servicio de Autenticación verifica las credenciales y, si son válidas, emite un JWT que incluye información del usuario y sus roles.
    3. El cliente almacena el JWT (ej., en localStorage o sessionStorage) y lo incluye en el encabezado Authorization: Bearer <token> de cada solicitud subsiguiente.
  + **Expiración:** Los tokens JWT tendrán un tiempo de vida limitado (ej., 15 minutos para tokens de acceso, 24 horas para refresh tokens) para mitigar riesgos de seguridad.
* **Autorización:**
  + **Mecanismo:** El Servicio de Autenticación (o un middleware de seguridad en cada servicio) validará el JWT y verificará que el usuario autenticado tiene los permisos requeridos para acceder al recurso o ejecutar la operación solicitada, basándose en el RBAC definido en el Capítulo 2.3 y 6.2.
  + **Respuestas de Error:**
    1. 401 Unauthorized: Si el token es inválido, expirado o no se proporciona.
    2. 403 Forbidden: Si el token es válido pero el usuario no tiene los permisos necesarios para el recurso/operación.

## Endpoints RESTful Clave (API v1)

A continuación, se describen los endpoints RESTful principales. Todos los endpoints seguirán el prefijo /api/v1.

**Gestión de Autenticación**

* **POST /api/v1/auth/login**
  + **Descripción:** Autentica un usuario y devuelve un token JWT.
  + **Request Body (JSON):**

JSON

{

"username": "string",

"password": "string"

}

* + **Response (200 OK, JSON):**

JSON

{

"access\_token": "string",

"token\_type": "bearer",

"expires\_in": 900 // segundos

}

* + **Response (401 Unauthorized, JSON):**

JSON

{

"code": "AUTHENTICATION\_FAILED",

"message": "Credenciales inválidas."

}

* **POST /api/v1/auth/refresh\_token**
  + **Descripción:** Refresca un token JWT expirado con un refresh token.
  + **Request Body (JSON):** {"refresh\_token": "string"}
  + **Response:** Similar a /login.

**Gestión de Usuarios y Roles (Requiere rol de 'Administrador')**

* **GET /api/v1/users**
  + **Descripción:** Obtiene una lista de todos los usuarios.
  + **QueryParams (Opcional):** role\_id, search, page, limit.
  + **Response (200 OK, JSON):** [{"id": "uuid", "username": "string", "email": "string", "roles": ["string"]}]
* **GET /api/v1/users/{user\_id}**
  + **Descripción:** Obtiene los detalles de un usuario específico.
* **POST /api/v1/users**
  + **Descripción:** Crea un nuevo usuario.
  + **Request Body (JSON):** {"username": "string", "password": "string", "email": "string", "roles": ["uuid"]}
* **PUT /api/v1/users/{user\_id}**
  + **Descripción:** Actualiza los detalles de un usuario.
* **DELETE /api/v1/users/{user\_id}**
  + **Descripción:** Elimina un usuario.
* **GET /api/v1/roles**
  + **Descripción:** Obtiene una lista de todos los roles disponibles.

**Gestión de Dispositivos y Tags (Requiere rol de 'Administrador' o 'Ingeniero')**

* **GET /api/v1/devices**
  + **Descripción:** Obtiene una lista de todos los dispositivos.
  + **QueryParams:** type, location, active.
  + **Response (200 OK, JSON):** [{"id": "uuid", "name": "string", "type": "string", "location": "string", ...}]
* **GET /api/v1/devices/{device\_id}**
  + **Descripción:** Obtiene los detalles de un dispositivo específico, incluyendo sus tags asociados.
* **POST /api/v1/devices**
  + **Descripción:** Crea un nuevo dispositivo.
* **PUT /api/v1/devices/{device\_id}**
  + **Descripción:** Actualiza los detalles de un dispositivo.
* **DELETE /api/v1/devices/{device\_id}**
  + **Descripción:** Elimina un dispositivo (también debería eliminar sus tags asociados o marcarlos como inactivos).
* **GET /api/v1/tags**
  + **Descripción:** Obtiene una lista de todos los tags.
  + **QueryParams:** device\_id, type, historized, writable, search.
* **GET /api/v1/tags/{tag\_id}**
  + **Descripción:** Obtiene los detalles de un tag específico.
* **POST /api/v1/tags**
  + **Descripción:** Crea un nuevo tag.
* **PUT /api/v1/tags/{tag\_id}**
  + **Descripción:** Actualiza los detalles de un tag.

**Operaciones de Control (Requiere rol de 'Operador' o superior)**

* **POST /api/v1/commands**
  + **Descripción:** Envía un comando de control a un dispositivo específico.
  + **Request Body (JSON):**

JSON

{

"device\_id": "uuid",

"command\_type": "string", // Ej: "START\_PUMP", "STOP\_PUMP", "OPEN\_VALVE", "CLOSE\_VALVE", "SET\_POSITION"

"value": "any", // Opcional, para comandos con valores (ej: posición de válvula)

"confirmation\_token": "string" // Token de confirmación para comandos críticos (ej. CAPTCHA, second factor)

}

* + **Response (202 Accepted, JSON):** Indica que el comando ha sido recibido y está siendo procesado (asíncronamente).

JSON

{

"command\_id": "uuid",

"status": "ACCEPTED",

"message": "Comando en cola para procesamiento."

}

* + **Response (400 Bad Request, JSON):** Validaciones fallidas.
  + **Response (409 Conflict, JSON):** El dispositivo no está en un estado válido para el comando.

**Consulta de Datos Históricos y Alarmas**

* **GET /api/v1/history/tags/{tag\_id}**
  + **Descripción:** Obtiene datos históricos para un tag específico.
  + **QueryParams:**
    - start: Timestamp de inicio (ISO 8601).
    - end: Timestamp de fin (ISO 8601).
    - interval: Intervalo de agregación (ej., 1m para 1 minuto, 1h para 1 hora).
    - aggregation\_function: Función de agregación (ej., mean, sum, min, max, first, last).
  + **Response (200 OK, JSON):**

JSON

[

{"time": "ISO 8601 timestamp", "value": "float"},

...

]

* **GET /api/v1/alarms/active**
  + **Descripción:** Obtiene todas las alarmas actualmente activas.
* **GET /api/v1/alarms/history**
  + **Descripción:** Obtiene el historial de alarmas (activas e inactivas).
  + **QueryParams:** start, end, severity, tag\_id, status (ej., 'ACTIVATED', 'ACKNOWLEDGED', 'CLEARED').
* **POST /api/v1/alarms/{alarm\_id}/acknowledge**
  + **Descripción:** Reconoce una alarma específica.
  + **Request Body (JSON):** {"user\_id": "uuid"} (opcional si se deduce del token JWT).

## Comunicación en Tiempo Real (WebSockets)

* **ws://your-scada-domain/ws/v1/realtime**
  + **Descripción:** Establece una conexión WebSocket para recibir actualizaciones de datos en tiempo real.
  + **Eventos Enviados por el Servidor (JSON):**
    - **tag\_update:** {"type": "tag\_update", "tag\_id": "uuid", "value": "any", "timestamp": "ISO 8601"}
    - **device\_status\_update:** {"type": "device\_status\_update", "device\_id": "uuid", "status": "string", "timestamp": "ISO 8601"}
    - **new\_alarm:** {"type": "new\_alarm", "alarm\_id": "uuid", "tag\_id": "uuid", "message": "string", "severity": "string", "timestamp": "ISO 8601"}
    - **alarm\_status\_update:** {"type": "alarm\_status\_update", "alarm\_id": "uuid", "status": "string", "acknowledged\_by": "string", "timestamp": "ISO 8601"}
  + **Subscripción (Opcional):** El cliente puede enviar un mensaje inicial para suscribirse solo a ciertos tag\_id o device\_id para optimizar el tráfico.

## Estructura de Respuesta de Errores (JSON)

Todas las respuestas de error de la API DEBEN seguir un formato estándar:

JSON

{

"status": "error",

"code": "ERROR\_CODE", // Código de error específico (ej., "INVALID\_INPUT", "RESOURCE\_NOT\_FOUND", "PERMISSION\_DENIED")

"message": "Descripción legible del error.",

"details": { // Opcional: Información adicional sobre el error

"field": "password",

"reason": "La contraseña no cumple con la política de seguridad."

}

}

## Documentación de la API (OpenAPI/Swagger)

* Se mantendrá un archivo de especificación OpenAPI (anteriormente Swagger) que documente todos los endpoints de la API, sus modelos de datos (schemas), parámetros, posibles respuestas y ejemplos.
* Esta especificación se generará y mantendrá actualizada para asegurar la coherencia entre la documentación y la implementación.
* Se proporcionará una interfaz interactiva de Swagger UI para facilitar las pruebas y la comprensión de la API por parte de los desarrolladores frontend y de integración.

# DISEÑO DE LA ARQUITECTURA DE SOFTWARE

Se detallan las directrices y herramientas específicas para la implementación del sistema SCADA23 utilizando el lenguaje de programación Python. El objetivo es establecer un marco de trabajo que asegure la calidad, la eficiencia, la mantenibilidad y la escalabilidad del código.

También se describe la arquitectura de software propuesta para la plataforma SCADA23, centrándose en la estructura interna, la organización de módulos, la identificación de entidades clave y la estrategia para el manejo de errores. El objetivo es proporcionar una visión clara de cómo se organizará y funcionará el código para cumplir con los requisitos funcionales y no funcionales del sistema.

## Convenciones de Codificación

Para garantizar la legibilidad, coherencia y mantenibilidad del código base, se adoptarán las siguientes convenciones de codificación para todo el proyecto Python:

* **Adherencia a PEP 8:** Todos los desarrolladores deberán seguir estrictamente el estándar [PEP 8 - Style Guide for Python Code](https://peps.python.org/pep-0008/). Esto incluye convenciones para nombres de variables, funciones, clases, espaciado, saltos de línea, etc. Se recomienda el uso de herramientas como flake8 o black para la verificación y auto-formateo del código.
* **Docstrings (PEP 257):** Todas las funciones, clases, métodos y módulos deberán incluir docstrings descriptivos, siguiendo las convenciones de [PEP 257 - Docstring Conventions](https://peps.python.org/pep-0257/). Los docstrings deben explicar el propósito, los parámetros de entrada, el valor de retorno y las posibles excepciones.
  + *Ejemplo de formato:* Se podría optar por el formato Google Style Docstrings o Sphinx reStructuredText para facilitar la generación automática de documentación.
* **Manejo de Logs:**
  + Se utilizará el módulo logging estándar de Python para toda la salida de logs.
  + Se definirán niveles de log claros (DEBUG, INFO, WARNING, ERROR, CRITICAL) y se usarán de forma consistente.
  + Se establecerán configuraciones para el manejo de logs en producción (ej., rotación de archivos, salida a consola, envío a sistemas de monitoreo centralizados).
  + Los mensajes de log deben ser claros, concisos y contener información contextual relevante (marca de tiempo, módulo, función, ID de dispositivo, etc.).
* **Comentarios:** Se utilizarán comentarios (#) para explicar la lógica compleja, decisiones de diseño no obvias o áreas que requieran atención futura (ej., TODO:). Los comentarios deben ser concisos y relevantes.
* **Nomenclatura:** Se utilizará un lenguaje claro y descriptivo para los nombres de variables, funciones y clases, evitando abreviaturas ambiguas.

## 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.

## Bibliotecas y Frameworks Clave

Para la construcción de los diferentes módulos del sistema SCADA23, se propone el uso de las siguientes bibliotecas y frameworks de Python, seleccionadas por su robustez, rendimiento, comunidad de soporte y adecuación a los requisitos de un sistema SCADA:

* **Para Operaciones Asíncronas y Concurrencia:**
  + **asyncio (Módulo Estándar de Python):** Se utilizará para la gestión de operaciones de I/O concurrentes y de larga duración, como la comunicación con dispositivos, la gestión de conexiones de red y la ejecución de tareas en segundo plano. asyncio permitirá un manejo eficiente de múltiples flujos de datos sin recurrir a threads complejos, optimizando el rendimiento y la capacidad de respuesta del sistema.
* **Para el Backend de la Interfaz Humano-Máquina (HMI):**
  + **FastAPI:** Framework web moderno, rápido (gracias a Starlette y Pydantic) y con tipado estático (soporte para *type hints* de Python). Es ideal para construir APIs RESTful robustas y eficientes que servirán los datos en tiempo real y las funcionalidades de control a la interfaz de usuario (sea web o de escritorio). Su generación automática de documentación interactiva (Swagger UI / ReDoc) facilitará el desarrollo del frontend.
  + *Alternativa (a evaluar):* **Flask:** Para proyectos con requisitos de backend más ligeros o si se prefiere una mayor flexibilidad y menos "magia", Flask puede ser una alternativa válida, aunque requeriría más configuración manual para funcionalidades como la validación de datos.
* **Para Interacción con Bases de Datos:**
  + **SQLAlchemy (ORM - Object Relational Mapper):** Proporcionará una capa de abstracción para interactuar con la base de datos relacional (ej. PostgreSQL, MySQL) seleccionada para el almacenamiento de datos históricos, alarmas y configuraciones. SQLAlchemy permite trabajar con objetos Python en lugar de sentencias SQL directas, mejorando la seguridad, la portabilidad y la productividad del desarrollo. Se utilizará tanto su Core (para consultas SQL directas cuando sea necesario) como su ORM (para modelado de datos).
* **Para Comunicación con Protocolos SCADA:**
  + **Modbus:**
    - **pymodbus:** Biblioteca de código abierto que soporta tanto Modbus TCP como Modbus RTU (serial). Será la opción principal para la comunicación con dispositivos y PLCs que utilicen este protocolo.
    - *Consideración:* En caso de requisitos muy específicos o de rendimiento extremo, se podría evaluar la implementación de partes del protocolo a bajo nivel con pyserial para Modbus RTU o socket para Modbus TCP, aunque pymodbus es generalmente suficiente.
  + **OPC UA:**
    - **asyncio-opcua o opcua-asyncio:** Bibliotecas de código abierto que proporcionan un cliente y servidor OPC UA compatible con asyncio. Si se requiere la integración con sistemas que utilizan OPC UA, estas bibliotecas serán fundamentales para una comunicación eficiente y asíncrona.
  + **Otros Protocolos:** En caso de necesitar otros protocolos industriales (ej., DNP3, Ethernet/IP), se buscarán bibliotecas Python existentes o se considerará la implementación de adaptadores específicos (ver Patrón Adapter).
* **Para Visualización de Datos (Backend):**
  + **Matplotlib, Pandas (si se realiza análisis de datos):** Aunque la visualización principal será en el HMI (frontend), estas bibliotecas pueden ser útiles para la generación de informes o análisis de datos históricos en el backend, o para la creación de gráficos que puedan ser servidos como imágenes.

## Identificación de Módulos y Componentes Principales

La arquitectura de software de SCADA23 se diseñará de forma modular para promover la separación de responsabilidades, facilitar el desarrollo concurrente y mejorar la mantenibilidad. Aunque el documento menciona una arquitectura de red, esta sección se enfoca en la estructura lógica del software. Los principales módulos y paquetes Python que conformarán el sistema son:

* **data\_acquisition (Adquisición de Datos):** Responsable de la comunicación con los dispositivos de campo (PLCs, sensores, actuadores) para la lectura de datos y el envío de comandos. Este módulo encapsulará la lógica de los protocolos de comunicación (Modbus, OPC UA, etc.).
  + Submódulos: protocol\_handlers (para Modbus, OPC UA, etc.), device\_drivers (abstracciones de tipos de dispositivos), polling\_engine.
* **control\_logic (Lógica de Control):** Contiene la inteligencia del sistema para el control automático de procesos, la gestión de bucles de control y la implementación de las reglas de negocio y validaciones (ej., lógica de arranque/parada de bombas basada en niveles).
  + Submódulos: automation\_routines, interlock\_manager, pid\_controllers (si aplica).
* **hmi\_backend (Backend de la Interfaz HMI):** Proporciona la API RESTful para la interfaz de usuario. Recibe solicitudes del frontend (visualización, comandos del operador) y se comunica con los módulos de adquisición de datos, lógica de control y base de datos para procesarlas y devolver las respuestas adecuadas.
  + Submódulos: api\_routes, data\_serializers, authentication\_authorization.
* **database\_manager (Gestor de Base de Datos):** Encargado de la interacción con la base de datos para el almacenamiento y la recuperación de datos históricos de proceso, eventos, alarmas, configuraciones y datos de usuario. Utilizará un ORM como SQLAlchemy para esta interacción.
  + Submódulos: models (definición de esquemas de datos), crud\_operations, db\_migrations.
* **alarm\_event\_manager (Gestor de Alarmas y Eventos):** Centraliza la detección, generación, almacenamiento y notificación de alarmas y eventos del sistema. Recibe entradas de data\_acquisition y control\_logic.
  + Submódulos: alarm\_detection, event\_logging, notification\_handlers.
* **configuration\_manager (Gestor de Configuración):** Maneja la carga, validación y gestión de la configuración del sistema (ej., umbrales, direcciones de dispositivos, parámetros de red).
* **utils (Utilidades):** Módulo para funciones auxiliares y herramientas comunes utilizadas en todo el sistema (ej., funciones de tiempo, helpers de validación, herramientas de cifrado).

## Clases y Estructuras de Datos Fundamentales

Aunque el diseño a nivel de clase completa se realizará en etapas posteriores de la ingeniería de software, se identificarán las entidades clave del dominio y sus atributos principales para guiar la estructura del código. Esto sentará las bases para el modelado de la base de datos y la representación en memoria de los elementos del sistema.

* **Device (Clase Base para Dispositivos):**
  + Atributos: id, name, type (Bomba, Válvula, Sensor), address (dirección Modbus, OPC UA Node ID), status (Online/Offline, Activo/Inactivo).
  + Métodos: read\_value(), write\_value(), get\_status().
* **Pump (Hereda de Device):**
  + Atributos: current\_state (Encendida, Apagada, Fallo, Arrancando, Parando), operating\_mode (Manual, Auto), flow\_rate (caudal actual), speed (velocidad, si aplica).
  + Métodos: start(), stop(), set\_manual\_mode(), set\_auto\_mode().
* **Valve (Hereda de Device):**
  + Atributos: current\_position (Abierta, Cerrada, Parcial), valve\_type (On/Off, Control).
  + Métodos: open(), close(), set\_position().
* **Sensor (Hereda de Device):**
  + Atributos: sensor\_type (Nivel, Presión, Caudal, Temperatura), current\_value, unit\_of\_measure, min\_range, max\_range.
  + Métodos: get\_reading().
* **Tank:**
  + Atributos: id, name, current\_level, capacity, min\_level\_threshold, max\_level\_threshold.
  + Métodos: get\_level\_status().
* **Alarm:**
  + Atributos: id, timestamp, type (Nivel Alto, Bomba Fallo, Comunicación), severity (Crítica, Mayor, Menor), message, status (Activa, Acusada, Borrada), source\_device\_id.
* **User:**
  + Atributos: id, username, hashed\_password, role (Operador, Administrador).

Estas clases servirán como la base para el modelado de datos en la base de datos y la representación en memoria de los elementos del sistema, facilitando la interacción con la lógica de control y la interfaz de usuario.

## Estrategia de Manejo de Errores y Excepciones

Una estrategia robusta de manejo de errores y excepciones es crucial para la estabilidad y fiabilidad de un sistema SCADA. Se establecerán las siguientes directrices:

* **Identificación de Tipos de Errores:**
  + **Errores de Comunicación:** Fallos al leer o escribir datos en dispositivos (Modbus, OPC UA).
  + **Errores de Hardware:** Fallos reportados por los dispositivos (ej., bomba en falla, sensor fuera de rango).
  + **Errores de Lógica de Negocio:** Incumplimiento de reglas de negocio (ej., intentar arrancar una bomba con válvula cerrada).
  + **Errores de Base de Datos:** Fallos en la conexión, consultas o integridad de datos.
  + **Errores de Configuración:** Parámetros de configuración inválidos o faltantes.
  + **Errores de Conexión de Red:** Fallos en la conectividad de red entre componentes.
* **Mecanismos de Detección:**
  + Uso de bloques try-except de Python para capturar excepciones en el punto donde ocurren.
  + Validación de entradas de datos y estados del sistema antes de ejecutar operaciones críticas.
  + Mecanismos de *timeouts* para operaciones de comunicación y respuesta de dispositivos.
  + Monitorización constante del estado de los dispositivos y las conexiones.
* **Registro de Errores:**
  + Todos los errores y excepciones detectadas serán registrados utilizando el módulo logging de Python.
  + Los logs incluirán información detallada: marca de tiempo, nivel de error, módulo/función donde ocurrió el error, mensaje descriptivo, y *stack trace* (si aplica).
  + Se establecerán diferentes niveles de log (WARNING, ERROR, CRITICAL) para diferenciar la severidad.
* **Manejo de Excepciones (Graceful Degradation):**
  + Siempre que sea posible, el sistema debe intentar recuperarse de los errores o degradar su funcionalidad de forma controlada en lugar de fallar completamente.
  + **Reintentos:** Para errores transitorios (ej., fallos de comunicación temporales), se implementarán mecanismos de reintento con *backoff* exponencial.
  + **Default Values/Fallback:** Si la lectura de un sensor falla, el sistema podría usar el último valor conocido o un valor por defecto seguro, notificando la falla.
  + **Aislamiento de Fallos:** Un error en un módulo no debe paralizar todo el sistema. Se diseñarán los módulos para ser lo más independientes posible.
* **Notificación de Errores:**
  + Los errores críticos o que afecten la operación normal del sistema deberán generar alarmas visibles en la HMI y, si es necesario, enviar notificaciones (email, SMS, etc.) a los operadores y personal de mantenimiento.
  + Los mensajes de error presentados al usuario deben ser claros, concisos y accionables, evitando mostrar detalles técnicos internos.

## Estrategia de Pruebas

Una estrategia de pruebas rigurosa es fundamental para asegurar la fiabilidad y el correcto funcionamiento del sistema SCADA23. Se adoptará un enfoque de pruebas multifacético que incluirá:

* **Pruebas Unitarias:**
  + **Alcance:** Prueba el componente más pequeño e independiente del código (funciones, métodos de clase) de forma aislada.
  + **Herramienta:** Se utilizará pytest como framework principal para las pruebas unitarias. pytest es flexible, potente y permite escribir pruebas de forma concisa.
  + **Mocks/Stubs:** Se utilizarán herramientas de mocking (unittest.mock o pytest-mock) para aislar los componentes bajo prueba, simulando el comportamiento de dependencias externas (ej., bases de datos, dispositivos de hardware, APIs externas).
  + **Cobertura de Código:** Se buscará una alta cobertura de pruebas unitarias para asegurar que la mayor parte de la lógica de negocio esté probada.
* **Pruebas de Integración:**
  + **Alcance:** Verifican la interacción entre diferentes módulos o componentes del sistema (ej., backend de HMI con base de datos, módulo de adquisición de datos con drivers de protocolo).
  + **Herramienta:** También se utilizará pytest, pero en un entorno donde los componentes reales interactúen (ej., un servidor de base de datos de prueba, un simulador de dispositivo Modbus).
  + **Frecuencia:** Se ejecutarán de forma regular, idealmente como parte de un pipeline de integración continua.
* **Pruebas de Sistema (End-to-End):**
  + **Alcance:** Prueban el sistema completo como si fuera utilizado por un usuario final, verificando que todas las funcionalidades operan correctamente juntas y cumplen con los requisitos funcionales.
  + **Herramienta:** Para la automatización, se podrían considerar herramientas como Selenium (para pruebas de la interfaz web del HMI) o scripts personalizados que simulen el flujo completo de datos desde la adquisición hasta la visualización y el control.
  + **Entorno:** Se ejecutarán en un entorno que simule lo más fielmente posible el entorno de producción.
* **Pruebas de Rendimiento y Carga:**
  + **Alcance:** Evalúan la capacidad del sistema para manejar un volumen esperado de datos y operaciones, y su tiempo de respuesta bajo carga.
  + **Herramienta:** Se podrían utilizar herramientas como Locust o JMeter para simular múltiples clientes HMI o una alta frecuencia de datos de dispositivos.
* **Pruebas de Seguridad:**
  + **Alcance:** Identifican vulnerabilidades en el sistema (ej., inyección SQL, XSS, fallos de autenticación/autorización).
  + **Herramienta:** Se combinarán auditorías manuales con herramientas automatizadas de análisis de seguridad (SAST/DAST) cuando sea apropiado.

## Gestión de Dependencias

Para asegurar un entorno de desarrollo consistente y la reproducibilidad de las construcciones del sistema, se implementará una gestión de dependencias rigurosa:

* **Entornos Virtuales:**
  + **venv (Módulo Estándar de Python) o conda:** Se utilizarán entornos virtuales para cada proyecto o microservicio de Python. Esto aísla las dependencias del proyecto del entorno global de Python del sistema, evitando conflictos de versiones entre diferentes proyectos.
  + Todos los desarrolladores deberán activar el entorno virtual antes de instalar dependencias o ejecutar el código del proyecto.
* **Archivo requirements.txt:**
  + Todas las dependencias del proyecto y sus versiones específicas serán listadas en un archivo requirements.txt (o similar si se usa Poetry o Pipenv).
  + Este archivo se mantendrá actualizado y será la fuente única de verdad para las dependencias del proyecto.
  + Para la instalación de dependencias, se utilizará el comando pip install -r requirements.txt.
  + Las dependencias de desarrollo (ej., pytest, flake8) se pueden listar en un requirements-dev.txt separado.
* **Versionado Semántico:** Se promoverá el uso de versionado semántico (Major.Minor.Patch) para las bibliotecas de terceros y para el propio código del proyecto SCADA23, facilitando la comprensión de los cambios y la gestión de compatibilidad.
* Estas directrices de implementación aseguran que el código de SCADA23 sea desarrollado de manera estructurada, eficiente y siguiendo las mejores prácticas de la ingeniería de software en Python.

# PLANIFICACIÓN DEL DESARROLLO

La implementación de la Plataforma SCADA23 se llevará a cabo a través de un enfoque por fases, permitiendo la validación progresiva de la tecnología y la mitigación de riesgos. Este capítulo describe la planificación general del proyecto, detallando los objetivos, el alcance, los hitos, los entregables y la estimación temporal para cada una de las dos fases principales: Fase 1 (TRL4) y Fase 2 (TRL7).

Se detalla la planificación para el desarrollo de la plataforma SCADA23, siguiendo una metodología ágil SCRUM.. La fase 1 se centra en el módulo de Agua Potable y la comunicación Modbus/TCP.

## Enfoque por Fases (TRL4 y TRL7)

La estrategia de desarrollo en dos fases se basa en el concepto de Niveles de Madurez Tecnológica (TRL - Technology Readiness Levels), permitiendo un desarrollo incremental y una validación rigurosa en cada etapa.

* **Fase 1: TRL4 – Demostración de Concepto y Viabilidad Básica**
  + **Objetivo:** Validar la conectividad básica con dispositivos industriales (Modbus TCP) y demostrar una funcionalidad elemental de supervisión y control en un entorno controlado (laboratorio o simulación). Esta fase busca confirmar la viabilidad técnica de los componentes principales y establecer las bases de la arquitectura.
  + **Alcance:** Funcionalidad mínima viable para la adquisición de datos, visualización básica y control manual de puntos seleccionados. Uso de tecnologías sencillas y rápidas de implementar.
  + **Beneficios:** Reducción del riesgo tecnológico, aprendizaje temprano sobre la integración de protocolos industriales, y establecimiento de un punto de partida funcional para la fase completa.
* **Fase 2: TRL7 – Plataforma Completa y Preparada para Entorno Operacional**
  + **Objetivo:** Desarrollar una plataforma SCADA completa, robusta, escalable y cibersegura, lista para su despliegue en entornos de producción. Esta fase implica la implementación de funcionalidades avanzadas, soporte multi-protocolo, alta disponibilidad y una interfaz de usuario profesional.
  + **Alcance:** Implementación de todos los requisitos funcionales y no funcionales detallados en el Capítulo 2, incluyendo soporte para Modbus TCP, PROFINET, OPC-UA, bases de datos avanzadas, APIs RESTful y una HMI web completa.
  + **Beneficios:** Sistema SCADA de vanguardia que optimiza las operaciones, mejora la seguridad, facilita la toma de decisiones y proporciona una solución a largo plazo para la gestión de infraestructuras críticas.

## Roles del Equipo

El equipo de desarrollo está compuesto por tres personas

* **Product Owner:** Rafael Ausejo Prieto (contacto: rafael.ausejo@confianza23.es) - Responsable de la visión del producto y la priorización del backlog.
* **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: JSP
  + Desarrollador 3: ESB

## Backlog del Producto

El Backlog del Producto está gestionado y priorizado por el Product Owner. A continuación, se listan algunos ítems iniciales:

**Usuario**

* RU001: Como usuario, quiero que la plataforma simule el comportamiento de PLCs para pruebas.
* RU002: Como usuario, quiero que la plataforma pueda capturar y analizar tráfico Modbus/TCP de PLCs reales.
* RU003: Como usuario, quiero una interfaz gráfica (HMI) para supervisar y controlar sensores y actuadores del sistema de agua potable.
* RU004: Como usuario, quiero que la plataforma soporte completamente el protocolo Modbus/TCP.
* RU005: Como usuario, quiero que la HMI muestre claramente el estado de los sensores y actuadores del sistema de agua potable.
* RU006: Como usuario, quiero poder controlar las válvulas de entrada y salida de los tanques desde el HMI.
* RU007: Como usuario, quiero que se visualicen los datos de los sensores (presión, temperatura, flujo, nivel) en tiempo real en el HMI.
* RU008: 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.
* RU009: Como usuario, quiero simular el llenado y vaciado de tres tanques de agua potable.

**Administrador**

* RA001: Como administrador, quiero que la plataforma cargue por defecto el módulo de "Agua Potable" y sea configurable para otros sectores.

**Desarrollador**

* RD001: Como desarrollador, quiero que el código sea modular y esté documentado.
* RD002: Como desarrollador, quiero que la plataforma sea desarrollada en Python 3.13.4 y se ejecute en Windows 11, utilizando Scapy y tcl/tk.
* RD003: Como desarrollador, quiero que los rangos de medición y puntos de ajuste se carguen desde un archivo de configuración.

## Planificación (Fase 1)

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

**Objetivos Detallados de la Fase 1**

* Desarrollar scripts Python para actuar como clientes Modbus TCP (lectura y escritura).
* Crear una aplicación SCADA básica con interfaz gráfica en Tkinter para visualizar datos y enviar comandos Modbus TCP.
* Establecer comunicación Modbus TCP bidireccional entre la aplicación SCADA y un dispositivo simulado o real.
* Validar la adquisición y visualización de al menos 5 puntos de datos (analógicos y digitales).
* Validar el control de al menos 2 puntos de salida digitales.
* Documentar el proceso de desarrollo y los resultados de las pruebas.

**Hitos y entregables**

|  |  |  |
| --- | --- | --- |
| **Hito** | **Descripción** | **Entregable(s)** |
| H1.1: Configuración de Entorno | Configuración de entornos de desarrollo Python y librerías básicas. | Entorno de Desarrollo Configurado. |
| H1.2: Desarrollo de Clientes Modbus TCP | Implementación de scripts Python para simulación de clientes de campo. | Scripts Python cliente Modbus TCP (lectura/escritura). |
| H1.3: Desarrollo de Módulo Modbus SCADA | Creación del componente de comunicación Modbus TCP para la aplicación SCADA. | Código Python del módulo Modbus TCP del SCADA. |
| H1.4: Desarrollo de Interfaz Tkinter | Diseño e implementación de la interfaz gráfica básica con Tkinter. | Código Python de la interfaz Tkinter. |
| H1.5: Integración y Pruebas Básicas | Integración del módulo Modbus con la interfaz Tkinter y pruebas de comunicación. | Prototipo SCADA TRL4 funcional. Informe de Pruebas Básicas. |
| H1.6: Documentación Fase 1 | Documentación técnica del código y manual de usuario básico para la demostración. | Documento de Diseño y Uso del Prototipo TRL4. |

* **Hito 1 (Sprint 1): 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 2): 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 3): 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 4): 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.

**Duración Estimada y Recursos (Fase 1)**

* **Duración Estimada:** 4-6 semanas.
* **Recursos Humanos:** 1 Desarrollador Python.
* **Recursos Materiales:**
  + Ordenador personal con Python y Tkinter instalado.
  + Un simulador Modbus TCP o un PLC/RTU real compatible con Modbus TCP para pruebas.
  + Conectividad de red LAN.

**Flujo de Trabajo y Metodología (Fase 1)**

1. Se utilizará una metodología ágil y ligera, posiblemente Scrum con ciclos cortos, para permitir una rápida iteración y validación.
2. **Planificación Sprint:** Definición de los objetivos semanales/bisemanales.
3. **Desarrollo y Codificación:** Implementación de las funcionalidades.
4. **Pruebas Unitarias y de Integración:** Verificación de cada componente y su interacción.
5. **Demostración:** Presentación del prototipo funcional al equipo de proyecto.
6. **Retroalimentación:** Recopilación de observaciones para futuras iteraciones o la siguiente fase.

## Planificación (Fase 2)

**Objetivos de la Fase 2**

* Diseñar e implementar una arquitectura de microservicios robusta con Django y FastAPI.
* Integrar bases de datos MySQL (configuración, eventos) e InfluxDB (series temporales).
* Desarrollar módulos de adquisición de datos para Modbus TCP, PROFINET y OPC-UA.
* Crear una API RESTful completa con Django REST Framework.
* Desarrollar una interfaz de usuario web interactiva y profesional (HMI).
* Implementar un sistema avanzado de gestión de alarmas, eventos y tendencias.
* Integrar mecanismos de ciberseguridad robustos en todas las capas del sistema.
* Implementar funcionalidades de gestión de usuarios y permisos (RBAC).
* Garantizar la escalabilidad, fiabilidad y alta disponibilidad del sistema.
* Desplegar el sistema utilizando Docker para facilitar su gestión y portabilidad.
* Generar documentación técnica y de usuario completa.

**Hitos y entregables (Fase 2)**

|  |  |  |
| --- | --- | --- |
| **Hito** | **Descripción** | **Entregable(s)** |
| H2.1: Diseño Arquitectónico Detallado | Definición exhaustiva de la arquitectura de microservicios, bases de datos y APIs. | Documento de Arquitectura de Software v2.0. Diagramas de Componentes. |
| H2.2: Configuración de Entorno de Desarrollo | Configuración de entornos de desarrollo con Docker y herramientas asociadas. | Entorno de Desarrollo Distribuido Operativo. |
| H2.3: Implementación de Backend Core (Django/FastAPI) | Desarrollo de la estructura base del backend, modelos de datos, autenticación. | Backend Core con APIs de Autenticación y Modelos de Datos en MySQL. |
| H2.4: Integración de Bases de Datos | Conexión y configuración de MySQL e InfluxDB. Optimización inicial. | Bases de Datos Configuradas y Optimizadas. |
| H2.5: Módulos de Adquisición (Modbus/PROFINET/OPC-UA) | Desarrollo e integración de los drivers de comunicación con FastAPI. | Módulos de Adquisición Funcionales. |
| H2.6: Desarrollo de APIs RESTful Completas | Exposición de todas las funcionalidades a través de Django REST Framework. | API RESTful Completa y Documentada (Swagger/OpenAPI). |
| H2.7: Desarrollo de HMI Web (Fase Inicial) | Creación de los primeros paneles de visualización y control en la interfaz web. | Prototipo de HMI Web con visualización de datos en tiempo real. |
| H2.8: Implementación de Gestión de Alarmas y Eventos | Desarrollo de la lógica de detección, notificación y gestión de alarmas. | Módulo de Alarmas y Eventos Funcional. |
| H2.9: Implementación de Módulo de Ciberseguridad | Integración de las medidas de seguridad (RBAC, cifrado, auditoría). | Módulo de Ciberseguridad Activo. |
| H2.10: Pruebas de Integración y Rendimiento | Pruebas exhaustivas de todos los componentes y rendimiento del sistema. | Informe de Pruebas de Integración y Rendimiento. |
| H2.11: Despliegue en Entorno de Prueba (TRL7) | Despliegue de la plataforma completa en un entorno simulado de operación. | Plataforma SCADA23 TRL7 Desplegada en Entorno de Prueba. |
| H2.12: Documentación Final y Manuales | Generación de la documentación técnica y de usuario exhaustiva. | Documento de Ingeniería de Detalle Final, Manuales de Usuario/Admin. |

**Duración Estimada y Recursos (Fase 2)**

* **Duración Estimada:** 6-9 meses (24-36 semanas).
* **Recursos Humanos:**
  + 1 Arquitecto de Software / Líder Técnico
  + 2-3 Desarrolladores Backend (Python/Django/FastAPI)
  + 1 Desarrollador Frontend (Web HMI)
  + 1 Especialista en Bases de Datos / DevOps
  + 1 Especialista en Ciberseguridad (asesoramiento continuo y revisión)
* **Recursos Materiales:**
  + Servidores (físicos o virtuales) con capacidad adecuada para producción.
  + Licencias de software (si aplica a alguna herramienta específica, aunque la mayoría son de código abierto).
  + Equipos de red (switches, routers, firewalls).
  + PLCs/RTUs reales para pruebas de integración con PROFINET y OPC-UA.
  + Herramientas de monitoreo y logging.

**Flujo de Trabajo y Metodología (Fase 2)**

1. Se recomienda una metodología ágil (Scrum) con ciclos de desarrollo más largos (sprints de 2-4 semanas) para gestionar la complejidad de la Fase 2.
2. **Definición de Sprints:** Planificación detallada de las funcionalidades a desarrollar en cada sprint.
3. **Diseño y Modelado:** Diseño específico de componentes, APIs y esquemas de base de datos para cada funcionalidad.
4. **Desarrollo:** Implementación de código, pruebas unitarias y de integración continua.
5. **Revisiones de Código:** Garantizar la calidad y robustez del código.
6. **Pruebas de Calidad:** Pruebas funcionales, de rendimiento, de carga, de seguridad, de usabilidad.
7. **Despliegue Continuo/Integración Continua (CI/CD):** Automatización de la construcción, prueba y despliegue del software.
8. **Demostraciones Periódicas:** Presentación de avances a los stakeholders.
9. **Gestión de Riesgos:** Identificación y mitigación proactiva de riesgos técnicos, operativos y de seguridad.
10. **Documentación Continua:** Actualización de la documentación a medida que avanza el desarrollo.

**Riesgos y mitigaciones**

|  |  |
| --- | --- |
| **Riesgo** | **Mitigación** |
| Complejidad de Integración de Protocolos | Fase 1 valida Modbus TCP.  Investigación y pruebas tempranas con librerías para PROFINET y OPC-UA en la Fase 2.  Uso de FastAPI para desacoplar los módulos de adquisición. |
| Rendimiento de Base de Datos (Series Temporales) | Elección de InfluxDB, optimizado para series temporales. Pruebas de carga tempranas.  Configuración de políticas de retención y compresión. |
| Ciberseguridad | Integración de la seguridad desde el diseño (Security by Design). Auditorías de seguridad periódicas.  Uso de frameworks robustos (Django).  Autenticación, autorización y cifrado en todas las capas.  Plan de respuesta a incidentes. |
| Escalabilidad | Arquitectura de microservicios/componentes desacoplados.  Uso de Docker/Kubernetes.  Diseño de bases de datos escalables. Pruebas de carga y estrés. |
| Cambios de Requisitos | Metodología ágil que permite la adaptación a cambios. Comunicación fluida y constante con los stakeholders. |
| Disponibilidad de Recursos Humanos | Planificación anticipada de la contratación o asignación de personal con las habilidades requeridas (Python, Django, FastAPI, Bases de Datos, Ciberseguridad, DevOps). |
| Conectividad con Equipos de Campo Existentes | Auditoría previa de los equipos existentes en las instalaciones. Pruebas de compatibilidad con diferentes versiones de los protocolos.  Consideración de gateways de protocolo si es necesario. |
| Costo del Proyecto | Control riguroso del presupuesto en cada fase.  Uso de tecnologías de código abierto para minimizar licencias. Estimaciones detalladas y revisión periódica de los costos. |

## 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 utiliza Git para el control de versiones, en github
* **Pruebas:** Se implementan pruebas unitarias y de integración para asegurar la calidad del código, con github actions
* **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.

# METODOLOGÍA DE DESARROLLO DE SOFTWARE

Este capítulo establece la planificación de alto nivel para el desarrollo de la plataforma SCADA23, definiendo las fases clave, los hitos de entrega y los criterios de "Definición de Terminado" (Definition of Done - DoD) para cada módulo principal. El objetivo es proporcionar una hoja de ruta clara para el equipo de desarrollo y los stakeholders, permitiendo una gestión eficaz del proyecto y una visibilidad continua del progreso.

## Metodología de Desarrollo

El proyecto SCADA23 se abordará utilizando una metodología de desarrollo **ágil e iterativa**, preferentemente **Scrum**, adaptada a los requisitos de un proyecto de ingeniería de detalle. Esto permitirá la entrega incremental de funcionalidades, la adaptación a cambios y la retroalimentación continua.

* **Sprints:** El desarrollo se organizará en sprints de 2 semanas.
* **Backlog de Producto:** Los requisitos funcionales y no funcionales detallados en los capítulos anteriores formarán la base del Product Backlog, que será priorizado por el Product Owner.
* **Revisiones Iterativas:** Al final de cada sprint, se realizará una revisión con los stakeholders para demostrar el progreso y obtener feedback.

## Fases de Desarrollo y Entregables Clave

El desarrollo se estructurará en las siguientes fases principales, con hitos definidos al final de cada una:

* **Fase 1: Configuración de Infraestructura y Componentes Base**
  + **Duración Estimada:** 4 semanas
  + **Descripción:** Configuración de la infraestructura cloud/servidores, despliegue de Kubernetes, configuración de las bases de datos (PostgreSQL, InfluxDB), y desarrollo de los servicios de autenticación y configuración básicos. Establecimiento de la pipeline CI/CD.
  + **Entregables Clave:**
    - Entorno de desarrollo y pruebas funcional.
    - Esquemas de bases de datos implementados.
    - Servicio de Autenticación operativo (login/logout).
    - Servicio de Configuración CRUD para usuarios y roles.
    - Pipeline CI/CD configurada para despliegues automatizados.
  + **Hito 1: "Infraestructura Operativa y Core de Seguridad"**
    - **Criterios de Aceptación del Hito:**
      * Los entornos de desarrollo y pruebas están configurados y accesibles.
      * Los servicios de autenticación y configuración básica están desplegados y son accesibles vía API.
      * La autenticación de usuarios funciona correctamente con la política de contraseñas definida (RCS-001).
      * Los logs de auditoría de login/logout se registran correctamente (RCS-006).
      * Las bases de datos están instaladas, configuradas y son accesibles desde los servicios.
* **Fase 2: Adquisición de Datos y Historización**
  + **Duración Estimada:** 6 semanas
  + **Descripción:** Implementación del Servicio de Comunicación para al menos un protocolo (ej., OPC UA), y el Servicio Historian para la ingesta y almacenamiento de datos de tags en InfluxDB.
  + **Entregables Clave:**
    - Servicio de Comunicación funcional (lectura de tags OPC UA).
    - Servicio Historian ingiriendo y persistiendo datos en InfluxDB.
    - API para consulta básica de datos históricos de tags.
    - Módulo de gestión de dispositivos y tags en el servicio de configuración.
  + **Hito 2: "Motor de Datos Funcional"**
    - **Criterios de Aceptación del Hito:**
      * El sistema es capaz de adquirir datos de al menos 500 tags vía OPC UA y persistirlos en InfluxDB con la latencia especificada (RF-001, RNF-001).
      * Los datos históricos de estos tags pueden ser consultados a través de la API (RF-002, RNF-002 para API).
      * La configuración de nuevos dispositivos y tags puede realizarse a través del Servicio de Configuración (RNF-009).
* **Fase 3: Motor de Alarmas y Control Básico**
  + **Duración Estimada:** 8 semanas
  + **Descripción:** Desarrollo del Motor de Alarmas y del Servicio de Control para gestionar los estados de los equipos y ejecutar comandos.
  + **Entregables Clave:**
    - Motor de Alarmas generando alarmas basadas en umbrales fijos.
    - Servicio de Control para comandos básicos (ej., iniciar/parar bomba, abrir/cerrar válvula).
    - API para ejecutar comandos y consultar/reconocer alarmas.
    - Implementación del patrón State para al menos un tipo de dispositivo (ej., BOMBA).
  + **Hito 3: "Supervisión y Control Esencial"**
    - **Criterios de Aceptación del Hito:**
      * El sistema genera alarmas automáticamente al exceder los umbrales configurados para al menos 10 tags (RF-005, RNF-003).
      * Los operadores pueden ejecutar comandos de inicio/parada para al menos 5 bombas o válvulas a través de la API, con registro de auditoría (RF-003, RCS-006).
      * El sistema maneja los estados de las bombas de acuerdo con las transiciones definidas (RF-004).
      * La funcionalidad de reconocimiento de alarmas está operativa vía API (RF-006).
* **Fase 4: Interfaz de Usuario y Experiencia (HMI)**
  + **Duración Estimada:** 10 semanas
  + **Descripción:** Desarrollo de la interfaz web principal, incluyendo sinópticos, visualización de alarmas, tendencias y control básico.
  + **Entregables Clave:**
    - Interfaz de usuario web funcional con navegación y autenticación.
    - Sinópticos dinámicos mostrando datos en tiempo real y estados de equipos (RF-007).
    - Panel de alarmas activas y control de reconocimiento.
    - Visualizador de tendencias históricas (RF-008).
    - Interfaz para ejecutar comandos de control desde la UI.
  + **Hito 4: "SCADA Operable"**
    - **Criterios de Aceptación del Hito:**
      * La interfaz de usuario es intuitiva y sigue los principios de diseño establecidos (Capítulo 7).
      * Los sinópticos se cargan y actualizan en tiempo real dentro de los tiempos de respuesta definidos (RNF-002).
      * El Operador puede monitorear el estado de la planta y ejecutar comandos de control de manera eficaz desde la UI (CU-001, CU-002).
      * Las alarmas son visibles y gestionables desde la interfaz de usuario (CU-003).
      * Los datos históricos pueden ser consultados visualmente en gráficos de tendencias (CU-004).
* **Fase 5: Funcionalidades Avanzadas, Ciberseguridad y Pruebas Finales**
  + **Duración Estimada:** 8 semanas
  + **Descripción:** Implementación de requisitos de ciberseguridad avanzados, lógica de negocio específica (reglas de negocio complejas, alternancia), informes, optimización de rendimiento y pruebas de sistema/integración.
  + **Entregables Clave:**
    - Todos los requisitos de ciberseguridad (Capítulo 2.3) implementados y probados.
    - Reglas de negocio complejas y lógica de control definidas en Capítulo 3 implementadas.
    - Módulo de informes funcionales.
    - Pruebas de rendimiento, escalabilidad y estrés completadas.
    - Documentación de usuario final y de mantenimiento.
  + **Hito 5: "Sistema SCADA23 Listo para Despliegue Piloto"**
    - **Criterios de Aceptación del Hito:**
      * El sistema cumple con todos los requisitos funcionales y no funcionales definidos.
      * Se han superado las pruebas de penetración y escaneos de vulnerabilidades.
      * La documentación técnica y de usuario está completa.
      * El sistema está estabilizado y preparado para un despliegue en un entorno de pre-producción/piloto.

## Definición de Terminado (Definition of Done - DoD)

Para cada incremento de funcionalidad (historia de usuario o tarea de desarrollo), se aplicará la siguiente "Definición de Terminado":

* El código fuente está implementado y cumple con los estándares de codificación.
* Se han escrito pruebas unitarias y de integración, y todas pasan (cobertura mínima del 80%).
* La funcionalidad ha sido probada manualmente por el equipo de QA.
* El código ha pasado la revisión por pares (code review).
* La documentación técnica (comentarios en el código, docstrings, actualizaciones del DID si aplica) está actualizada.
* La funcionalidad se ha integrado en la rama principal (main/develop) y ha pasado la pipeline CI/CD.
* Los artefactos de despliegue están generados y listos.
* Las funcionalidades de seguridad relevantes para el incremento han sido verificadas.
* (Opcional, pero recomendable) La funcionalidad ha sido demostrada al Product Owner y aceptada.

## Equipo de Desarrollo (Roles Clave)

* **Product Owner:** Responsable de la visión del producto y la priorización del backlog.
* **Scrum Master:** Facilita el proceso Scrum y elimina impedimentos.
* **Arquitecto de Software:** Garantiza la coherencia arquitectónica y el cumplimiento de los patrones de diseño.
* **Desarrolladores Backend:** Responsables de la lógica de negocio, APIs, servicios de datos, integraciones.
* **Desarrolladores Frontend:** Responsables de la interfaz de usuario (UI/UX).
* **Ingenieros de QA/Automatización:** Creación y ejecución de pruebas, garantía de calidad.
* **Ingenieros DevOps:** Gestión de infraestructura, CI/CD, despliegue.

## Gestión de Cambios

Una vez que los requerinmientos sean "congelados" y aprobados, cualquier solicitud de cambio futuro DEBERÁ seguir un proceso formal de gestión de cambios que incluirá:

* **Solicitud de Cambio (RFC - Request for Change):** Documentación del cambio propuesto, incluyendo su justificación, impacto estimado (alcance, tiempo, costo) y riesgos.
* **Análisis de Impacto:** Evaluación detallada del impacto del cambio en los requisitos, la arquitectura, el diseño, la planificación y los recursos.
* **Aprobación Formal:** El cambio debe ser aprobado por los stakeholders clave del proyecto (ej., Product Owner, Gerente de Proyecto, Arquitecto).
* **Actualización de Documentación:** Todos los documentos afectados (incluyendo este DID) DEBEN ser actualizados con el número de versión y la fecha del cambio.

# ESPECIFICACIÓN DE LA ARQUITECTURA DETALLADA DE SOFTWARE

Este capítulo detalla la arquitectura de software de la Plataforma SCADA23, profundizando en la descripción de cada componente, su rol dentro del sistema y las tecnologías específicas utilizadas para su implementación en cada una de las fases de desarrollo.

## Arquitectura Detallada de Software - Fase 1 (TRL4)

La arquitectura de software de la Fase 1 es una implementación básica y monolítica, diseñada para demostrar la viabilidad de la comunicación y el control fundamental.

**Componentes Principales**

La aplicación SCADA de la Fase 1 se compone de un único programa Python que integra tres módulos lógicos principales:

* **Módulo de Comunicación Modbus TCP Cliente:** Responsable de establecer y mantener la conexión con el esclavo Modbus TCP y de enviar/recibir tramas de datos.
* **Módulo de Lógica de Procesamiento y Control:** Interpreta los datos recibidos, actualiza los valores internos y procesa las solicitudes de control del usuario.
* **Módulo de Interfaz de Usuario (Tkinter):** Proporciona la representación gráfica y permite la interacción del operador.

**Detalles de Implementación y Tecnologías**

* **Lenguaje de Programación:** Python 3.x
* **Librería Modbus TCP:** pymodbus o una implementación de cliente Modbus TCP más sencilla desarrollada ad-hoc si el alcance es muy limitado. La librería manejará la serialización/deserialización de la trama Modbus y la comunicación de sockets TCP.
* **Librería para la Interfaz Gráfica (GUI):** tkinter (parte de la librería estándar de Python).
  + **Elementos Gráficos:** Etiquetas (Label) para mostrar valores numéricos, botones (Button) para enviar comandos (ej., arrancar/parar), campos de entrada (Entry) para valores a escribir.
  + **Actualización de Datos:** Se utilizará after() de Tkinter para programar la lectura y actualización periódica de los datos desde el dispositivo Modbus TCP. Esto simulará un ciclo de refresco simple.
* **Estructura del Código:**
  + Un archivo principal (ej. scada\_trl4.py) que contendrá la lógica principal de la aplicación.
  + Funciones dedicadas para la lectura/escritura Modbus TCP.
  + Funciones para la construcción y actualización de la interfaz de usuario.
  + Variables globales o un objeto centralizado simple para almacenar los valores de los tags y su estado.
* **Flujo de Datos:**
  + - El módulo de comunicación Modbus TCP intenta establecer una conexión con el esclavo Modbus TCP configurado.
    - Cada cierto intervalo de tiempo (ej. 5 segundos), se llama a una función que realiza lecturas Modbus TCP de los registros definidos (ej. Holding Registers para analógicas, Coils para digitales).
    - Los datos leídos son procesados por el módulo de lógica (ej. conversión de tipo de dato, escalado básico si aplica).
    - El módulo de interfaz de usuario actualiza los elementos gráficos en pantalla con los nuevos valores.
    - Cuando el operador interactúa con un botón o campo de entrada, el módulo de interfaz llama a una función del módulo de lógica.
    - El módulo de lógica genera un comando Modbus TCP (escritura de Coils o Holding Registers) y lo envía a través del módulo de comunicación.
    - La respuesta del dispositivo (si la hay) se procesa para confirmar la operación.
* **Gestión de Errores:** Manejo básico de excepciones para errores de comunicación TCP (ej. conexión rechazada, timeout). Reintentos de conexión simples.
* **Configuración:** La dirección IP, puerto Modbus y la lista de registros a leer/escribir serán configurados directamente en el código fuente del script o en un archivo de configuración plano (ej. config.ini). No habrá una interfaz de usuario para la gestión de la configuración.

**Diagrama de Componentes de Software (Fase 1)**

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

## Arquitectura Detallada de Software - Fase 2 (TRL7)

La arquitectura de la Fase 2 es una evolución hacia un sistema distribuido y modular, aprovechando microservicios y bases de datos especializadas para garantizar escalabilidad, rendimiento y robustez.

**Componentes Principales y Su Interconexión**

* **Servicios de Adquisición de Datos (FastAPI Data Collectors):**
  + **Propósito:** Recolectar datos en tiempo real de los dispositivos de campo utilizando protocolos industriales.
  + **Tecnologías:** **FastAPI** (para APIs de alto rendimiento y asíncronas), librerías específicas para cada protocolo (pymodbus para Modbus TCP, librerías de terceros para PROFINET si existen en Python, asyncua para OPC-UA).
  + **Funcionamiento:** Cada colector se dedica a un protocolo o a un grupo de dispositivos. Expondrán una API interna para ser configurados por el Core SCADA y enviarán los datos procesados directamente a InfluxDB o a una cola de mensajes para su posterior ingesta.
* **Core SCADA (Django Backend):**
  + **Propósito:** El cerebro del sistema. Gestiona la lógica de negocio, usuarios, configuración, alarmas, eventos y expone la API principal.
  + **Tecnologías:** **Django** (framework web completo), **Django REST Framework (DRF)** (para construir APIs RESTful).
  + **Sub-Módulos (Ejemplos):**
    - **Módulo de Gestión de Modelos:** Define y gestiona los modelos de datos (Dispositivos, Tags, Alarmas, Usuarios, Roles, Vistas, etc.) almacenados en MySQL.
    - **Módulo de Autenticación y Autorización:** Maneja el inicio de sesión, la gestión de usuarios y la aplicación de permisos basados en roles (RBAC).
    - **Módulo de Lógica de Alarmas:** Procesa los datos en tiempo real (directamente desde InfluxDB o notificaciones de los colectores) contra umbrales y condiciones, generando y gestionando alarmas.
    - **Módulo de Cálculos y Variables Derivadas:** Ejecuta algoritmos sobre los datos para generar KPIs o nuevas variables (ej. promedios, eficiencias).
    - **Módulo de Informes:** Genera informes programados o bajo demanda.
    - **Módulo de Auditoría:** Registra todas las acciones críticas del usuario y del sistema.
* **Base de Datos Relacional (MySQL):**
  + **Propósito:** Almacenar datos estructurados y relacionales que cambian con menos frecuencia:
    - Configuración de la plataforma (dispositivos, tags, mapeos, umbrales de alarmas).
    - Información de usuarios, roles y permisos.
    - Logs de eventos y auditoría.
    - Estado de las alarmas (activas/reconocidas).
    - Metadatos del sistema.
  + **Tecnologías:** **MySQL Server**, gestionado por el ORM de Django.
* **Base de Datos de Series Temporales (InfluxDB):**
  + **Propósito:** Almacenar grandes volúmenes de datos de series temporales (mediciones de sensores, estados de actuadores) de manera eficiente.
  + **Tecnologías:** **InfluxDB**, con escritura optimizada a través de la API InfluxDB (posiblemente desde los colectores directamente o vía el Core SCADA). Lenguaje de consulta **Flux** para recuperación de datos de tendencias.
* **Interfaz de Usuario Web (HMI Frontend):**
  + **Propósito:** Proporcionar la experiencia de usuario para la supervisión, control y gestión del sistema SCADA.
  + **Tecnologías:** Se construirá como una aplicación web. Puede ser una Single Page Application (SPA) utilizando frameworks de JavaScript (React, Angular, Vue.js) que se comunica con el backend a través de las APIs RESTful, o bien, si se busca un desarrollo más integrado con Django y una menor complejidad de frontend, se podría utilizar el propio sistema de plantillas de Django (Django Templates) con librerías JavaScript ligeras (ej., Chart.js, HTMX para interactividad). Dada la ambición de un sistema "profesional", se inclinaría a un SPA moderno.
  + **Funcionalidades:** Paneles de control personalizables, sinópticos animados, gráficos de tendencias interactivas, tablas de datos en tiempo real, gestión de alarmas, configuración del sistema, gestión de usuarios.
* **Servidor Web / Proxy Inverso (Nginx/Apache):**
  + **Propósito:** Servir la aplicación web estática (archivos HTML, CSS, JS), actuar como puerta de enlace y balanceador de carga para los servicios de FastAPI y Django, y gestionar las conexiones HTTPS/TLS.
  + **Tecnologías:** **Nginx** o **Apache**. Configuración para redirección de tráfico, caching, compresión, y seguridad TLS.
* **Contenedorización y Orquestación (Docker/Docker Compose/Kubernetes):**
  + **Propósito:** Empaquetar cada servicio en contenedores ligeros y aislados, facilitando el despliegue, la gestión de dependencias y la escalabilidad.
  + **Tecnologías:** **Docker** (para la creación de imágenes de contenedores), **Docker Compose** (para la orquestación de múltiples contenedores en un solo host para desarrollo/despliegues pequeños), **Kubernetes** (para la orquestación a gran escala, alta disponibilidad y autoescalado en producción).

**Flujo de Datos y Control Detallado (Fase 2)**

1. **Adquisición de Datos:** Los **FastAPI Data Collectors** (uno por cada tipo de protocolo o grupo de dispositivos) inician la comunicación con los dispositivos de campo. Realizan lecturas periódicas o se suscriben a cambios de valores.
   * Los datos crudos son normalizados (escalado, unidades) y enriquecidos con metadatos del tag.
   * Los datos ya procesados se insertan directamente en **InfluxDB** como puntos de serie temporal.
2. **Procesamiento de Alarmas y Lógica de Negocio:**
   * El **Core SCADA (Django)** puede obtener datos de InfluxDB para evaluación de condiciones de alarma o cálculos, o los **FastAPI Data Collectors** pueden enviar notificaciones de eventos relevantes directamente al Core SCADA (ej. vía una API RESTful o un sistema de mensajería como RabbitMQ/Kafka si se necesita mayor resiliencia).
   * El módulo de alarmas del Core SCADA evalúa las condiciones (ej. valor > umbral), cambia el estado de las alarmas en **MySQL** y activa notificaciones.
   * El módulo de lógica de control procesa las reglas de automatización y envía comandos a los dispositivos a través de los **FastAPI Data Collectors**.
3. **Interfaz de Usuario (HMI Web):**
   * El navegador del cliente accede al **Servidor Web (Nginx)**, que sirve los archivos estáticos de la HMI.
   * La HMI realiza llamadas a las **APIs RESTful** del **Core SCADA (Django REST Framework)** y/o directamente a las APIs de los **FastAPI Data Collectors** para:
     + Obtener datos en tiempo real (WebSockets sobre FastAPI para push updates o polling rápido).
     + Consultar datos históricos de InfluxDB (vía API del Core SCADA).
     + Gestionar alarmas (reconocer, filtrar).
     + Configurar el sistema (dispositivos, tags, usuarios) – estas acciones persisten en MySQL.
     + Enviar comandos de control, que son validados por el Core SCADA (permisos, rangos) y luego enrutados a los **FastAPI Data Collectors** para su ejecución en el dispositivo.
4. **Almacenamiento:**
   * **InfluxDB:** Ingesta continua de puntos de datos de series temporales.
   * **MySQL:** Actualizaciones de estado de alarmas, inserción de eventos de auditoría, lectura/escritura de configuración del sistema y gestión de usuarios.
5. **Ciberseguridad:**
   * **Nginx:** Terminación TLS para HTTPS, WAF (Web Application Firewall) básico, rate limiting.
   * **Django:** Autenticación de usuarios, autorización basada en roles (RBAC), protección CSRF, protección XSS.
   * **FastAPI:** Autenticación de API keys/tokens, validación de esquemas de datos.
   * **Bases de Datos:** Credenciales seguras, segmentación de red, cifrado en reposo (opcional), auditoría de acceso.
   * **Protocolos Industriales:** Uso de OPC-UA con perfiles de seguridad, Modbus TCP Security si el dispositivo lo soporta, o tunelización segura (VPNs).

**Diagrama de Componentes de Software (Fase 2)**

**Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.**

**Justificación de la Elección de Tecnologías (Fase 2)**

* **Python:** Lenguaje de programación versátil, con una vasta colección de librerías para desarrollo web, comunicación industrial y análisis de datos. Su legibilidad y rapidez de desarrollo lo hacen ideal.
* **Django:** Framework web maduro y robusto, con un ORM potente, un sistema de autenticación/autorización integrado, y una comunidad activa. Excelente para la lógica de negocio compleja, la gestión de usuarios y la configuración del sistema.
* **FastAPI:** Framework web moderno y de alto rendimiento para construir APIs. Su naturaleza asíncrona lo hace ideal para la adquisición de datos en tiempo real y la gestión de múltiples conexiones de forma eficiente.
* **Django REST Framework (DRF):** Extensión de Django que facilita enormemente la creación de APIs RESTful. Proporciona serializadores, autenticación y permisos listos para usar.
* **MySQL:** Base de datos relacional de código abierto ampliamente utilizada, ideal para datos estructurados que requieren integridad transaccional, como la configuración, los eventos y los usuarios.
* **InfluxDB:** Base de datos NoSQL de series temporales, optimizada para la ingesta, almacenamiento y consulta eficiente de grandes volúmenes de datos con marca de tiempo. Es fundamental para el rendimiento del SCADA en la gestión de históricos de variables.
* **Tkinter (Fase 1):** Selección pragmática para la Fase 1. Es una librería estándar de Python, lo que simplifica la instalación y el entorno de desarrollo, perfecta para una demostración de concepto rápida y sin dependencias externas significativas.
* **Docker/Kubernetes:** Permiten empaquetar la aplicación y sus dependencias en contenedores, asegurando que el software se ejecute de la misma manera en cualquier entorno. Facilitan el escalado horizontal, la alta disponibilidad y la gestión del ciclo de vida del software en producción.

Esta arquitectura detallada y la justificación de las tecnologías reflejan un diseño consciente de las necesidades de un sistema SCADA moderno, que equilibra el rendimiento, la escalabilidad, la fiabilidad y la seguridad para el exigente sector del agua potable.

# ARQUITECTURA HARDWARE

Este capítulo describe los requisitos de hardware y la infraestructura de red necesaria para el despliegue y operación de la Plataforma SCADA23. Se diferenciarán las necesidades entre la Fase 1 (TRL4), orientada a la demostración de concepto, y la Fase 2 (TRL7), diseñada para un entorno de producción robusto y escalable.

## Arquitectura Hardware e Infraestructuras Asociadas - Fase 1 (TRL4)

La Fase 1 se enfoca en la validación de la comunicación básica Modbus TCP y la funcionalidad SCADA elemental. Por lo tanto, los requisitos de hardware e infraestructura son mínimos y están pensados para un entorno de laboratorio o demostración.

**Componentes de Hardware (Fase 1)**

* **Servidor SCADA (Equipo Principal):**
  + **Tipo:** Un ordenador personal (portátil o de escritorio) estándar.
  + **CPU:** Procesador de doble núcleo (ej. Intel Core i3 o similar).
  + **RAM:** 4 GB de RAM.
  + **Almacenamiento:** Disco duro HDD o SSD de 128 GB o superior (suficiente para el sistema operativo y el software del SCADA).
  + **Sistema Operativo:** Windows 10/11, macOS, o cualquier distribución de Linux compatible con Python 3 y Tkinter.
  + **Puertos:** Al menos un puerto Ethernet disponible.
* **Dispositivo de Campo (Simulado/Real):**
  + **Simulador Modbus TCP:** Puede ser software ejecutándose en el mismo equipo del SCADA o en otro PC. No requiere hardware específico.
  + **PLC/RTU real (Opcional):** Un Controlador Lógico Programable (PLC) o Unidad Terminal Remota (RTU) con soporte para comunicación Modbus TCP (ej. Siemens S7-1200, Wago PFC, Schneider Modicon).
    - **Conectividad:** Puerto Ethernet para Modbus TCP.

**Infraestructura de Red (Fase 1)**

* **Topología:** Red local básica (LAN).
* **Componentes:**
  + **Switch Ethernet:** Un switch de red de 4-8 puertos.
  + **Cables Ethernet:** Cableado UTP Cat5e/Cat6 para conectar el servidor SCADA y el dispositivo de campo.
* **Conectividad:** La comunicación se realizará exclusivamente a través de la red local. No se requiere acceso a internet ni configuraciones de red complejas (VPNs, firewalls avanzados).
* **Consideraciones de Seguridad:** En esta fase, la seguridad de la red se basará en el aislamiento físico del entorno de demostración. No se implementarán medidas de ciberseguridad a nivel de red, más allá de la configuración básica de un switch.

**Diagrama de Infraestructura (Fase 1)**

## Arquitectura Hardware e Infraestructuras Asociadas - Fase 2 (TRL7)

La Fase 2 exige una infraestructura robusta y escalable para soportar la operación crítica de un sistema SCADA completo, incluyendo alta disponibilidad, rendimiento y ciberseguridad.

**Componentes de Hardware (Fase 2)**

* **Servidores SCADA (Físicos o Virtuales):**
  + **Tipo:** Se recomienda un clúster de servidores o máquinas virtuales de alto rendimiento. Para entornos de producción, se sugiere la virtualización para optimizar recursos y facilitar la gestión.
  + **CPU:** Múltiples núcleos de alto rendimiento (ej. Intel Xeon o AMD EPYC, 8-16 núcleos por servidor como punto de partida).
  + **RAM:** Mínimo 32 GB de RAM por servidor, escalable a 64 GB o más, especialmente para bases de datos y módulos de adquisición de datos con alta concurrencia.
  + **Almacenamiento:**
    - **SSD NVMe:** Imprescindible para las bases de datos (MySQL e InfluxDB) debido a su alta velocidad de lectura/escritura y bajas latencias. Un mínimo de 500 GB a 1 TB por servidor, dependiendo del volumen de datos históricos esperado.
    - **Configuración de RAID:** Para redundancia y rendimiento del almacenamiento.
  + **Tarjetas de Red (NIC):** Múltiples tarjetas de red Gigabit Ethernet (o 10 Gigabit Ethernet para grandes despliegues) para separar el tráfico (ej. tráfico OT, tráfico IT, replicación de BD).
  + **Sistema Operativo:** Distribución de Linux (ej. Ubuntu Server, CentOS, Debian) optimizada para servidores.
* **Dispositivos de Campo (PLCs, RTUs, Sensores, Actuadores):**
  + **PLCs de Gama Industrial:** Compatibles con Modbus TCP, PROFINET, OPC-UA (ej. Siemens S7-1500, Rockwell ControlLogix, Beckhoff TwinCAT).
  + **RTUs:** Para telecontrol en ubicaciones remotas con conectividad robusta.
  + **Sensores y Actuadores Inteligentes:** Con capacidad de comunicación directa o a través de módulos I/O que soporten los protocolos definidos.
* **Hardware de Red y Ciberseguridad:**
  + **Firewalls Industriales (OT):** Para la segmentación entre la red OT y la red IT, y para proteger los activos críticos de control.
  + **Routers y Switches Gestionables:** Con capacidades de VLAN, QoS (Quality of Service) y seguridad de puerto para la segmentación de red.
  + **Hardware de Redundancia:** Load balancers (hardware o software como Nginx, HAProxy) para distribuir la carga entre los servidores SCADA y asegurar la alta disponibilidad.
  + **Servidores de Copia de Seguridad:** Para realizar y almacenar las copias de seguridad de las bases de datos y configuraciones.
  + **Hardware para VPNs:** Servidores VPN o firewalls con capacidades VPN para acceso remoto seguro.

**Infraestructura de Red (Fase 2)**

* **Segmentación de Red (OT/IT):** Crucial para la ciberseguridad.
  + **Red OT (Operational Technology):** Red aislada para la comunicación entre los servidores SCADA (módulos de adquisición) y los dispositivos de campo (PLCs, RTUs). Tráfico Modbus TCP, PROFINET, OPC-UA. Controlada por firewalls industriales.
  + **Red IT (Information Technology):** Red para la interfaz de usuario web, bases de datos, y comunicación con otros sistemas corporativos (ERP, GMAO). Tráfico HTTPS, SQL.
  + **DMZ (Zona Desmilitarizada):** Posiblemente una DMZ para los servidores web y APIs expuestas, protegida por firewalls.
* **Conectividad WAN (para Telecontrol):**
  + **VPNs:** Implementación de Redes Privadas Virtuales (VPN) seguras (IPsec, OpenVPN, WireGuard) para la comunicación con RTUs y PLCs en ubicaciones remotas sobre redes públicas (3G/4G/5G, ADSL, Fibra).
  + **Redes Industriales:** Consideración de soluciones de redes inalámbricas industriales (ej. LoRaWAN, NB-IoT) o fibra óptica para grandes distancias y entornos adversos.
* **Alta Disponibilidad de Red:**
  + **Redundancia de Caminos:** Implementación de rutas de red redundantes y protocolos de enrutamiento tolerantes a fallos (ej. STP, VRRP).
  + **Fuentes de Alimentación Ininterrumpida (UPS):** Para proteger los equipos de red y servidores de cortes de energía.
* **Sincronización Horaria (NTP):** Todos los equipos en la red (servidores, PLCs, RTUs) deben sincronizarse con un servidor NTP central para asegurar la coherencia de las marcas de tiempo en los datos.

**Diagrama de Infraestructura (Fase 2)**

**Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.**

**Justificación de la Elección de Hardware e Infraestructura (Fase 2)**

* **Servidores de Alto Rendimiento:** Necesarios para manejar la carga de trabajo de múltiples servicios, procesamiento de datos en tiempo real, gestión de bases de datos y soporte a múltiples usuarios.
* **Almacenamiento SSD NVMe:** Crítico para InfluxDB debido a la alta tasa de ingesta de datos de series temporales y para MySQL para asegurar un rendimiento óptimo en las operaciones transaccionales.
* **Redundancia (Hardware y Software):** Esencial para sistemas SCADA que operan en entornos críticos. La redundancia a nivel de servidores, bases de datos y red minimiza el tiempo de inactividad ante fallos.
* **Segmentación de Red (OT/IT):** Medida fundamental de ciberseguridad. Aísla las redes de control de las redes corporativas, reduciendo la superficie de ataque y el impacto de posibles incidentes. Los firewalls industriales son clave aquí.
* **Conectividad WAN Segura (VPN):** Permite la integración de dispositivos remotos de forma segura, protegiendo los datos en tránsito sobre redes públicas.
* **Sincronización Horaria (NTP):** Imprescindible para la correcta correlación de eventos y datos en un sistema distribuido, especialmente para el análisis de tendencias y la trazabilidad.

Esta arquitectura hardware e infraestructura de red avanzada garantizará que la Plataforma SCADA23 en su versión TRL7 sea una solución fiable, de alto rendimiento, segura y escalable para las exigencias del sector del agua potable.

# CIBERSEGURIDAD DEL SISTEMA

La ciberseguridad es un pilar fundamental en el diseño y desarrollo de la Plataforma SCADA23, especialmente dada la criticidad de las infraestructuras que supervisa y controla, como las del sector del agua potable. Este capítulo detalla el enfoque de ciberseguridad adoptado, las medidas implementadas y las consideraciones clave en cada fase del proyecto, desde la validación de concepto hasta el despliegue en producción.

**Introducción a la Ciberseguridad en SCADA**

Los sistemas SCADA, tradicionalmente aislados, se han vuelto cada vez más interconectados, exponiéndolos a un mayor riesgo de ataques cibernéticos. Un incidente de ciberseguridad en un sistema SCADA puede tener consecuencias devastadoras, incluyendo interrupciones de servicio, daños a equipos, impacto ambiental y riesgo para la salud pública. Por ello, SCADA23 incorpora un enfoque de "Ciberseguridad desde el Diseño" (Security by Design), integrando la protección en cada etapa del ciclo de vida del desarrollo.

Los objetivos principales de la ciberseguridad en SCADA23 son:

* **Confidencialidad:** Proteger la información sensible de accesos no autorizados.
* **Integridad:** Asegurar que los datos y los comandos no sean alterados de forma no autorizada o accidental.
* **Disponibilidad:** Garantizar que el sistema y los recursos estén accesibles y operativos cuando se necesiten.
* **Autenticidad:** Verificar la identidad de usuarios, dispositivos y sistemas que interactúan con SCADA23.
* **Trazabilidad:** Registrar todas las acciones relevantes para auditoría y análisis forense.

**Enfoque de Ciberseguridad - Fase 1 (TRL4)**

En la Fase 1, el enfoque de ciberseguridad es pragmático y limitado, dada la naturaleza de demostración de concepto del prototipo. La principal medida de seguridad es el **aislamiento del entorno**.

**Medidas de Ciberseguridad Implementadas (Fase 1)**

* **Aislamiento de Red:** El sistema SCADA TRL4 y los dispositivos de campo (simulados o reales) operarán en una red local (LAN) físicamente aislada de cualquier red externa o de internet. Esta es la medida más crítica para mitigar riesgos en esta fase.
* **Dependencias Mínimas:** El software de la Fase 1 (scripts Python, aplicación Tkinter) tendrá un número muy limitado de dependencias externas, reduciendo la superficie de ataque potencial asociada a librerías de terceros.
* **Puertos Estándar:** La comunicación Modbus TCP se realizará sobre el puerto estándar (502/TCP). No se implementarán cifrados ni autenticaciones a nivel de protocolo Modbus TCP.
* **Conciencia de Seguridad en el Desarrollo:** Aunque no se implementarán controles de seguridad complejos, los desarrolladores seguirán prácticas básicas de codificación segura para evitar vulnerabilidades obvias (ej., no almacenar credenciales en texto plano, validación mínima de entradas si fuera necesario).
* **No Autenticación/Autorización de Usuarios:** La aplicación Tkinter no tendrá un sistema de autenticación de usuarios. Se asume que el acceso físico al ordenador que ejecuta el SCADA está controlado.

**Riesgos Mitigados (Fase 1)**

* Ataques desde internet o redes externas (gracias al aislamiento).
* Vulnerabilidades graves en librerías de terceros (por la mínima dependencia).

**Riesgos no Mitigados (Fase 1)**

* Acceso no autorizado al sistema desde la red local aislada.
* Manipulación de datos o comandos por un usuario interno no autorizado.
* Ataques de denegación de servicio internos.
* Vulnerabilidades inherentes al protocolo Modbus TCP (falta de autenticación, cifrado).

**Enfoque de Ciberseguridad - Fase 2 (TRL7)**

La Fase 2 representa el desarrollo de una plataforma SCADA de grado industrial. Aquí, la ciberseguridad es una consideración integral y se aborda en múltiples capas y niveles de la arquitectura.

**Medidas de Ciberseguridad Implementadas (Fase 2)**

La estrategia de ciberseguridad se basa en un modelo de "defensa en profundidad" (Defense-in-Depth) y abarca los siguientes dominios:

**Seguridad de la Red**

* **Segmentación de Red (OT/IT/DMZ):**
  + **Red OT (Industrial/Control):** Aislada lógicamente de la red IT. Solo los servicios de adquisición de datos del SCADA tendrán acceso a esta red. Se utilizarán firewalls industriales (ej. para Modbus TCP, PROFINET, OPC-UA).
  + **Red IT (Corporativa):** Para la interfaz de usuario web, servicios de gestión y bases de datos.
  + **Zona Desmilitarizada (DMZ):** Si es necesario, se implementará una DMZ para los componentes expuestos a la red IT o internet (servidor web, load balancer, API Gateway), con reglas estrictas de firewall entre la DMZ, la red IT y la red OT.
* **Firewalling y Listas de Control de Acceso (ACLs):** Reglas estrictas en los firewalls para permitir solo el tráfico necesario entre zonas y entre componentes.
* **VPN para Acceso Remoto:** Todo el acceso remoto al sistema SCADA (ej., para administradores, operadores remotos, dispositivos RTU) se realizará exclusivamente a través de conexiones VPN cifradas y autenticadas.
* **Monitorización de Tráfico de Red:** Uso de herramientas para detectar anomalías o tráfico malicioso (ej. IDS/IPS a nivel de red).
* **Sincronización Horaria Segura (NTP):** Asegurar que todos los sistemas utilicen NTP seguro (ej. NTPsec) para la sincronización horaria, crucial para la correlación de eventos de seguridad.

**Seguridad del Host y del Sistema Operativo**

* **Hardening del Sistema Operativo:** Configuración segura de los servidores Linux (deshabilitar servicios innecesarios, políticas de contraseñas robustas, uso de SELinux/AppArmor, gestión de parches).
* **Gestión de Vulnerabilidades:** Escaneo regular de vulnerabilidades en los sistemas operativos y librerías. Proceso definido para la aplicación de parches y actualizaciones de seguridad.
* **Antimalware y Whitelisting:** Implementación de soluciones antimalware o, preferiblemente, whitelisting de aplicaciones en servidores críticos para permitir solo la ejecución de software autorizado.
* **Logging y Monitorización:** Configuración de logging detallado de eventos del sistema operativo y aplicaciones. Integración con un sistema de gestión de eventos e información de seguridad (SIEM) para la correlación y análisis de logs.

**Seguridad de la Aplicación y de los Datos**

* **Autenticación Robusta:**
  + **Contraseñas:** Requisitos de complejidad, longitud mínima, rotación periódica. Almacenamiento de hashes de contraseñas seguros (ej., Argon2, Bcrypt).
  + **Bloqueo de Cuentas:** Tras múltiples intentos fallidos de inicio de sesión.
  + **Autenticación de Dos Factores (2FA):** Opcional, pero recomendado para usuarios privilegiados (administradores).
  + **SSO (Single Sign-On):** Posible integración con soluciones de SSO existentes en la organización (ej. LDAP, Active Directory) si se requiere.
* **Autorización Basada en Roles (RBAC):**
  + Sistema de roles y permisos granulares (ej., Administrador, Operador, Supervisor, Solo Lectura).
  + Control de acceso a funcionalidades de la interfaz de usuario, operaciones de control, configuración y acceso a datos específicos.
* **Cifrado de Comunicaciones:**
  + **HTTPS/TLS:** Todas las comunicaciones entre el cliente web y el backend (Django, FastAPI) se realizarán a través de HTTPS/TLS, utilizando certificados digitales válidos.
  + **Cifrado de Protocolos Industriales:** Utilización de perfiles de seguridad en OPC-UA (OPC UA Security), y evaluación del uso de Modbus TCP Security si el hardware de campo lo soporta. En caso contrario, se confiará en la VPN o el aislamiento de la red OT.
* **Integridad de Datos:**
  + Validación de entradas y salidas para prevenir ataques como inyección SQL, XSS, CSRF (protecciones nativas de Django).
  + Uso de funciones hash o firmas digitales para asegurar la integridad de configuraciones críticas o firmware.
* **Gestión Segura de Credenciales:** Las credenciales de acceso a bases de datos, APIs y otros servicios se gestionarán de forma segura (variables de entorno, gestores de secretos, nunca en texto plano en el código fuente).
* **Auditoría y Trazabilidad:**
  + Registro detallado de todas las acciones de los usuarios (inicios de sesión, cambios de configuración, operaciones de control) con marcas de tiempo precisas, usuario y detalle de la acción.
  + Almacenamiento inmutable de logs en las bases de datos (MySQL) y sistemas de logging centralizados.
* **Gestión de Sesiones:** Sesiones seguras (HttpOnly, Secure cookies), timeouts de sesión configurables, invalidación de sesiones al cierre de sesión y regeneración de IDs de sesión.
* **Manejo Seguro de Errores:** Evitar la exposición de información sensible en mensajes de error.
* **Protección contra Ataques DoS/DDoS:** Implementación de limitación de tasas (rate limiting) en las APIs y configuración de Nginx/Apache para mitigar ataques de denegación de servicio.

**Seguridad de las Bases de Datos**

* **Autenticación y Autorización:** Cuentas de usuario de base de datos con privilegios mínimos (Principio del Menor Privilegio).
* **Cifrado en Reposo (opcional pero recomendado):** Cifrado de los datos almacenados en los discos donde residen las bases de datos (ej. cifrado de disco completo, o funciones de cifrado de la propia BD).
* **Auditoría de Acceso a Datos:** Configuración de la base de datos para registrar accesos y modificaciones a datos sensibles.
* **Backups Seguros:** Copias de seguridad regulares, cifradas y almacenadas en ubicaciones seguras, con un plan de recuperación de desastres probado.

**Cumplimiento de Estándares (Fase 2)**

Aunque no se certificará formalmente en esta etapa, el diseño de ciberseguridad de SCADA23 se alineará con las mejores prácticas y recomendaciones de estándares relevantes para la seguridad industrial y SCADA, tales como:

* **IEC 62443:** Serie de estándares internacionales para la ciberseguridad de sistemas de automatización y control industrial (IACS). SCADA23 buscará satisfacer los requisitos clave de seguridad para los componentes de software.
* **NIST Cybersecurity Framework:** Marco voluntario para gestionar y reducir el riesgo de ciberseguridad para infraestructuras críticas.
* **ISO 27001:** Aunque es un estándar de gestión de seguridad de la información, sus principios se aplicarán en la gestión de la ciberseguridad del proyecto.

**Proceso de Desarrollo Seguro (SDLC)**

Para garantizar la ciberseguridad, se integrarán actividades de seguridad en cada etapa del ciclo de vida de desarrollo del software (SDLC) de la Fase 2:

* **Requisitos:** Definición de requisitos de seguridad claros desde el inicio.
* **Diseño:** Modelado de amenazas (Threat Modeling) para identificar y mitigar posibles vulnerabilidades en la arquitectura.
* **Implementación:** Revisiones de código de seguridad, uso de herramientas de análisis de código estático (SAST).
* **Pruebas:** Pruebas de penetración (Penetration Testing), escaneos de vulnerabilidades dinámicas (DAST), pruebas de seguridad funcional.
* **Despliegue:** Hardening del entorno de producción, configuración segura de los componentes.
* **Operación y Mantenimiento:** Monitorización continua, gestión de parches, respuesta a incidentes.

**Respuesta a Incidentes de Ciberseguridad**

Se definirá un plan básico de respuesta a incidentes para la Fase 2, que incluirá:

* Detección y análisis de incidentes (basado en logs y alertas).
* Contención y erradicación del incidente.
* Recuperación del sistema.
* Análisis post-incidente y lecciones aprendidas.

En resumen, la ciberseguridad no es un añadido, sino un componente intrínseco de la Plataforma SCADA23 en su desarrollo de la Fase 2 (TRL7), asegurando un sistema robusto y confiable para la protección de infraestructuras críticas.

# PRUEBAS DEL SISTEMA

Las pruebas son una fase crítica en el ciclo de vida de desarrollo de la Plataforma SCADA23, asegurando que el sistema cumpla con los requisitos especificados, funcione de manera fiable y sea seguro. Este capítulo describe la estrategia de pruebas, los tipos de pruebas a realizar, los entornos de prueba y las responsabilidades asociadas para cada una de las fases del proyecto (TRL4 y TRL7).

**Estrategia General de Pruebas**

La estrategia de pruebas de SCADA23 se basa en un enfoque iterativo e incremental, alineado con la metodología por fases del proyecto. Se buscará una combinación de pruebas manuales y automatizadas, comenzando con pruebas de componentes individuales y progresando hacia la validación del sistema completo. La ciberseguridad se integrará en todas las etapas de las pruebas.

**Principios de Prueba:**

* **Verificación y Validación:** Asegurar que el sistema está siendo construido correctamente (verificación) y que se está construyendo el sistema correcto (validación de requisitos).
* **Cobertura:** Intentar cubrir la mayor cantidad posible de funcionalidades y escenarios, incluyendo casos de uso normales, casos límite y escenarios de error.
* **Automatización:** Maximizar la automatización de pruebas siempre que sea posible para mejorar la eficiencia, repetibilidad y consistencia de las pruebas, especialmente en la Fase 2.
* **Seguridad:** Integrar pruebas de seguridad en todas las fases para identificar y mitigar vulnerabilidades.
* **Documentación:** Registrar los planes de prueba, casos de prueba, resultados y defectos encontrados.

**Plan de Pruebas - Fase 1 (TRL4)**

La Fase 1 se centra en la verificación de la conectividad básica y la funcionalidad mínima viable. Las pruebas serán de menor escala y mayormente manuales.

**Objetivos de Pruebas (Fase 1)**

* Verificar la correcta comunicación Modbus TCP (lectura/escritura) entre el SCADA y el dispositivo de campo.
* Confirmar que la interfaz Tkinter muestra correctamente los datos adquiridos.
* Asegurar que los comandos de control se envían y ejecutan correctamente en el dispositivo.
* Validar la estabilidad básica de la comunicación durante periodos cortos.

**Tipos de Pruebas (Fase 1)**

* **Pruebas Unitarias (Básicas):** Pruebas de funciones individuales de Python (ej., parseo de tramas Modbus, lógica de escalado básica).
* **Pruebas de Integración:** Verificación de la comunicación entre el módulo Modbus del SCADA y el dispositivo de campo/simulador.
* **Pruebas Funcionales:**
  + **Lectura de Datos:** Verificar que los valores leídos de Holding Registers, Input Registers, Coils y Discrete Inputs se corresponden con los valores esperados en el dispositivo.
  + **Escritura de Comandos:** Verificar que al activar un control en Tkinter (ej., un botón), el estado de un Coil o el valor de un Holding Register en el dispositivo cambia correctamente.
  + **Actualización de Interfaz:** Confirmar que la interfaz se refresca con los nuevos datos dentro del tiempo esperado.
* **Pruebas de Usabilidad (Demostración):** Verificación de que la interfaz es intuitiva para realizar las operaciones básicas para las que fue diseñada.
* **Pruebas de Resistencia (Básicas):** Realizar pruebas de conexión/desconexión del dispositivo para ver el comportamiento del SCADA.

**Entorno de Pruebas (Fase 1)**

* **Hardware:** El mismo ordenador que ejecuta el SCADA TRL4 y un simulador Modbus TCP o un PLC/RTU real (si está disponible para esta fase).
* **Red:** Una red LAN aislada y básica (Ethernet).
* **Herramientas:** Consola de Python, un editor de código, posiblemente un analizador de tráfico de red básico (Wireshark) para verificar las tramas Modbus TCP.

**Responsabilidades (Fase 1)**

* **Desarrollador Principal:** Es el responsable de diseñar, ejecutar y documentar las pruebas en esta fase, dada la simplicidad del prototipo.

**Criterios de Aceptación (Fase 1)**

* La aplicación SCADA se conecta y se mantiene conectada al dispositivo Modbus TCP simulado/real durante la demostración.
* Todos los puntos de datos configurados se leen y se muestran correctamente en la interfaz de Tkinter.
* Todos los puntos de control configurados pueden ser escritos y su estado se refleja en el dispositivo.
* No hay errores críticos que bloqueen la aplicación durante la demostración.

**Plan de Pruebas - Fase 2 (TRL7)**

La Fase 2 implica un plan de pruebas exhaustivo y multicapa, dado el tamaño, la complejidad y la criticidad de la Plataforma SCADA completa. Se utilizarán herramientas y metodologías más avanzadas.

**Objetivos de Pruebas (Fase 2)**

* Validar todos los requisitos funcionales y no funcionales definidos en el Capítulo 2.
* Asegurar la robustez, escalabilidad, rendimiento y alta disponibilidad del sistema.
* Identificar y mitigar todas las vulnerabilidades de seguridad conocidas.
* Verificar la integración de todos los componentes de software (colectores, backend, bases de datos, frontend).
* Confirmar la compatibilidad con múltiples protocolos industriales (Modbus TCP, PROFINET, OPC-UA).
* Validar la usabilidad y la experiencia de usuario de la interfaz web HMI.
* Probar la capacidad de recuperación ante fallos del sistema.

**Tipos de Pruebas (Fase 2)**

* **Pruebas Unitarias:** Para cada función y método de los módulos de FastAPI, Django y el frontend. Se automatizarán con frameworks como unittest o pytest en Python y librerías de testing en JavaScript (ej. Jest, React Testing Library).
* **Pruebas de Integración:**
  + **Entre Componentes del Backend:** Verificación de la comunicación entre servicios (FastAPI con Django, Django con MySQL/InfluxDB).
  + **Backend-Frontend:** Validación de las APIs RESTful y WebSockets.
  + **SCADA-Dispositivo de Campo:** Pruebas exhaustivas de conectividad y comunicación con PLCs/RTUs reales utilizando Modbus TCP, PROFINET y OPC-UA.
* **Pruebas Funcionales:**
  + **Casos de Uso Completos:** Pruebas de los flujos de trabajo de los operadores (ej. configuración de un nuevo dispositivo, gestión de alarmas, control de una bomba).
  + **Gestión de Alarmas:** Probar la activación, notificación, reconocimiento y historización de alarmas bajo diversas condiciones.
  + **Control y Automatización:** Validación de la lógica de control, secuencias y recetas.
  + **Gestión de Usuarios y Permisos:** Verificar que el RBAC funciona según lo diseñado y que los usuarios solo tienen acceso a las funcionalidades y datos autorizados.
  + **Generación de Informes:** Probar la correcta generación de informes con datos precisos.
* **Pruebas No Funcionales:**
  + **Pruebas de Rendimiento y Carga:** Medir los tiempos de respuesta del sistema bajo diferentes cargas (ej. número de puntos de datos, frecuencia de adquisición, número de usuarios concurrentes) para asegurar que cumple con los RNF. Herramientas como Locust o JMeter.
  + **Pruebas de Estrés:** Llevar el sistema al límite de sus capacidades para identificar cuellos de botella y puntos de ruptura.
  + **Pruebas de Escalabilidad:** Verificar que el sistema puede escalar añadiendo recursos (ej. nuevos contenedores Docker).
  + **Pruebas de Fiabilidad y Recuperación:**
    - Simulación de fallos de componentes (ej. caída de un servidor, desconexión de red, fallo de una base de datos).
    - Verificación de la capacidad de auto-recuperación y la continuidad del servicio (ej. failover de bases de datos, reconexión automática de drivers).
    - Pruebas de respaldo y restauración de datos.
  + **Pruebas de Usabilidad:** Evaluación de la interfaz de usuario por parte de usuarios finales para asegurar su intuición, eficiencia y ergonomía.
  + **Pruebas de Compatibilidad:** Asegurar la compatibilidad con diferentes navegadores web.
  + **Pruebas de Portabilidad:** Verificar el despliegue en diferentes entornos de servidor (ej. máquinas virtuales, diferentes proveedores de cloud).
* **Pruebas de Ciberseguridad:**
  + **Escaneo de Vulnerabilidades:** Uso de herramientas automatizadas para identificar vulnerabilidades conocidas en código, librerías y dependencias (SAST/DAST).
  + **Pruebas de Penetración (Pen Testing):** Realizadas por equipos de seguridad internos o externos para simular ataques y descubrir vulnerabilidades.
  + **Pruebas de Configuración Segura:** Verificar que el hardening del sistema operativo, bases de datos y servidores web se ha aplicado correctamente.
  + **Pruebas de Fuzzing:** Para los módulos de comunicación industrial, enviar datos malformados para probar la robustez ante entradas inesperadas.
  + **Pruebas de Autenticación/Autorización:** Intentos de bypass, elevación de privilegios, ataques de fuerza bruta.

**Entorno de Pruebas (Fase 2)**

* **Entorno de Desarrollo/Integración Continua:** Servidores para ejecución de pruebas unitarias y de integración automatizadas (parte del pipeline CI/CD).
* **Entorno de Staging/Pre-Producción:** Un entorno espejo de producción, con hardware y software similares, para pruebas de rendimiento, estrés, fiabilidad y seguridad.
* **Laboratorio de Pruebas Industriales:** Un laboratorio con PLCs, RTUs y sensores reales para probar la integración con los protocolos industriales en condiciones controladas.
* **Herramientas:**
  + **Gestión de Pruebas:** Jira, TestLink, Azure DevOps, etc.
  + **Automatización de Pruebas:** Pytest, Selenium (para web), Cypress/Playwright, Locust/JMeter.
  + **Análisis de Código Estático (SAST):** SonarQube.
  + **Escáneres de Vulnerabilidades:** Nessus, OpenVAS.
  + **Pen Testing Tools:** Kali Linux tools, Nmap, Metasploit, Burp Suite.
  + **Monitorización:** Prometheus, Grafana (para métricas de rendimiento).

**Responsabilidades (Fase 2)**

* **Equipo de QA (Quality Assurance):** Diseño de planes de prueba detallados, creación y ejecución de casos de prueba, gestión de defectos, informes de calidad.
* **Desarrolladores:** Pruebas unitarias, pruebas de integración a nivel de componente, corrección de defectos.
* **Especialistas en Ciberseguridad:** Realización de pruebas de penetración, auditorías de configuración, asesoramiento en seguridad.
* **Equipo de DevOps:** Gestión de los entornos de prueba, herramientas de CI/CD, monitorización.
* **Usuarios Finales/Expertos en Dominio:** Pruebas de aceptación de usuario (UAT) para validar la idoneidad funcional y de usabilidad.

**Criterios de Aceptación (Fase 2)**

* Todos los requisitos funcionales y no funcionales críticos deben ser validados.
* Tasa de defectos graves/críticos por debajo de un umbral definido.
* Cumplimiento de los umbrales de rendimiento (tiempos de respuesta, latencia, capacidad de procesamiento).
* Informe de pruebas de seguridad que no identifique vulnerabilidades de riesgo crítico/alto.
* Demostración de la capacidad de recuperación del sistema ante fallos.
* Aprobación por parte de los usuarios finales en las UAT.
* Documentación de pruebas completa y actualizada.

La implementación rigurosa de este plan de pruebas es fundamental para entregar una Plataforma SCADA23 de alta calidad, fiable, segura y lista para operar en entornos industriales críticos.

# IMPLEMENTACIÓN Y DESPLIEGUE

Este capítulo describe las metodologías, herramientas y procedimientos para la implementación y el despliegue de la Plataforma SCADA23. Se abordarán las particularidades de cada fase, desde el entorno de desarrollo y pruebas de concepto de la Fase 1 (TRL4) hasta el despliegue en un entorno de producción robusto y operativo para la Fase 2 (TRL7).

## Metodología de despliegue

**Implementación y Despliegue - Fase 1 (TRL4)**

La implementación y el despliegue en la Fase 1 son de naturaleza sencilla y manual, enfocados en la rapidez y la validación del concepto.

**Metodología de Implementación (Fase 1)**

* **Desarrollo Rápido de Prototipos:** Se utilizará un enfoque ágil y directo. El código se desarrollará de forma iterativa, con ciclos cortos de codificación, pruebas y depuración.
* **Minimalismo:** Solo se implementarán las funcionalidades esenciales para demostrar la viabilidad de la comunicación Modbus TCP y la interfaz básica.

**Herramientas de Implementación (Fase 1)**

* **Entorno de Desarrollo Integrado (IDE):** Cualquier IDE básico para Python (ej., VS Code, PyCharm Community Edition, Sublime Text) o incluso un editor de texto simple.
* **Consola de Línea de Comandos:** Para ejecutar los scripts Python y observar la salida.
* **Librerías Python:** Instalación manual de pymodbus (si se usa) mediante pip. tkinter viene por defecto con la mayoría de las instalaciones de Python.

**Proceso de Despliegue (Fase 1)**

El despliegue en esta fase es un proceso manual y directo:

1. **Preparación del Entorno:**
   * Instalar Python 3.x en el ordenador principal (servidor SCADA).
   * Instalar pymodbus (si no se usa una implementación ad-hoc) y cualquier otra dependencia menor a través de pip.
   * Asegurar que el entorno de red esté configurado (conexión Ethernet entre el PC y el dispositivo de campo/simulador).
2. **Copia de Código:** Copiar los scripts Python de la aplicación SCADA TRL4 al ordenador principal.
3. **Configuración Básica:** Si hay un archivo config.ini o similar, ajustar la dirección IP y el puerto Modbus TCP del dispositivo de campo. Si la configuración está embebida en el código, modificarla directamente.
4. **Ejecución:** Iniciar la aplicación SCADA ejecutando el script principal de Python desde la línea de comandos (ej., python scada\_trl4.py).
5. **Simulador/Dispositivo de Campo:** Asegurar que el simulador Modbus TCP esté en ejecución o que el PLC/RTU esté encendido y configurado como esclavo Modbus TCP con los registros correctos.
6. **Verificación Visual:** Observar la interfaz de Tkinter para confirmar la adquisición de datos y probar los controles manuales.

**Consideraciones Clave (Fase 1)**

* **Simplicidad:** Priorizar la simplicidad sobre la robustez o la escalabilidad.
* **No Persistencia:** Los datos no se almacenan en una base de datos; la información se pierde al cerrar la aplicación.
* **Cero Automatización:** No se utilizan herramientas de CI/CD; todo el proceso es manual.
* **Entorno Aislado:** El despliegue se realizará exclusivamente en un entorno de red aislado para evitar riesgos de seguridad innecesarios en esta etapa inicial.

**Implementación y Despliegue - Fase 2 (TRL7)**

La implementación y el despliegue de la Fase 2 son procesos complejos y automatizados, orientados a la producción, la alta disponibilidad y la ciberseguridad.

**Metodología de Implementación (Fase 2)**

* **Desarrollo Dirigido por Pruebas (TDD) / Integración Continua (CI):** El desarrollo se guiará por pruebas unitarias y de integración para asegurar la calidad del código. Los cambios se integrarán y probarán de forma continua en un repositorio centralizado.
* **Despliegue Continuo (CD):** Se automatizará el proceso de construcción, prueba y despliegue del software en los diferentes entornos (desarrollo, staging, producción).
* **Infraestructura como Código (IaC):** Las configuraciones de los servidores, la red y el despliegue de contenedores se definirán mediante código (ej. archivos Docker Compose, manifiestos Kubernetes, Ansible), lo que garantiza la repetibilidad y consistencia.
* **Microservicios / Contenedores:** Cada componente (backend, frontend, bases de datos, drivers) se construirá como un servicio independiente, empaquetado en contenedores Docker.

**Herramientas de Implementación y Despliegue (Fase 2)**

* **Control de Versiones:** Git (GitHub, GitLab, Bitbucket) para gestionar el código fuente.
* **Plataforma de Contenedores:** Docker para la creación y gestión de imágenes de contenedores.
* **Orquestación de Contenedores:**
  + **Docker Compose:** Para despliegues en un solo host o entornos de desarrollo/staging.
  + **Kubernetes (K8s):** Para despliegues distribuidos, alta disponibilidad, escalabilidad y auto-recuperación en entornos de producción. Se utilizarán manifiestos YAML para describir el despliegue de los servicios.
* **Automatización de CI/CD:**
  + **GitLab CI/CD, GitHub Actions, Jenkins, Azure DevOps Pipelines:** Para automatizar los flujos de trabajo de integración y despliegue continuo (ej., construir imágenes Docker, ejecutar pruebas, desplegar en clústeres).
* **Gestión de Configuración:**
  + **Ansible, Chef, Puppet:** Para la automatización de la configuración del sistema operativo y la instalación de dependencias en los servidores.
* **Monitorización y Logging:**
  + **Prometheus y Grafana:** Para la monitorización de métricas de rendimiento y salud del sistema.
  + **ELK Stack (Elasticsearch, Logstash, Kibana) o Grafana Loki:** Para la agregación, búsqueda y visualización de logs de todos los componentes.
* **Gestión de Secretos:** HashiCorp Vault, Kubernetes Secrets, o variables de entorno para almacenar credenciales de forma segura.

**Proceso de Despliegue (Fase 2)**

El proceso de despliegue de la Fase 2 es altamente estructurado y automatizado:

1. **Configuración de Infraestructura (IaC):**
   * Provisionar los servidores (físicos o virtuales) en la red IT y OT.
   * Configurar la segmentación de red (VLANs, firewalls, rutas) según el diseño de ciberseguridad.
   * Instalar Docker (y Kubernetes si aplica) y sus dependencias en los servidores.
   * Configurar el servidor web (Nginx/Apache) como proxy inverso y para HTTPS/TLS.
2. **Construcción de Imágenes Docker:**
   * Para cada servicio (Django backend, FastAPI drivers, frontend), se construirán imágenes Docker a partir del código fuente. Este proceso se automatiza en el pipeline de CI.
   * Las imágenes se almacenarán en un registro de contenedores seguro (ej. Docker Hub privado, GitLab Container Registry, Azure Container Registry).
3. **Despliegue de Bases de Datos:**
   * Desplegar contenedores de MySQL e InfluxDB.
   * Configurar la replicación y alta disponibilidad para MySQL.
   * Configurar las políticas de retención y almacenamiento para InfluxDB.
   * Crear los esquemas de base de datos y usuarios con los permisos mínimos necesarios.
4. **Despliegue de Servicios SCADA:**
   * Utilizar Docker Compose o manifiestos Kubernetes para desplegar los contenedores de los servicios SCADA (Core SCADA, Data Collectors, etc.).
   * Configurar variables de entorno y montar volúmenes persistentes para datos y logs.
   * Configurar los puntos de montaje de red para acceder a los dispositivos de campo desde los Data Collectors.
5. **Configuración Inicial del SCADA:**
   * A través de la interfaz web o scripts de inicialización, configurar los dispositivos de campo, tags, umbrales de alarmas, usuarios y roles. Esta configuración se almacenará en MySQL.
6. **Integración con Dispositivos de Campo:**
   * Conectar físicamente los PLCs/RTUs a la red OT.
   * Configurar las direcciones IP y los parámetros de comunicación de los dispositivos de campo.
   * Verificar la conectividad de los Data Collectors con los dispositivos.
7. **Pruebas Post-Despliegue:**
   * Ejecutar pruebas de humo (smoke tests) para verificar que todos los servicios están en marcha y funcionando correctamente.
   * Realizar pruebas de integración, rendimiento y seguridad en el entorno de despliegue.
8. **Monitorización Activa:**
   * Activar la monitorización del sistema (Prometheus/Grafana) y el logging centralizado (ELK/Loki) para supervisar la salud del sistema, el rendimiento y detectar anomalías.

**Consideraciones Clave (Fase 2)**

* **Ambientes de Despliegue:** Se mantendrán al menos tres ambientes (desarrollo, staging/test, producción) para asegurar que los cambios se prueban a fondo antes de llegar a la producción.
* **Rollback Plan:** Tener un plan de rollback bien definido y probado para revertir a una versión anterior en caso de un fallo crítico en el despliegue.
* **Actualizaciones (Zero Downtime):** Diseñar el despliegue para permitir actualizaciones de software con interrupción mínima o nula del servicio (zero downtime deployments) utilizando estrategias de Kubernetes como "Rolling Updates".
* **Documentación:** Mantener una documentación exhaustiva de todo el proceso de despliegue, configuraciones, y playbook de operaciones.
* **Formación:** Capacitar al personal de operaciones y mantenimiento sobre el despliegue, monitorización y resolución de problemas del sistema.

Esta estrategia de implementación y despliegue robusta y automatizada garantiza que la Plataforma SCADA23 se entregue de manera eficiente, segura y con alta disponibilidad, lista para las demandas operacionales del sector del agua potable.

## Despliegue en Windows 111

Este capítulo aborda aspectos clave relacionados con la organización del desarrollo del proyecto SCADA23 en un entorno Windows 11, el proceso de despliegue y las consideraciones de seguridad específicas a nivel de implementación. Estas directrices son fundamentales para garantizar la calidad del código, la eficiencia en la operación y la protección del sistema contra amenazas.

**Estructura del Repositorio de Código**

Para mantener un proyecto Python organizado y facilitar la colaboración en un entorno Windows, se propone la siguiente estructura inicial para el repositorio de código de SCADA23. Esta estructura fomenta la modularidad y la separación de responsabilidades:

Pantalla de computadora con letras

El contenido generado por IA puede ser incorrecto.

**Proceso de Despliegue del Sistema en Windows 11**

* El despliegue del sistema SCADA23 se diseñará para ser robusto y reproducible en un entorno Windows 11. Se optará por un enfoque basado en entornos virtuales de Python y servicios del sistema operativo para la ejecución en producción.
* **Preparación del Entorno:**
  + El sistema se desplegará en un servidor con Windows 11 o Windows Server, con la versión de Python adecuada preinstalada.
  + Se creará un directorio dedicado para la aplicación SCADA23, por ejemplo, C:\SCADA23\_APP.
  + Se recomienda crear un usuario de sistema dedicado con los permisos mínimos necesarios para ejecutar la aplicación.
* **Gestión de Dependencias con venv:**
  + Dentro del directorio de la aplicación, se creará y activará un entorno virtual de Python (venv).
  + Todas las dependencias del proyecto, especificadas en requirements.txt, se instalarán dentro de este entorno virtual. Esto asegura el aislamiento de dependencias y la reproducibilidad.
* **Ejecución del Backend (Servicios):**
  + Los componentes de backend de Python (ej., hmi\_backend, data\_acquisition, control\_logic) se configurarán para ejecutarse como servicios de Windows.
  + Se utilizarán herramientas como NSSM (Non-Sucking Service Manager) o scripts de PowerShell para registrar los scripts Python como servicios de Windows, asegurando que se inicien automáticamente con el sistema y se puedan gestionar fácilmente (iniciar, detener, reiniciar).
  + Cada servicio Python se ejecutará utilizando su entorno virtual venv correspondiente.
* **Servidor Web (para HMI):**
  + Si la HMI es una aplicación web, el backend de FastAPI se ejecutará utilizando un servidor WSGI asíncrono como Uvicorn.
  + Para la producción, se puede configurar un proxy inverso (ej., Nginx en Windows o IIS con extensiones de URL Rewrite) para servir los archivos estáticos del frontend de la HMI y enrutar las solicitudes a la API de FastAPI que se ejecuta en Uvicorn. Esto también facilita la implementación de HTTPS.
* **Base de Datos:**
  + La base de datos (ej., PostgreSQL o MySQL) se instalará y configurará en el mismo servidor o en un servidor de base de datos dedicado, siguiendo las mejores prácticas de seguridad y rendimiento.
  + El database\_manager de SCADA23 se conectará a esta instancia.
* **Monitorización del Despliegue:**
  + Se utilizarán las herramientas de monitorización de rendimiento de Windows (Monitor de Rendimiento, Visor de Eventos) para supervisar el uso de recursos y el estado de la aplicación.
  + Los logs generados por la aplicación Python se centralizarán en archivos en disco, que pueden ser analizados localmente o integrados con soluciones de gestión de logs si están disponibles.

**Consideraciones de Seguridad en la Implementación para Windows 11**

* La seguridad es un requisito no funcional crítico para la plataforma SCADA23. Además de los requisitos generales, se adoptarán las siguientes prácticas específicas durante la codificación y el despliegue en un entorno Windows para mitigar riesgos:
* **Manejo Seguro de Credenciales:**
  + **Nunca codificar credenciales directamente en el código.** Todas las credenciales (bases de datos, APIs externas, accesos a dispositivos) se gestionarán mediante variables de entorno de Windows o archivos de configuración cifrados (ej., utilizando bibliotecas de cifrado de Python como cryptography para claves AES).
  + Se limitará el acceso a estos secretos a solo los usuarios de sistema y servicios que los necesiten, bajo el principio de privilegio mínimo.
  + Para información extremadamente sensible, se podría considerar el uso del Gestor de Credenciales de Windows o herramientas de terceros.
* **Validación Estricta de Entradas (Input Validation):**
  + Todas las entradas de datos de usuarios (a través de la HMI) y de dispositivos (lecturas de sensores, comandos) serán validadas rigurosamente.
  + Se evitarán inyecciones de código (SQL Injection, Command Injection) mediante el uso de parámetros parametrizados en las consultas a bases de datos (SQLAlchemy ORM) y la sanitización de todas las entradas antes de su procesamiento o ejecución.
  + Las validaciones se realizarán tanto en el frontend como, y especialmente, en el backend.
* **Gestión de Sesiones y Autenticación/Autorización:**
  + Se implementarán mecanismos robustos de autenticación de usuarios (ej., basados en tokens JWT si la HMI es web).
  + Se aplicará un control de acceso basado en roles (RBAC) para la autorización, asegurando que los usuarios solo puedan realizar las acciones para las que tienen permisos (ej., un operador solo puede ver ciertos datos, un administrador puede modificar configuraciones).
  + Las sesiones de usuario deben tener tiempos de vida limitados y gestionarse de forma segura.
* **Principios de Mínimo Privilegio (Windows):**
  + Los servicios de Windows que ejecutan los componentes de SCADA23 deben configurarse para ejecutarse con las cuentas de usuario de sistema dedicadas y con los permisos mínimos necesarios. Evitar ejecutar servicios como LocalSystem a menos que sea estrictamente necesario y se comprendan los riesgos.
  + Los directorios de la aplicación y de datos deben tener permisos NTFS restringidos para que solo el usuario del servicio y los administradores autorizados puedan acceder a ellos.
* **Actualizaciones y Parches de Seguridad:**
  + Se establecerá un proceso para la monitorización de vulnerabilidades conocidas en las bibliotecas de Python y frameworks utilizados (pip-audit u otras herramientas de SCA - Software Composition Analysis).
  + Se aplicarán rápidamente los parches de seguridad del sistema operativo Windows y de Python.
* **Logging y Auditoría de Seguridad:**
  + Se registrarán todos los eventos de seguridad relevantes (intentos de inicio de sesión fallidos, cambios de configuración, comandos críticos ejecutados) con marcas de tiempo y detalles del usuario/origen.
  + Estos logs se escribirán en archivos que pueden ser configurados para rotación y, si es necesario, integrados con el Visor de Eventos de Windows o un sistema de gestión de eventos e información de seguridad (SIEM).
* **Comunicación Segura:**
  + Toda la comunicación entre componentes del sistema (backend HMI a base de datos, módulos internos) se protegerá utilizando cifrado (ej., TLS/SSL) cuando sea posible, especialmente si atraviesan redes no confiables.
  + Las interfaces de red expuestas al exterior (ej., API de HMI) deberán estar protegidas con TLS/HTTPS.
  + Se configurará el Firewall de Windows para permitir solo el tráfico necesario hacia y desde la aplicación SCADA23.
* **Manejo de Errores Detallado (Seguridad):**
  + Los mensajes de error expuestos al usuario final no deben revelar información sensible sobre la arquitectura interna del sistema, las rutas de archivos o los detalles de la base de datos. Se usarán mensajes genéricos y se registrarán los detalles técnicos internamente.

# CÓDIGO FUENTE DEL REPOSITORIO

Se muestra a continuación el fichero README.md alojado en github:

## Fichero README.md

# <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.

Este capítulo aborda aspectos clave relacionados con la organización del desarrollo del proyecto SCADA23 en un entorno Windows 11, el proceso de despliegue y las consideraciones de seguridad específicas a nivel de implementación. Estas directrices son fundamentales para garantizar la calidad del código, la eficiencia en la operación y la protección del sistema contra amenazas.

## Estructura del repositorio de código

Para mantener un proyecto Python organizado y facilitar la colaboración en un entorno Windows, se propone la siguiente estructura inicial para el repositorio de código de SCADA23. Esta estructura fomenta la modularidad y la separación de responsabilidades:

Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

## 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.

El módulo inicial implementado por defecto es el del Sector de Agua Potable. Este viene preconfigurado con el proceso de ETAP que se describe, con las funcionalidades básicas de conectividad para el sector de agua potable.

|  |  |  |
| --- | --- | --- |
| **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 |

# ANEXO: PATRONES DE DISEÑO

Para asegurar la modularidad, escalabilidad, mantenibilidad y robustez de la implementación en Python de la plataforma SCADA23, se considerará la aplicación estratégica de patrones de diseño de software. Estos patrones ofrecen soluciones probadas a problemas comunes en el desarrollo de software, promoviendo un código más limpio, flexible y fácil de extender.

## Patrones de Diseño de Software utilizados en SCADA23

La aplicación de patrones de diseño de software será fundamental para construir un sistema modular, escalable y mantenible. Se considerará la implementación de los siguientes patrones:

* **Observer:** Para la propagación de cambios de estado (ej., actualización de valores de sensores, cambio de estado de actuadores, activación de alarmas) desde los módulos de adquisición de datos y control hacia la HMI, el gestor de alarmas y el sistema de logging.
* **Singleton:** Para asegurar que solo exista una instancia de recursos globales o configuraciones críticas, como la conexión a la base de datos, el gestor de configuración del sistema o el logger centralizado.
* **Factory Method:** Para la creación flexible de objetos de dispositivos o manejadores de protocolo, permitiendo añadir fácilmente nuevos tipos de dispositivos o protocolos sin modificar el código existente en las capas superiores.
* **Command:** Para encapsular las solicitudes de control de los operadores (ej., arrancar/parar bombas), facilitando el registro de acciones, la implementación de colas de comandos y la posibilidad de rehacer/deshacer operaciones en ciertos contextos.
* **State:** Para gestionar el comportamiento de objetos complejos (ej., bombas, válvulas) en función de su estado interno, simplificando la lógica de control y haciendo el código más legible y extensible.
* **Adapter:** Para integrar librerías de terceros o drivers de protocolo con interfaces incompatibles, presentando una interfaz unificada al resto del sistema.

La selección y aplicación de estos patrones se documentará explícitamente en el código y en la documentación técnica de bajo nivel para asegurar la comprensión y la coherencia del diseño.

A continuación, se describen algunos de los patrones de diseño que serán evaluados para su aplicación en diferentes módulos del sistema SCADA23:

## Patrones Creacionales

Los patrones creacionales se encargan de la creación de objetos, proporcionando mecanismos para crear objetos de manera flexible y reutilizable.

* **Singleton:**
  + **Propósito:** Garantizar que una clase tenga una única instancia y proporcionar un punto de acceso global a ella.
  + **Aplicación potencial en SCADA23:**
    - **Gestión de Conexiones a Base de Datos:** Podría utilizarse para asegurar que solo exista una instancia de la conexión a la base de datos a lo largo de la aplicación, optimizando los recursos.
    - **Gestor de Configuración:** Una única instancia para cargar y gestionar la configuración global del sistema (ej., parámetros de red, umbrales de alarmas).
    - **Logger Centralizado:** Un objeto de log que se utiliza en toda la aplicación para escribir eventos y mensajes, asegurando un punto de control unificado para el registro.
* **Factory Method (Método de Fábrica):**
  + **Propósito:** Proporcionar una interfaz para crear objetos en una superclase, pero permitiendo que las subclases alteren el tipo de objetos que se crearán.
  + **Aplicación potencial en SCADA23:**
    - **Creación de Dispositivos/Sensores:** Permitiría crear diferentes tipos de objetos de dispositivos (ej., Bomba, Válvula, SensorNivel) a través de una interfaz común, abstractos del proceso de instanciación específico de cada tipo de dispositivo. Esto es útil si el sistema necesita soportar varios fabricantes o modelos con ligeras variaciones en su inicialización.
    - **Manejadores de Protocolo:** Si se necesitan diferentes implementaciones para Modbus RTU, Modbus TCP, OPC UA, un método de fábrica podría instanciar el manejador de protocolo adecuado basado en la configuración.

## Patrones Estructurales

Los patrones estructurales se ocupan de la composición de clases y objetos, facilitando la organización de las relaciones entre ellos.

* **Adapter (Adaptador):**
  + **Propósito:** Permitir que interfaces incompatibles trabajen juntas.
  + **Aplicación potencial en SCADA23:**
    - **Integración de Protocolos/Drivers de Hardware:** Si el sistema necesita comunicarse con dispositivos que utilizan protocolos de comunicación muy diferentes (ej., Modbus, DNP3, OPC UA, MQTT), un patrón Adapter podría envolver la lógica específica de cada protocolo, presentando una interfaz unificada al resto del sistema. Esto abstrae la complejidad de cada driver.
    - **Integración con Sistemas Legados:** Si SCADA23 debe interactuar con sistemas existentes con APIs o formatos de datos antiguos, un Adapter podría traducir las solicitudes y respuestas.

## Patrones de Comportamiento

Los patrones de comportamiento se centran en la comunicación entre objetos y la asignación de responsabilidades.

* **Observer (Observador):**
  + **Propósito:** Definir una dependencia uno-a-muchos entre objetos de modo que cuando un objeto cambie de estado, todos sus dependientes sean notificados y actualizados automáticamente.
  + **Aplicación potencial en SCADA23:**
    - **Monitorización de Valores de Sensores:** Cuando el valor de un sensor cambia (ej., nivel de tanque, presión), múltiples componentes pueden necesitar ser notificados: la HMI para actualización de visualización, el módulo de alarmas para verificación de umbrales, y el módulo de control automático para tomar decisiones.
    - **Gestión de Alarmas y Eventos:** Un sistema de publicación/suscripción donde los generadores de alarmas (Sujetos) notifican a los suscriptores (Observadores) como el HMI, el sistema de logging o el módulo de notificación por email.
    - **Estados de Equipo:** Cuando el estado de una bomba o válvula cambia, los componentes dependientes (ej., lógica de control de interbloqueos, historial de eventos) son actualizados automáticamente.
* **Command (Comando):**
  + **Propósito:** Encapsular una solicitud como un objeto, lo que permite parametrizar clientes con diferentes solicitudes, poner las solicitudes en cola o registrarlas, y soportar operaciones que pueden ser deshechas.
  + **Aplicación potencial en SCADA23:**
    - **Control de Actuadores Remotos:** Cada acción del operador (ej., "Arrancar Bomba X", "Abrir Válvula Y") podría ser encapsulada como un objeto Command. Esto permitiría registrar todas las acciones del operador, reintentar comandos fallidos, o incluso implementar una funcionalidad de "deshacer" para ciertas operaciones (aunque esto es más complejo en SCADA).
    - **Cola de Comandos:** Para operaciones que no requieren respuesta inmediata o que deben ejecutarse en secuencia, los comandos pueden encolarse y procesarse de forma asíncrona.
* **State (Estado):**
  + **Propósito:** Permitir que un objeto altere su comportamiento cuando su estado interno cambia. El objeto parecerá cambiar su clase.
  + **Aplicación potencial en SCADA23:**
    - **Control de Bombas/Válvulas:** Un objeto Bomba podría tener diferentes estados (ej., 'Parada', 'Arrancando', 'Funcionando', 'Fallida'). El comportamiento de los métodos (ej., arrancar(), parar()) cambiaría dependiendo del estado actual de la bomba, simplificando la lógica condicional.
    - **Estados de Proceso:** El sistema global podría tener estados como 'Inicializando', 'Operación Normal', 'Parada de Emergencia', 'Modo Mantenimiento', con comportamientos distintos asociados a cada estado.

La selección y aplicación de estos patrones se realizará de manera pragmática, priorizando la claridad, la simplicidad y la optimización del rendimiento en el contexto de un sistema SCADA. El uso de patrones de diseño facilitará la colaboración entre desarrolladores, la adición de nuevas funcionalidades y la adaptación del sistema a futuros requisitos sin necesidad de reestructuraciones importantes del código base. Se promoverá la documentación de los patrones aplicados en el código mediante comentarios y docstrings claros.

# ANEXO: TECHNOLOGY READINESS LEVELS

Se muestran a continuación los niveles de desarrollo:

* **TRL 1:** Nivel más bajo de la disponibilidad de la tecnología software. Se está investigando un nuevo dominio software por parte de la comunidad científica a nivel de investigación básica. Este nivel comprende el desarrollo de los usos básicos, así como las propiedades básicas de la arquitectura software, las formulaciones matemáticas y los algoritmos generales.
* **TRL 2:** Se comienza a investigar las aplicaciones prácticas del nuevo software aunque las posibles aplicaciones son todavía especulativas.
* **TRL 3:** Se comienza una actividad intensa de I+D y se comienza a demostrar la viabilidad del nuevo software a través de estudios analíticos y de laboratorio.
* **TRL 4:** Se comienzan a integrar los diferentes componentes de software básico para demostrar que pueden funcionar conjuntamente.
* **TRL 5:** En este nivel la nueva tecnología software se encuentra preparada para integrarse en sistemas existentes y los algoritmos pueden ejecutarse en procesadores con características similares a las de un entorno operativo.
* **TRL 6:** En este nivel se pasaría de las implementaciones a nivel de propotipo de laboratorio a implementaciones completas en entornos reales.
* **TRL 7:** En este nivel la tecnología software está preparada para su demostración y prueba con sistemas HW/SW operativos.
* **TRL 8:** En este nivel todas las funcionalidades del nuevo software se encuentran simuladas y probadas en escenarios reales.
* **TRL 9:** En este nivel la nueva tecnología software se encuentra totalmente disponible y se puede utilizar en cualquier entorno real.

Fuente: <https://www.mintur.gob.es/publicaciones/publicacionesperiodicas/economiaindustrial/revistaeconomiaindustrial/393/notas.pdf>