ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

DESARROLLO DE UN PROTOTIPO DE SOFTWARE PARA ADQUISICIÓN Y GESTIÓN DE DATOS DE EQUIPOS EN REDES INDUSTRIALES, PARA LA EMPRESA HIESELAT S.A

TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN ELECTRÓNICA Y REDES DE INFORMACIÓN

JONATHAN EDUARDO PINZÓN RUEDA

jonathan.pinzon@epn.edu.ec

DIRECTOR: Ing. XAVIER ALEXANDER CALDERÓN HINOJOSA MSc.

xavier.calderon@epn.edu.ec

Quito, Marzo 2019

AVAL

Certifico que el presente trabajo fue desarrollado por Jonathan Eduardo Pinzón Rueda, bajo mi supervisión.

_____

MSc. Xavier Calderón
DIRECTOR DEL TRABAJO DE TITULACIÓN

DECLARACIÓN DE AUTORÍA

Yo, Jonathan Eduardo Pinzón Rueda, declaro bajo juramento que el trabajo aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración cedo mis derechos de propiedad intelectual correspondientes a este trabajo, a la Escuela Politécnica Nacional, según lo establecido por la Ley de Propiedad Intelectual, por su Reglamento y por la normatividad institucional vigente.

Jonathan Eduardo Pinzón Rueda

DEDICATORIA

A mi familia, mis hermanos Xavier y Bryan gracias por siempre estar ahí

A mis padres Inesita y Sixto gracias por nunca dejar de creer en mí

Y sobre todo a las dos personas que son el motor de mi vida y la razón por la cual hago todo a pesar de los obstáculos que se presenten:

Esposa mía Adriana y mi hija Paula Zoé, todo en mi vida por y para ustedes

AGRADECIMIENTO

A DIOS Y A MI VIRGENSITA DEL QUINCHE QUE NUNCA ME HAN DEJADO SOLO.

A mis amigos de mi vida universitaria: Pame, Fer, Karlita, Dany, Aleja, Gio, Vico, Carlos, Santi, Andrés, mis ñaños Xavi y Leo y todos mis demás amigos que siempre creyeron en mí y estuvieron brindándome su apoyo en todo momento.

A mis amigos de la carrera “Los Panas Redes”: Vane, Liss, Belén, Roberto, Edu, Ángel, Galo, Andrés L., Andrés A. y Diego gracias por todas las alegrías y momentos tristes que hemos pasado en todo este tiempo.

A la familia Hieselat: Jhonny, Vito, Luisin, Gabriel y Daniel, siempre gracias por todo el apoyo, oportunidades y confianza han depositado en mí.

A mis compañeros de trabajo “Los Sositos”: Juan, Franklin y Fernando, gracias por todo el apoyo en esta etapa final y quedo inmensamente agradecido porque sin su apoyo no lo hubiera logrado

A todas esas personas que llegaron siempre en el momento justo para darme una mano sin tener ninguna obligación de hacerlo: Primo Klever, Fercho, Cris, Jeff, y mi ñaño Ricky infinitamente agradecido amigos gracias por todo el apoyo.

A mi familia: Coquito, Guambra, Inesita y Viejito les quiero mucho, gracias por nunca dejar de creer en mí, por el apoyo en todo sentido y sobre todo gracias por entenderme en los momentos más difíciles.

Y sobre todo a mis AMORES, Adriana y Paula toda una vida para agradecer el amor y la comprensión que han tenido conmigo a lo largo de este camino lleno de alegrías y momentos difíciles las amo con mi vida…. SIEMPRE GRACIAS MIS SUKAS.

ÍNDICE DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ÍNDICE DE CÓDIGOS

RESUMEN

Actualmente el mundo de la industria gira alrededor del eje de la automatización, por lo cual casi todos los procesos deben ser automatizados. La mayoría de las herramientas utilizadas en el proceso de automatización son propietarias, además de tener un costo elevado, el cual es justificado al momento de realizar la relación costo/beneficio en el desarrollo de un determinado proyecto de automatización.

Si se analiza las herramientas de software utilizadas para este tipo de tareas, se observa que existe una amplia gama, esto debido a que el mundo de la automatización es extenso y por lo tanto necesita herramientas cada vez más completas, y específicas dependiendo del área en el cual se las vaya a utilizar.

Sin embargo en Ecuador no existe desarrollo de software libre como herramienta para implementar sistemas de automatización.

En algunos lugares del mundo se empezó a desarrollar este tipo de herramientas ya que su precio es altamente costoso y si se parte como escenario un proyecto pequeño, donde la relación costo/beneficio indica que no es rentable desarrollar el proyecto, es de vital importancia contar con herramientas como estas para abaratar los costos.

Por tal motivo en este trabajo de titulación se plantea el desarrollo de un prototipo de software para la empresa Hieselat S.A, empresa dedicada al diseño e implantación del control y automatización industrial, con el cual se pueda realizar algunas de las funcionalidades más básicas e importantes con las que cuenta el software que actualmente se utiliza en la industria para desarrollar proyectos de automatización.

Al desarrollar este prototipo de software, la empresa Hieselat S.A contará con una herramienta de software de licenciamiento libre, con el cual podrá realizar proyectos de automatización a pequeña escala.

PALABRAS CLAVE: Relación Costo/Beneficio, Arquitectura Maestro/Esclavo, ModBus/TCP, HMI (*Human-Machine Interface*).

ABSTRACT

Currently the world of industry revolves around the axis of automation, which is why almost all processes must be automated. Most of the tools used in the automation process are proprietary, in addition to having a high cost, which is justified when making the cost / benefit in the development of a particular automation project.

If you analyze the software tools used for this type of tasks, it is observed that there is a wide range, this because the world of automation is extensive and therefore needs more and more complete tools, and specific depending on the area in which will be used.

However, in Ecuador there is no free software development as a tool to implement automation systems.

In some parts of the world, this type of tool was developed, since its price is highly expensive and if a small project is part of the scenario, where the cost / benefit ratio indicates that it is not profitable to develop the project, it is of vital importance to with tools like these to lower costs.

For this reason, in this titling work, the development of a software prototype for the company Hieselat SA, a company dedicated to the design and implementation of industrial automation and control, is proposed, with which it can perform some of the most basic and important functionalities with those that the software currently used in the industry has to develop automation projects.

When developing this software prototype, the company Hieselat S.A will have a software tool of free licensing, with which it will be able to carry out automation projects on a small scale.

KEYWORDS: Cost/Benefit Relation, Mater/Slave Architecture, ModBus/TCP, HMI (Human-Machine Interface).

1 INTRODUCCIÓN

Actualmente el contar con software que facilite la adquisición y manipulación de datos sobre un sistema, es de vital importancia independientemente del área en el que este sea utilizado. Más aún si se trata de un software de licenciamiento libre o de un costo relativamente bajo con el cual se pueda disminuir los costos de implementación de un proyecto.

Centrándose en el área industrial casi todos sus procesos son automatizados, para ello se utiliza tanto software como hardware especialmente diseñado para cumplir con las funcionalidades que demanda la implementación de un proceso de control y automatización. Tanto software como hardware se encuentran a disposición en el mercado, pero a un costo muy elevado.

El precio de un software licenciado oscila entre los $3.000 a $6.000 dólares para un solo equipo, si se habla de *Intouch*[1]. Este precio depende del número de variables (*tags*) que pueda manejar y si es la versión de desarrollo o solamente la de *runtime* [1].

A pesar de esto no se puede observar un desarrollo de herramientas libres o de bajo costo, con las cuales se pueda implementar cualquier proyecto de automatización sin discriminar si su relación costo/beneficio lo hace rentable o no.

Por tal motivo en este trabajo de titilación se desarrolló un prototipo de software con el cual se empiece a contar con un software libre como herramienta para desarrollar un proyecto de control y automatización a pequeña escala.

Para fines de diseño de software se utilizó “Historias de Usuario”, una forma natural de análisis funcional y descripción de requisitos de un determinado cliente. Además de una fase de pruebas y verificación de los diseños de interfaces gráficas con las que cuenta el prototipo de software.

Teniendo presente la importancia del desarrollo de este tipo de herramientas, también se consideran las limitaciones que presenta al ser comparada con las herramientas licenciadas que se ofrecen actualmente en el mercado para desarrollar proyectos de control y automatización industrial. Sin embargo es un prototipo con el cual se pueden cubrir necesidades básicas y puntuales para proyectos a pequeña escala, y sobre todo contar con un software base sobre el cual a futuro se podrían seguir añadiendo funcionalidades un poco más complejas y para algún tipo específico de industria.

[1] Intouch es un sistema interactivo diseñado para la visualización, la supervisión y el control de procesos industriales.

1.1 Objetivos

El objetivo general de este Proyecto Técnico es: Desarrollar un prototipo de software con herramientas de software libre, para poder adquirir y gestionar las variables de dispositivos de una red industrial implementada por tres esclavos y un maestro, donde los esclavos estarán representados por dos simuladores ModBus y un PLC (*Programmable Logic Controller*), el Maestro será el prototipo de software.

Los objetivos específicos de este Proyecto Integrador son.

- Analizar los conceptos fundamentales del protocolo ModBus/TCP y sus funciones, así como también el modelo de arquitectura de red Maestro/Esclavo.
- Diseñar los módulos que conforman el prototipo de software.
- Implementar los módulos previamente definidos en el alcance, los cuales conforman el prototipo de software.
- Analizar los resultados obtenidos en la implementación del prototipo de softwar

1.2 Alcance

Se pretende desarrollar un prototipo de software el cual realice las funciones de adquisición y gestión de datos de los dispositivos de redes industriales.

En cuanto a la adquisición de datos, se van a leer datos del tipo *Coil Status*[2], *Input Status*[3], *Holding Registers*[4] e *Input Registers*[5]. También escribir datos del tipo *Holding Registers* y *Coil Status*. Estos tipos de datos se encuentran almacenados en tablas dentro de los dispositivos que manejan el protocolo ModBus/TCP.

Por otro lado se va a gestionar los datos, lo que implica el almacenamiento del valor que tengan los registros en una base de datos y su manipulación. Es decir de acuerdo a la información que se obtenga en la lectura, esta podrá ser procesada y como resultado se escribirá en las salidas de un determinado dispositivo (esclavo).

Todo esto se realizará empleando la arquitectura de red Maestro/Esclavo. Para esto se desarrollará únicamente el maestro, que en este caso será el prototipo de software; los esclavos están ya desarrollados y son representados por los equipos a gestionar.

Los esclavos serán dos simuladores ModBus y un equipo físico como: PLC, controlador de velocidad o un sensor, que manejen el protocolo de comunicación ModBus/TCP.

[2] *Coil Status*, es una bobina o una salida digital de un dispositivo ModBus que actúa como esclavo.
[3] *Input Status*, es una entrada digital de un dispositivo ModBus que actúa como esclavo.
[4] *Holding Registers*, es un registro de salida del tipo palabra (2 bytes) de un dispositivo ModBus que actúa como esclavo.
[5] *Input Registers*, es un registro de entrada del tipo palabra (2 bytes) de un dispositivo ModBus que actúa como esclavo.

Este software será implementado en el lenguaje de programación Java. Además, se utilizará el IDE (*Integrated Development Environment*) de Eclipse.

Para la parte de generación de registros se va a recolectar información de las variables temperatura, caudal de entrada y nivel del tanque. Información como: nombre de la variable, número de esclavo a la que pertenece, nombre del historial a generar, valor de la variable, fecha y hora de muestreo. Todo este conjunto de información será almacenado en una base de datos utilizando el sistema de gestión MySQL para posteriormente poder manipularla de acuerdo a las necesidades del usuario.

Además se realizará interfaces gráficas para la adquisición y gestión de datos utilizando como criterios de diseño los requerimientos dados por la empresa Hieselat S.A y ratificados por los resultados arrojados en las encuestas que se realizarán a ingenieros que han trabajado en la industria con software para monitoreo y control de datos.

1.3 Marco Teórico

1.3.1 Protocolo ModBus [2]

El protocolo ModBus es una estructura de mensajería desarrollada por Modicon en 1979. Se utiliza para establecer la comunicación Maestro-Esclavo/Cliente-Servidor entre dispositivos inteligentes.

ModBus es un estándar de facto, abierto y además es el protocolo de red más utilizado en el entorno de fabricación industrial. Ha sido implementado por cientos de proveedores en miles de dispositivos diferentes para transferir E/S discretas / analógicas y registrar datos entre dispositivos de control.

Este protocolo se usa en múltiples aplicaciones que tienen la arquitectura del tipo maestro-esclavo, para establecer la comunicación entre dispositivos inteligentes, sensores e instrumentos que utilizan este protocolo.

1.3.2 ModBus TCP/IP [3]

El protocolo TCP permite establecer un gran número de conexiones concurrentes, de este modo el Cliente (Maestro) puede reusar una conexión previamente establecida o crear una nueva en el momento de realizar una transacción de datos.

La arquitectura TCP/IP usa un conjunto de protocolos, siendo los más importantes los protocolos:

TCP: Protocolo de Control de Transmisión.

IP: Protocolo de Internet.

Combinando una red física (Ethernet) con un estándar de red universal (TCP/IP), ModBus ofrece una red abierta y accesible para el intercambio de datos de proceso. Es simple de implementar para cualquier dispositivo que admita sockets TCP/IP [2].

ModBus TCP/IP se ha vuelto omnipresente debido a su apertura, simplicidad, desarrollo de bajo costo y hardware mínimo requerido para soportarlo. Existen varios cientos de dispositivos ModBus TCP/IP disponibles en el mercado, y cada año se desarrollan más.

ModBus TCP/IP se usa para intercambiar información entre dispositivos, monitorear y programarlos, también se usa para administrar E/S distribuidas, siendo el protocolo preferido por los fabricantes de este tipo de dispositivos [2].

1.3.3 Mecanismo de Conexión en ModBus/TCP [4]

En ModBus, cada solicitud del maestro es tratada de forma independiente por el esclavo, sin relación con las anteriores. Esto facilita proveer transacciones de datos robustas, requiriendo mínima información de recuperación para mantener una transacción en cualquiera de los dos terminales. De otro lado, las operaciones de programación esperan una comunicación orientada a la conexión, es decir, las máquinas de origen y de destino deben establecer un canal de comunicaciones antes de transferir datos. Este tipo de operaciones son implementadas de diferentes maneras por las diversas variantes de MODBUS (ModBus RTU[6], ModBus ASCII[7], ModBus PLUS[8]), etc.

En ModBus/TCP una conexión se establece inicialmente en la capa de aplicación y esta única conexión puede llevar múltiples transacciones independientes. Además, TCP permite establecer un gran número de conexiones concurrentes, de modo que un Maestro puede reusar una conexión previamente establecida como crear una nueva en el momento de requerir una transacción de datos. En ModBus/TCP se usa el protocolo orientado a la conexión TCP en lugar del protocolo orientado a datagramas UDP. Se busca mantener el control de una transacción individual encerrándola en una conexión que pueda ser identificada, supervisada y cancelada sin requerir acción específica por parte de las aplicaciones Cliente y Servidor. El monitoreo continuo de datos o "*streaming data*" no es

[6] ModBus RTU, se utiliza en la comunicación serie y hace uso de una representación binaria compacta de los datos para el protocolo de comunicación [5, p. 9].
[7] ModBus ASCII, se utiliza en la comunicación serie y hace uso de caracteres ASCII para el protocolo de comunicación [5, p. 9].
[8] ModBus PLUS, es una versión extendida del protocolo y privativa de Schneider Electric y a diferencia de las otras variantes, soporta comunicaciones peer-to-peer entre múltiples masters [5, p. 9].

muy eficiente con el protocolo ModBus/TCP, debido básicamente a la sobrecarga que impone el protocolo de transporte TCP.

Esta sobrecarga se debe al servicio de entrega de datos confiable y el acuse de recibo para cada paquete transmitido, incrementándose considerablemente el tráfico en la red cuando se monitorea en todo momento un esclavo ModBus/TCP particular. Sin embargo, esto tiende a ser un problema menor a medida que Ethernet aumenta de velocidad. La solución para la supervisión de datos continuos sobre una red Ethernet es la utilización del protocolo de transporte UDP, pero se pierde confiabilidad.

1.3.4 Capas del Protocolo ModBus [6]

La designación de ModBus no corresponde propiamente a un estándar de red que incluye todos los aspectos desde el nivel físico hasta el de aplicación, sino a un protocolo de mensajes, posicionado en la capa de aplicación o nivel 7 del modelo OSI (*Open System Interconnection*), como se muestra en la Figura 1.1.

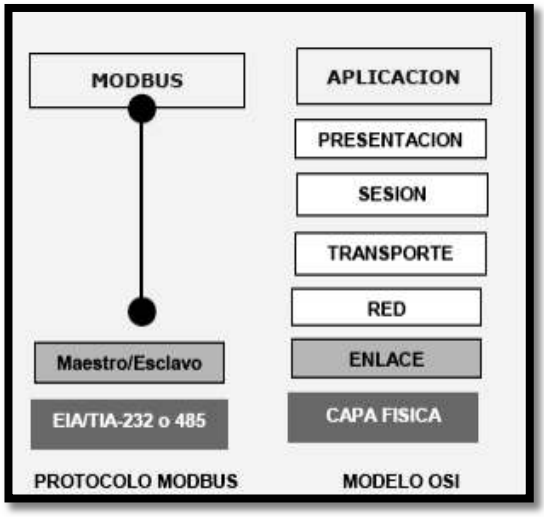


Figura 1.1. Capas del protocolo ModBus[7]

1.3.5 Unidad de Datos de Protocolo (PDU) ModBus [8]

El formato de la PDU de ModBus está estructurado como un campo denominado código de función seguido por un conjunto de datos asociado. El tamaño y el contenido de estos datos son definidos por el código de función y la PDU completa (código de función y datos) no puede exceder de 253 bytes de tamaño, tal como se indica en la Figura 1.2.

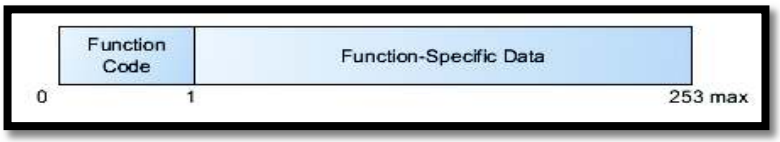


Figura 1.2. PDU de ModBus

Cada código de función tiene un comportamiento específico que los esclavos pueden implementar de manera flexible en base al comportamiento de la aplicación deseada.

La especificación de la PDU define conceptos básicos para el acceso y manipulación de datos; sin embargo, un esclavo puede manejar datos de una manera que no esté definida explícitamente en la especificación.

El código de función es el primer elemento que será validado. Si el código de función no es reconocido por el dispositivo que recibe la solicitud, responde con una excepción. Si se acepta el código de función, el dispositivo esclavo comienza a descomponer los datos de acuerdo con la definición de la función.

1.3.6 Unidad de Datos de Aplicación (ADU) ModBus [9]

En la trama se definen los campos y el tamaño asignado para cada uno. Como se muestra en la Figura 1.3.

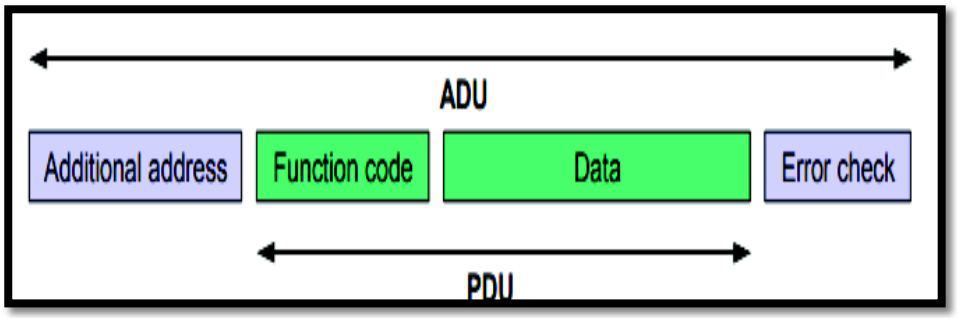


Figura 1.3. ADU ModBus

ID: Dirección del esclavo, números enteros, normalmente desde 1, hasta 255.

FUNCIÓN: Tipo de solicitud que se le realiza al esclavo, codificada de manera numérica, normalmente números que inician en 1, 2, 3, 4, etc.

DATO: Este campo se emplea para enviar información complementaria a la solicitud realizada al esclavo, o se emplea para responder a la solicitud, por lo tanto el campo DATO tiene una estructura diferente cuando el mensaje lo envía el maestro o cuando es la respuesta de un esclavo.

CRC: Realiza una verificación por redundancia cíclica (CRC) de 16 bits (2 bytes).

Trama Maestro

Para el maestro el campo “Dato” está integrado por dos subcampos, la dirección y la longitud. En la dirección se indica al esclavo en qué dirección debe buscar lo que se ha solicitado a través de la función. La longitud indica a partir de esa dirección cuántos elementos se deben tomar. Esto se puede observar en la Figura 1.4.

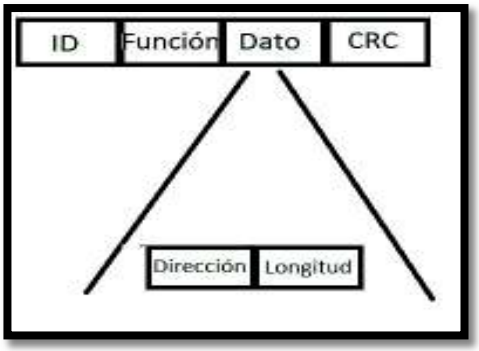


Figura 1.4. Estructura de la Trama Maestro

Trama Esclavo

La estructura se mantiene para el esclavo pero nuevamente el cambio está en el campo de Dato, pues aparecen dos sub campos que son: Número de bytes para dar la respuesta y la respuesta en sí. Esto se observa en la Figura 1.5.

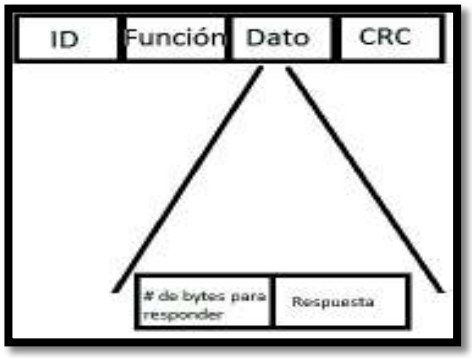


Figura 1.5. Estructura de la Trama Esclavo

Tamaño de la Trama

Ahora se analiza cada campo, cuantos bits, bytes o *Words* ocupan, para el caso del Protocolo ModBus se organizan bloques de tamaño mínimo de *Bytes* como lo muestra la estructura de la Figura 1.6.

Figura 1.6. Tamaño de la Trama ModBus

1.3.7 Formato de la Trama ModBus/TCP [10]

Una cabecera es empleada en TCP/IP para la identificación de la unidad MODBUS (capa aplicación). Es conocida como cabecera MBAP (*MODBUS Application Protocol Header*). Esta cabecera contiene algunas diferencias con respecto a la capa de aplicación del MODBUS RTU en línea serie, estas son las que se indica en la Figura 1.7.

- El campo dirección esclavo MODBUS empleado en MODBUS línea serie es sustituido por un único *byte*, identificador de unidad. El identificador de unidad es empleado para la comunicación a través de dispositivos como puentes, *routers* y *gateways* que emplean una única dirección IP que soportan múltiples e independientes unidades finales de MODBUS.

- Todas las peticiones y respuestas MODBUS están diseñadas para poder verificar que el mensaje ha finalizado. Para los códigos de la función donde la PDU de MODBUS tiene una longitud fija, con el código de la función es suficiente. Para los códigos de función cuya longitud no es fija posee un campo de datos adicional que actúa como contador de *bytes*.

- Cuando ModBus es transmitido en TCP, se le añade en la cabecera una información adicional de longitud del mensaje, que permite conocer los límites del mismo, incluso si el mensaje es enviado en múltiples paquetes.

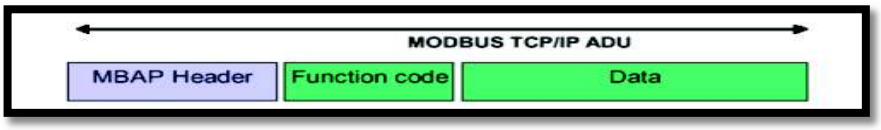


Figura 1.7. Trama ModBus/TCP ADU

Descripción de la Cabecera MBAP [8]

La cabecera MBAP contiene los siguientes campos que se muestran en la Tabla 1.1.

Tabla 1.1. Campos de la Cabecera MBAP

Campos	Longitud	Descripción	Cliente	Servidor
Identificador de trama	2 bytes	Identificación de una petición MODBUS/ Respuesta trama	Inicialización por el cliente (petición)	Recogido por el servidor desde la petición recibida
Identificador de protocolo	2 bytes	0 = protocolo MODBUS	Inicialización por el cliente (petición)	Recogido por el servidor desde la petición recibida
Longitud	2 bytes	Número de bytes	Inicialización por el cliente (petición)	Inicializado por el servidor (Respuesta)
Identificador de unidad	1 byte	Identificador del esclavo remoto conectado en la línea serie o en otro tipo de bus	Inicialización por el cliente (petición)	Recogido por el servidor desde la petición recibida

La cabecera tiene una longitud de 7 bytes:

- **Identificador de trama:** Es empleado para la transacción, el servidor ModBus copia en la respuesta el identificador de la trama de la petición.

- **Identificador de protocolo:** Es empleado para los sistemas multiplexados. El protocolo ModBus es identificado por el valor 0.

- **Longitud:** Este campo es un contador de bytes de los siguientes campos, incluyendo el identificador de unidad y el campo de datos.
- **Identificador de unidad:** Este campo es empleado para identificar al esclavo remoto.

1.3.8 Proceso de Comunicación ModBus [11]

La unidad de datos de la aplicación ModBus está construida por el cliente que inicia una transacción ModBus. La función indica al servidor qué tipo de acción realizar. La aplicación MODBUS establece el formato de una solicitud iniciada por un cliente. Como se muestra en la Figura 1.8.

El campo de código de función de una unidad de datos ModBus está codificado en un byte. Los códigos válidos están en el rango de 1-255 decimal (el rango 128 - 255 está reservado y se usa para respuestas de excepciones). Cuando se envía un mensaje de un cliente a un dispositivo de servidor, el campo de código de función le dice al servidor qué tipo de acción realizar. El código de función "0" no es válido. Los códigos de subfunción se agregan a algunos códigos de funciones para definir múltiples acciones.

El campo de datos de los mensajes enviados desde un cliente a los dispositivos del servidor contiene información adicional que el servidor usa para tomar la acción definida por el código de la función. Esto puede incluir elementos como direcciones discretas y de registro, la cantidad de elementos que se manejarán y el recuento de bytes de datos reales en el campo.

El campo de datos puede ser inexistente (de longitud cero) en ciertos tipos de solicitudes, en este caso el servidor no requiere ninguna información adicional. El código de función solo especifica la acción.

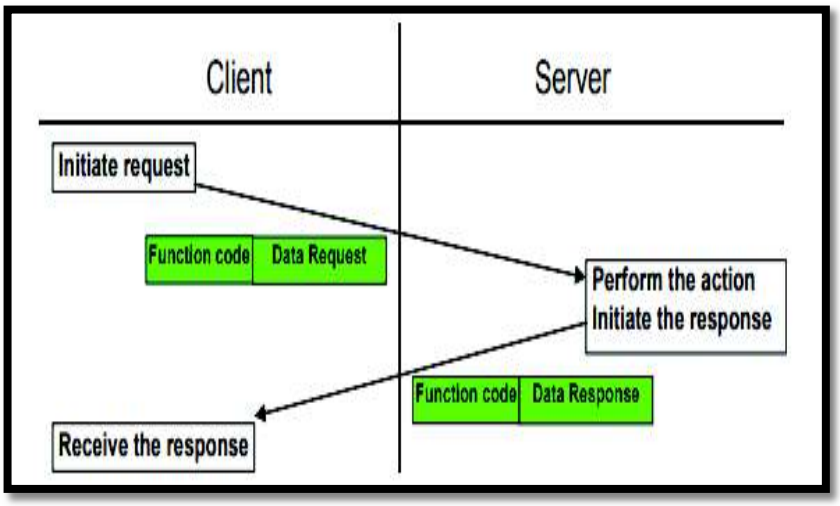


Figura 1.8. Transacción ModBus Libre de Errores

Para una respuesta de excepción, el servidor devuelve un código que es equivalente a la función original código de la PDU de solicitud con su bit más significativo puesto a lógica1. Lo descrito se observa en la Figura 1.9.

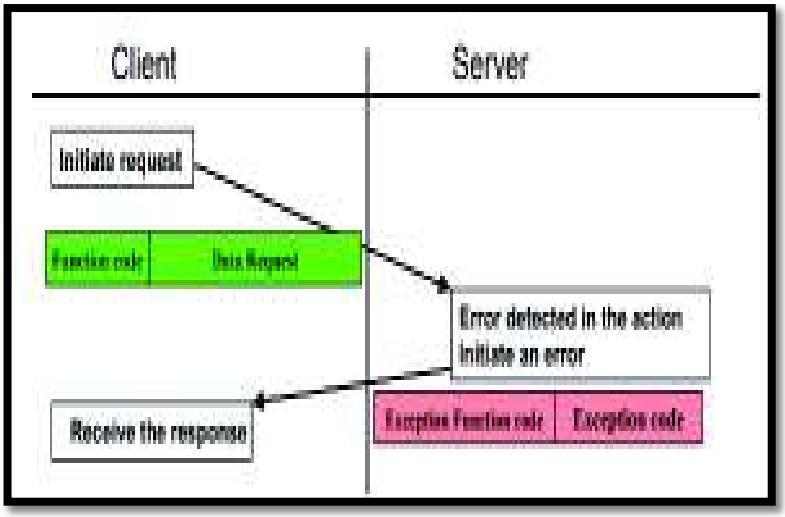


Figura 1.9. Transacción ModBus con Respuesta de Excepción

1.3.9 Acceso y Modelo de Datos en ModBus [8]

Los datos disponibles por medio de ModBus son almacenados, en general, en uno de los cuatro bancos de datos o rangos de dirección: bobinas, entradas discretas, registros de retención y registros de entrada. Al igual que con gran parte de la especificación, los nombres pueden variar dependiendo de la industria o de la aplicación. Por ejemplo, los registros de retención pueden denominarse como registros de salida y las bobinas pueden denominarse como salidas digitales o discretas.

Estos bancos de datos definen el tipo y los derechos de acceso de los datos contenidos. Los dispositivos esclavos tienen acceso directo a estos datos, los cuales son alojados localmente en los dispositivos. Los datos disponibles por medio de ModBus generalmente son un subconjunto de la memoria principal del dispositivo. En contraste, los maestros ModBus deben solicitar el acceso a estos datos a través de diversos códigos de función. El comportamiento de cada bloque se describe en la Tabla 1.2.

Tabla 1.2. Bloques de Modelo de Datos de ModBus

Bloque de Memoria	Tipo de Datos	Acceso de Maestro	Acceso de Esclavo
Bobinas	Booleano	Lectura/Escritura	Lectura/Escritura
Entradas Discretas	Booleano	Solo Lectura	Lectura/Escritura
Registros de Retención	Palabra Sin Signo	Lectura/Escritura	Lectura/Escritura
Registros de Entrada	Palabra Sin Signo	Solo Lectura	Lectura/Escritura

1.3.10 Códigos de Función [8]

En contraste con el modelo de datos que puede variar considerablemente de un dispositivo a otro, los códigos de función y sus datos son definidos explícitamente por el estándar. Cada función sigue un patrón. Primero, el esclavo valida entradas como el código de función, dirección de datos y el rango de datos. Después, ejecuta la acción solicitada y envía una respuesta adecuada al código. Si cualquier paso en este proceso falla, se regresa una excepción al solicitante. El transporte de datos para estas solicitudes es la PDU.

Debido a que el tamaño del paquete está limitado a 253 bytes, los dispositivos están limitados a la cantidad de datos que pueden ser transferidos. Los códigos de función más comunes pueden transferir entre 240 y 250 bytes de datos del modelo de datos de esclavos, dependiendo del código. Estos códigos pueden ser los siguientes.

a. Códigos Clase 0

Los códigos Clase 0 generalmente son considerados el mínimo para un dispositivo ModBus útil, ya que dan a un maestro la habilidad de leer o escribir en el modelo de datos. Los códigos de la clase 0 se presentan en la Tabla 1.3.

Tabla 1.3. Compatibilidad con Códigos Clase 0

Código	Descripción
3	Leer Múltiples Registros
16	Escribir a Múltiples Registros

b. Códigos Clase 1

Los códigos de función Clase 1 consisten en los otros códigos necesarios para tener acceso a todos los tipos del modelo de datos. En la definición original, esta lista incluye el código de función 7 (leer excepción). Sin embargo, este código es definido por la especificación actual como un código para serial únicamente, estos códigos se muestran en la Tabla 1.4.

c. Códigos Clase 2

Los códigos de función Clase 2 son funciones más especializadas que son implementadas con menos frecuencia. Por ejemplo, Leer/Escribir Múltiples Registros puede ayudar a reducir el número total de ciclos de solicitud-respuesta, pero el comportamiento aún puede ser implementado con códigos Clase 0, estos códigos se muestran en la Tabla 1.5.

Tabla 1.4. Compatibilidad con Códigos Clase 1

Código	Descripción
1	Leer Bobinas
2	Leer Entradas Discretas
4	Leer Registros de Entrada
5	Escribir a Bobina Individual
6	Escribir a Registro Individual
7	Leer Estado de Excepción (únicamente serial)

Tabla 1.5. Compatibilidad con Códigos Clase 2

Código	Descripción
15	Escribir a Múltiples Bobinas
20	Leer Registro de Archivo
21	Escribir a Registro de Archivo
22	Escribir a Registro con Máscara
23	Leer/Escribir Múltiples Registros
24	Leer FIFO

1.3.11 Ventajas del Protocolo ModBus/TCP [3]

a. Simplicidad

ModBus TCP/IP simplemente toma el conjunto de instrucciones ModBus y envuelve TCP/IP a su alrededor. Si ya se tiene un controlador ModBus y se entiende las conexiones Ethernet y TCP / IP, se puede tener un controlador en funcionamiento y hablando con una PC en unas pocas horas.

b. Costos

Los costos de desarrollo son bajos. Se requiere hardware mínimo y el desarrollo es sencillo en cualquier sistema operativo.

c. Estándar Ethernet

No se requieren *chipsets* especiales y puede usar tarjetas Ethernet estándar de PC para hablar con su dispositivo recientemente implementado. A medida que disminuye el costo de Ethernet, se beneficia de la reducción del precio del hardware y, a medida que el rendimiento mejora de 10 a 100 Mbps y luego a 1 Gbps, su tecnología avanza, protegiendo su inversión. Ya no se debe estar vinculado a un proveedor por soporte, sino que se beneficia de los miles de desarrolladores que están haciendo de Ethernet e Internet las

herramientas de red del futuro. Este esfuerzo se ha complementado con la asignación del bien conocido puerto Ethernet 502 para el protocolo ModBus TCP / IP.

c. Protocolo abierto

El protocolo ModBus fue transferido de Schneider Electric a la Organización ModBus en abril de 2004, lo que indica un compromiso con la apertura. La especificación está disponible sin cargo para descargar, y no se requieren tarifas de licencia posteriores para usar los protocolos ModBus o ModBus TCP/IP.

d. Disponibilidad de muchos dispositivos

La interoperabilidad entre dispositivos de diferentes proveedores y la compatibilidad con una gran base instalada de dispositivos compatibles con ModBus hace que ModBus sea una buena opción.

1.3.12 Arquitectura Maestro Esclavo [8]

ModBus es un protocolo de solicitud-respuesta implementado usando una relación Maestro-Esclavo.

En una relación maestro-esclavo, la comunicación siempre se produce en pares, un dispositivo debe iniciar una solicitud y luego esperar una respuesta y el dispositivo de inicio (el maestro) es responsable de iniciar cada interacción, este proceso se observa en la Figura 1.10.

Por lo general, el maestro es una HMI (Interfaz Humano-Máquina) o sistema SCADA[9] y el esclavo es un sensor, PLC o PAC (Controlador de Automatización Programable). El contenido de estas solicitudes y respuestas, y las capas de la red a través de las cuales se envían estos mensajes, son definidos por las diferentes capas del protocolo[8].

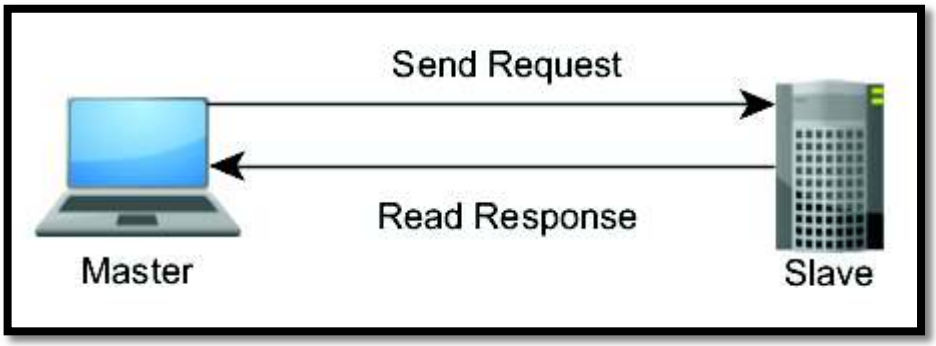


Figura 1.10 Una Relación de Red Maestro-Esclavo

[9] SCADA (*Supervisory Control and Data Adquisition*, es decir, Supervisión, Control y Adquisición de Datos) no es una tecnología concreta sino un tipo de aplicación[12].

El paradigma Maestro/Esclavo o Cliente/Servidor es quizás el más conocido de los paradigmas para aplicaciones de red. Se usa para describir un modelo de interacción entre dos procesos. Este modelo es una comunicación basada en una serie de preguntas y respuestas, que asegura que si dos aplicaciones intentan comunicarse, una comienza la ejecución y espera indefinidamente que la otra le responda y luego continua con el proceso como se muestra en la Figura 1.11.

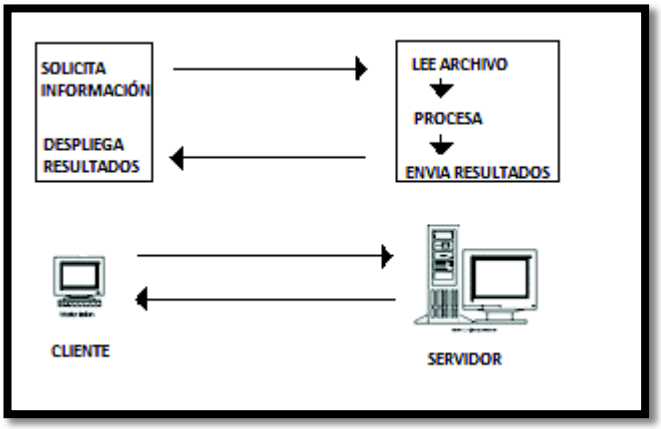


Figura 1.11. Paradigma Cliente/Servidor

1.3.13 Funciones de ModBus TCP [13]

Función 1 (*Read Coil Status*), Función 2 (*Read Input Status*)

Permite realizar la lectura del estado de las *Input Status* (Entradas Digitales) o *Coil Status* (Salidas Digitales). Para ello el Maestro solicita el número de bits que desea leer a partir de una determinada dirección. Cada dirección se corresponde con un registro de 1 bit con el estado de la entrada digital. El esclavo responde indicando el número de bits que retorna y sus valores. En la trama de respuesta se aprovechan todos los bits del byte, y puede haber hasta 256 bytes.

En la Figura 1.12 se muestra un ejemplo de la estructura de una trama de una petición del maestro hacia el esclavo y la respuesta desde el esclavo hacia el maestro.

Petición del maestro	**Respuesta del esclavo**
NºEsclavo	NºEsclavo
Código Operación: 0x01 o 0x02	Código Operación: 0x01 o 0x02
Dirección del registro (de 1 bit) a leer H	Nº bytes leídos: 1 byte
Dirección del registro (de 1 bit) a leer L	Octetos: max 256 bytes
Nº de bits que se desea leer H	CRC (16): H, L
Nº de bits que se desea leer L	
CRC (16): H,L	

Figura 1.12. Formato Petición/Respuesta Funciones 1 y 2

Función 3 (*Read Holding Registers*), Función 4 (*Read Input Registers*)

Permite realizar la lectura del valor de los *Input Registers* (Registros de Entrada) o *Holding Registers (*Registros de Salida). El maestro indica la dirección base y número de palabras a leer a partir de esta, mientras que el esclavo indica en la respuesta el número bytes retornados, seguido de estos valores. Aunque en realidad se está escribiendo en el rango de registros o valores numéricos, los registros son direccionados a partir de la dirección 0 (así el registro @40001 se direcciona 0).

En la Figura 1.13 se muestra un ejemplo de la estructura de una trama de una petición del maestro hacia el esclavo y la respuesta desde el esclavo hacia el maestro.

Petición del maestro	**Respuesta del esclavo**
NºEsclavo Código Operación: 0x03 Dirección del registro Nº de datos que se desea leer: max 128 datos CRC (16): H L	NºEsclavo Código Operación: 0x03 Nº de bytes leídos: 1 byte Datos: Max 128 datos CRC (16): H L
Petición: [0A][03][00][00][00][0A][71][76] 0A numero de esclavo 03 función de lectura 00 00 registro donde se va a comenzar la lectura 00 0A número de registros a leer: 10 71 76 CRC	**Respuesta:**[0A][03][14][00][00][08][4D][00][00][23][28][00][00][0F] [A0][00][00][00][90][00][00][00][60][CB][2E] 0A Número del esclavo que responde, 10 en decimal 03 Función de lectura - la que se ha utilizado en la pregunta 14 Número de bytes recibidos (20). 00 00 08 4D 2125 en decimal 00 00 23 28 9000 en decimal 9000 00 00 0F A0 4000 en decimal 00 00 00 90 144 en decimal 00 00 00 60 96 en decimal CB 2E CRC

Figura 1.13. Formato Petición/Respuesta Función 3

Función 15 *(Force Multiple Coil)*

Permite la modificación simultánea de varias Salidas Digitales en el esclavo, pasando del estado OFF (‘0’) o a ON (‘1’) según convenga. Así en el comando se pasan la dirección inicial (dirección del primer bit o mando a modificar) y la cantidad y estado de cada uno de los sucesivos mandos (bits) a modificar.

Aunque el estado de las Salidas Digitales se especifica bit a bit, las tramas se componen de bytes, y esto obliga a enviar los estados en grupos de 8. El esclavo no debería hacer caso a los bits sobrantes, es decir, no debería considerar los que queden por encima del

último bit indicado en el campo “cantidad de mandos a modificar”. Así, si quisiéramos modificar 12 mandos o relés a partir de la dirección 7, indicaríamos como dirección origen la dirección 7, como cantidad de mandos a modificar 12, y en el campo de estado de mandos: 0x3C , 0x0B (el esclavo no considerará los que queden por encima del 12o bit).

En la Figura 1.14 se muestra un ejemplo de la estructura de una trama de una petición del maestro hacia el esclavo y la respuesta desde el esclavo hacia el maestro.

Petición del máster (modo RTU):	**Respuesta del esclavo (modo RTU):**
NºEsclavo	NºEsclavo
Código Operación: 0x0F	Código Operación: 0x0F
Dirección inicial de los mandos (bits) H	Dirección inicial de los mandos o bits consecutivos H
Dirección inicial de los mandos (bits) L	Dirección del mando o bit L
Cantidad de mandos (bits) H	Cantidad de mandos o bits H
Cantidad de mandos (bits) L	Cantidad de mandos o bits L
Cantidad de bytes enviados con el estado de los mandos (bits)	CRC(16): H L
Estado de los 8 primeros mandos (bits) a modificar	
Estado de los 8 siguientes mandos (bits) a modificar	
…	
Estado de los 8 últimos mandos (bits) a modificar	
CRC(16): H L	

Figura 1.14. Formato Petición/Respuesta Función 15

Función 16 (*Preset Multiple Registers*)

Permite realizar la escritura en un grupo de Registros de Salida. Se debe especificar la dirección a partir de la que se quiere actualizar los valores, y la lista de valores que se quiere asignar a estos registros. Aunque se está escribiendo en el rango de registros o valores numéricos, los registros son direccionados a partir de la dirección 0 (es decir el registro @40001 se direcciona 0).

En la Figura 1.15 se muestra un ejemplo de la estructura de una trama de una petición del maestro hacia el esclavo y la respuesta desde el esclavo hacia el maestro.

Petición del maestro:	**Respuesta del esclavo:**
NºEsclavo	NºEsclavo
Código Operación: 0x10 (16 en decimal)	Código Operación: 0x10 (16 en decimal)
Dirección base de los datos: 2 bytes	Dirección base de los datos: 2 bytes
Número de datos: 2 bytes	Número de datos: 2 bytes
Valor del dato 0: 2 bytes	CRC (16): H L
Valor del dato 1: 2 bytes	
…	
Valor del dato n: 2 bytes	
CRC (16): H L	

Figura 1.15. Formato Petición/Respuesta Función 16

1.3.14 Sockets [14]

Normalmente, un servidor se ejecuta en una máquina específica y tiene un *socket* asociado a un número de puerto específico. El servidor simplemente espera a la escucha en el socket a que un cliente se conecte con una petición como se muestra en la Figura 1.16. El cliente conoce el nombre de la máquina sobre la que está ejecutándose el servidor y el número de puerto al que está conectado. Solicitar una conexión consiste en intentar establecer una cita con el servidor en el puerto de la máquina servidora.

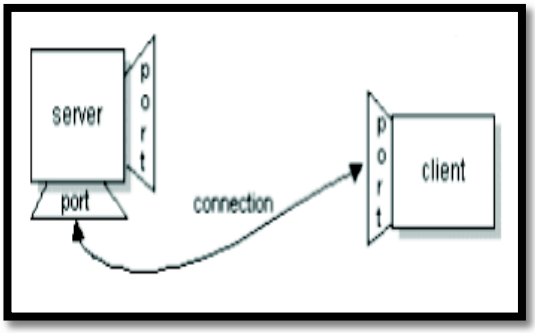


Figura 1.16. Socket Servidor

El servidor acepta la conexión, pero antes, el servidor crea un nuevo *socket* en un puerto diferente como se muestra en la Figura 1.17. Es necesario crear un nuevo *socket* (y consecuentemente un número de puerto diferente) de forma que en el *socket* original se continúe a la escucha de las peticiones de nuevos clientes mientras se atiende a las necesidades del cliente conectado. En el cliente, si se acepta la conexión, el *socket* se crea satisfactoriamente y se puede utilizar para comunicarse con el servidor.

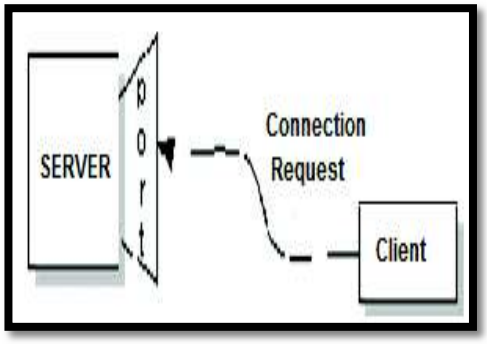


Figura 1.17. Socket Cliente

Un *socket* es el extremo final de un enlace punto-a-punto que comunica a dos programas ejecutándose en una red. Los *sockets* siempre están asociados a un número de puerto que es utilizado por TCP para identificar la aplicación a la que está destinada la solicitud y poder redirigírsela.

Clase Socket y ServerSocket

La clase Socket del paquete java.net es fácil de usar comparada con la que proporcionan otros lenguajes. Java oculta las complejidades derivadas del establecimiento de la conexión de red y del envío de datos a través de ella. En esencia, el paquete java.net proporciona la misma interfaz de programación que se utiliza cuando se trabaja con archivos[15].

La clase ServerSocket es la que se utiliza a la hora de crear servidores, de la misma manera como la clase Socket se utiliza para crear clientes.

1.3.15 Software Utilizado para Control y Monitoreo de Datos Intouch [16]

Wonderware InTouch es el HMI y software de visualización de procesos más avanzado y conocido en el mundo. Ofrece una innovación de primer nivel, gráficos nítidos, facilidad de uso y una conectividad de muy alto nivel.

InTouch es la tecnología gráfica más sofisticada y el producto más intuitivo del mercado para visualización de procesos y gestión de datos.

Wonderware InTouch, permite que los usuarios puedan crear rápidamente aplicaciones estandarizadas y reutilizables de visualización e instalables con un solo clic en toda la empresa, incluyendo a los usuarios móviles.

InTouch viene equipado con una completa biblioteca de símbolos gráficos y carátulas previamente construidos y comprobados que contienen más de 500 símbolos gráficos *ArchestrA* diseñados de manera profesional, en su mayoría con “inteligencia” personalizable ya incorporada, que proporciona el acceso en modo “arrastrar y soltar” a componentes de ingeniería previamente construidos.

InTouch permite desarrollar de forma rápida y sencilla vistas gráficas a medida de sus procesos en tiempo real.

InTouch es una interfaz HMI que ayuda a generar sistemas SCADA. Para esto utiliza dos partes fundamentales de su sistema: *WindowMaker* y *WindowViewer.*

WindowMaker es donde se desarrolla el HMI, como se dijo anteriormente InTouch cuenta con una biblioteca de símbolos gráficos predeterminados, lo cual hace más factible la construcción del HMI como tal se torne un trabajo sencillo.

Para ello solo se arrastran los gráficos predeterminados y se asocia cada uno de sus parámetros con las variables de un dispositivo que esté formando parte de la red industrial acerca de la cual se está desarrollando el HMI.

Antes de poder asociar las variables de los componentes del HMI con las variables de los dispositivos de la red industrial tales como: PLC, sensores, variadores de velocidad, etc; se debe registrar los dispositivos utilizando la herramienta SMC (*System Managment Control*) que es un programa que viene en conjunto con *InTouch*.

En SMC se puede registrar cada uno de los dispositivos dentro del sistema *InTouch*, para ello utilizamos su dirección IP. Aquí se puede configurar características como: nombre, *Gateway* y parámetros para la comunicación con los demás dispositivos de la red.

Una vez registrados los dispositivos, en la parte de *WindowMaker* se crea automáticamente un "*access name*" con el mismo nombre con el cual se cargó el dispositivo en el SMC y así de esta manera ahora ya se puede asociar el esclavo directamente con una de las variables del HMI que se esté desarrollando.

1.3.16 Metodología Kanban [17, p. 1]

Desarrollo Ágil con Kanban

Kanban, llega como metodología de gestión de proyectos, de la mano de la automotriz Toyota, representando estadísticamente, la metodología ágil que menor resistencia presenta en las compañías acostumbradas a las metodologías tradicionales. La palabra Kanban, de origen japonés, se compone de dos términos: Kan que puede traducirse como "visual" y ban, como "insignia", siendo una traducción aproximada, "insignia visual". Un modelo de Tablero Kanban se muestra en la Figura 1.18.

Orígenes: De Toyota al Software

Kanban se basa en un sistema de producción que dispara trabajo solo cuando existe capacidad para procesarlo. El disparador de trabajo es representado por tarjetas kanban de las cuales se dispone de una cantidad limitada.

Cada tarjeta Kanban acompaña a un item de trabajo durante todo el proceso de producción, hasta que éste, es empujado fuera del sistema, liberando una tarjeta. Un nuevo ítem de trabajo, solo podrá ser ingresado/aceptado si se dispone de una tarjeta kanban libre.

Este proceso de producción, donde un trabajo se introduce al sistema solo cuando existe disponibilidad para procesarlo, se denomina pull (tirar) en contrapartida al mecanismo push (empujar), donde el trabajo se introduce en función de la demanda.

En el desarrollo de Software, Kanban fue introducido por David Anderson de la Unidad de Negocios XIT de Microsoft, en 2004, reemplazando el sistema de tarjetas por un tablero visual similar al de Scrum, pero con características extendidas que se muestra a continuación.

Características Principales de Kanban [18, p. 5]

1. Visualiza el flujo de trabajo

- Divide el trabajo en bloques, escribe cada elemento en una tarjeta y ponlo en el muro.
- Utiliza columnas con nombre para ilustrar dónde está cada elemento en el flujo de trabajo.

2. Limita el WIP (*Work in Progress*)

Asigna límites concretos a cuántos elementos pueden estar en progreso en cada estado del flujo de trabajo. ·

3. Mide el *lead time*

Tiempo medio para completar un elemento, a veces llamado "tiempo de ciclo" optimiza el proceso para que el lead time sea tan pequeño y predecible como sea posible.

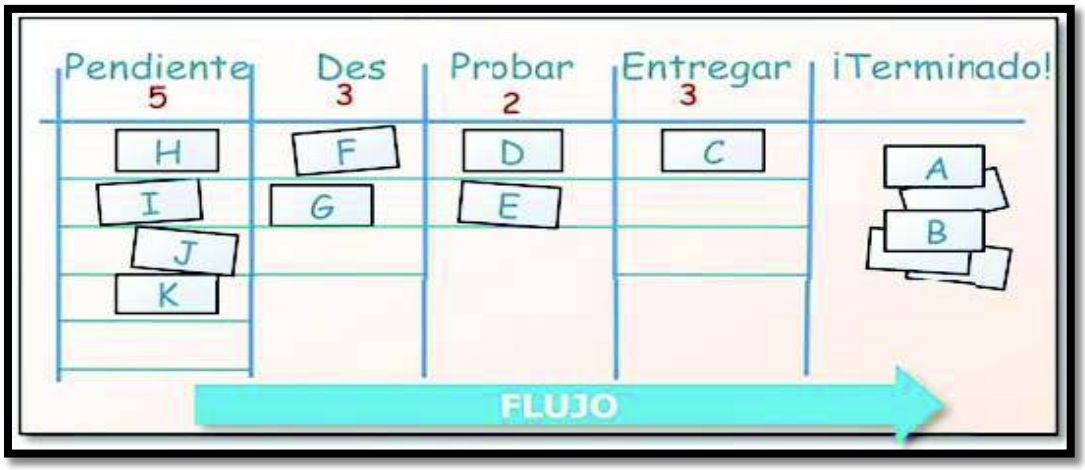


Figura 1.18. Tablero Kanban

Trello [19]

Trello es una aplicación basada en el método Kanban y sirve para gestionar tareas permitiendo organizar el trabajo en grupo de forma colaborativa mediante tableros virtuales compuestos de listas de tareas en forma de columnas.

Es utilizada para la gestión de proyectos ya que se pueden representar distintos estados y compartirlas con diferentes personas que formen el proyecto. Con ella se intenta mejorar las rutinas de trabajo de un equipo generando prioridades, tiempos, avisos y otras opciones perfectas para organizar un proyecto en el que colaboran varias personas.

Trello se basa en el método Kanban donde se utilizan las fases *To Do*, *Doing* y *Done*.

Esta organización hace gestionar bien el desarrollo de un proyecto y con Trello basta con listar todas las tareas que componen un proyecto e ir colocándolas en tres columnas según su estado.

- 1ª columna: To Do – Por hacer.
- 2ª columna: Doing – Haciéndose.
- 3ª columna: Done – Hecho.

1.3.17 Herramientas para el Desarrollo de Software

Historias de Usuario

Según [20]: El énfasis en la creación de sistemas de usuario está en la interacción hablada entre desarrolladores y usuarios, no en la comunicación escrita. En las historias de usuario, el desarrollador ante todo busca identificar los requerimientos valiosos del usuario de negocios. Generalmente los usuarios estarán ocupados diariamente en las conversaciones con los desarrolladores sobre el significado de las historias de usuario que han escrito. Estas conversaciones frecuentes son interacciones determinadas que tienen como su meta la prevención de malos entendidos o malas interpretaciones de los requerimientos del usuario. Por lo tanto las historias de usuario sirven como recordatorios para los desarrolladores que deben sostener conversaciones que deben sostener conversaciones seguras para dichos requerimientos.

Las características de las historias de usuario son:

- Independientes unas de otras.
- Negociables.
- Valoradas por los clientes o usuarios.
- Estimables.
- Pequeñas.
- Verificables.

Las Historias de Usuario se basan en una regla de palabras.

COMO <rol>
QUIERO <Acción>
PARA <Funcionalidad>
A esto se lo llama comúnmente “Enunciado de la Historia de Usuario” [20, p. 29].

Basándose en el concepto anterior sobre una historia de usuario se desarrolló el siguiente formato presentado en la Figura 1.19 para utilizarlo en el desarrollo de este prototipo de software.

Figura 1.19. Formato de Historia de Usuario.

El cual será detallado a continuación.

ID historia de usuario: Indica un identificador único de historia de usuario

Como: rol

Quiero: Acción

Para: Funcionalidad

Validación: indica la manera de como validar la funcionalidad de dicho módulo.

a. Diagramas

Los diagramas en el desarrollo de software son de vital importancia ya que con ellos se puede representar acciones e interacciones entre distintas abstracciones en el sistema. En este caso para el desarrollo del software se utilizarán cuatro tipos de diagramas, los cuales se describen a continuación.

a.1. Diagrama de Clase

Un diagrama de clase representa los propósitos fundamentales de UML (*Unified Modeling Language*, es decir Lenguaje Unificado de Modelado) ya que con el se puede separar los elementos de diseño de la codificación del sistema. UML ha sido establecido como un modelo estandarizado para describir un enfoque de programación orientado a objetos. Dado que las clases son el bloque de construcción de los objetos, los diagramas de clase son los bloques de construcción de UML. Los componentes de creación de diagramas en un diagrama de clase pueden representar las clases que realmente van a ser programadas, los objetos principales, o las relaciones entre clases y objetos.

a.2. Diagrama de Casos de Uso [21]

Los casos de uso son una técnica para especificar el comportamiento de un sistema. Un caso de uso es una secuencia de interacciones entre un sistema y alguien o algo que usa alguno de sus servicios. Entonces un caso de uso, es una forma de expresar cómo alguien o algo externo a un sistema lo usa. Cuando dice “alguien o algo” se hace referencia a que los sistemas son usados no sólo por personas, sino también por otros sistemas de hardware y software.

Los casos de uso son representados a través de diagramas, estos diagramas muestran, la forma en como ese “alguien o algo” opera con el sistema y el orden en como los elementos interactúan (operaciones o casos de uso).

Un diagrama de casos de uso consta de los siguientes elementos.

- **Actor:** Una definición previa, es que un Actor es un rol que un usuario juega con respecto al sistema. Es importante destacar el uso de la palabra rol, pues con esto se especifica que un Actor no necesariamente representa a una persona en particular, sino más bien la labor que realiza frente al sistema.

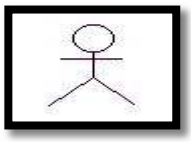


Figura 1.20. Representación Gráfica de un actor

- **Caso de Uso:** Es una operación/tarea específica que se realiza tras una orden de algún agente externo, sea desde una petición de un actor o bien desde la invocación desde otro caso de uso.

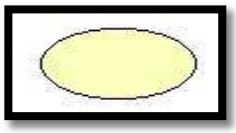


Figura 1.21. Representación Gráfica de Caso de Uso

- **Relaciones**
  - **Asociación:** Es el tipo de relación más básica que indica la invocación desde un actor o caso de uso a otra operación (caso de uso). Dicha relación se denota con una línea simple.

Figura 1.22. Representación Gráfica de Asociación

- **Dependencia o Instanciación:** Es una forma muy particular de relación entre clases, en la cual una clase depende de otra, es decir, se instancia (se crea). Dicha relación se denota con una flecha punteada.

Figura 1.23. Representación Gráfica de Dependencia o Instanciación

- **Generalización:** Este tipo de relación es uno de los más utilizados, cumple una doble función dependiendo de su estereotipo, que puede ser de Uso (<<uses>>) o de Herencia (<<*extends*>>). Este tipo de relación está orientado exclusivamente para casos de uso (y no para actores).

  - ***Extends***: Se recomienda utilizar cuando un caso de uso es similar a otro (características).

  - **Uses**: Se recomienda utilizar cuando se tiene un conjunto de características que son similares en más de un caso de uso y no se desea mantener copiada la descripción de la característica. De lo anterior cabe mencionar que tiene el mismo paradigma en diseño y modelamiento de clases, en donde está la duda clásica de usar o heredar.

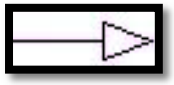


Figura 1.24. Representación Gráfica de Generalización

a.3. Diagrama de Actividades [21]

Un diagrama de Actividades consiste en mostrar el conjunto de actividades que ocurre durante un proceso, así mismo indica las posibles rutas que pueden irse desencadenando en el caso de uso o sirve para modelar la lógica detallada en una regla de negocio. También ayuda a auxiliar a los miembros del equipo de desarrollo a entender como es utilizado el sistema y cómo reacciona en determinados eventos. En muchos sentidos, diagramas UML de actividad son el equivalente orientado a objetos de los diagramas de flujo y diagramas de flujos de datos (DFD) de desarrollo estructurado, pero se puede decir que un diagrama de actividades describe el problema, mientras un diagrama de flujo describe la solución.

En la Figura 1.25 se presenta una representación gráfica de un diagrama de actividades.

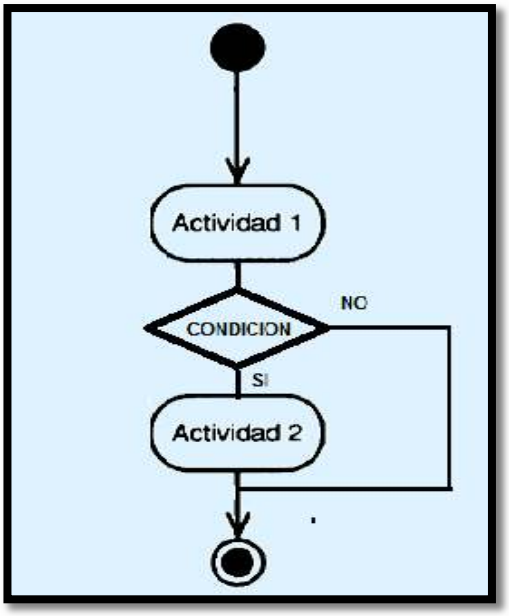


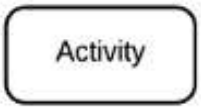
Figura 1.25. Diagrama de Actividades

Símbolos y notación para diagramas de actividades

A continuación se muestra un diagrama de actividades desglosado en sus elementos individuales.

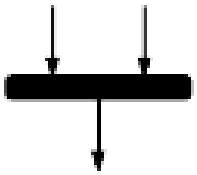

Un círculo negro es la notación estándar para un estado inicial antes de que transcurra una actividad. Lo puedes usar solo o puedes usar una nota para aclarar aún más el punto inicial.


El círculo negro similar a un botón de radio seleccionado es el símbolo UML para el estado final de una actividad. Como se muestra en dos ejemplos anteriores, también se pueden usar notas para explicar un estado final.

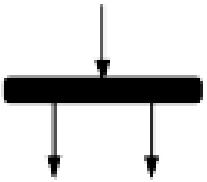


Los símbolos de actividades son los componentes básicos de un diagrama de actividades y comúnmente tienen una descripción corta de la actividad que representan.

Las flechas representan el flujo de dirección del diagrama de flujo. La flecha indica la dirección de las actividades en curso.



Una conjunción combina dos actividades simultáneas en un flujo en el que transcurre solo una actividad a la vez.

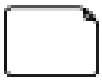


Una bifurcación divide el flujo de una actividad en dos actividades simultáneas.

El texto de condición se coloca al lado de un marcador de decisión para indicarte bajo qué condición un flujo de actividad debe bifurcarse en esa dirección.

Un marcador en forma de diamante es el símbolo estándar para una decisión. Siempre hay al menos dos caminos que salen de una decisión y el texto de condición te permite saber qué opciones se excluyen mutuamente.

El marcador de flujo final muestra el punto final para un proceso en un flujo. La diferencia entre un nodo de flujo final y un nodo de estado final es que este último representa el final de todos los flujos en una actividad.



La figura que se usa para notas.

a.4. Diagrama Entidad-Relación

Un diagrama Entidad-Relación (DER) es una herramienta de modelado de datos que describe cómo las "entidades", como personas, objetos o conceptos, se relacionan entre sí dentro de un sistema. Los diagramas ER se usan a menudo para diseñar o depurar bases de datos relacionales en los campos de ingeniería de software, sistemas de información empresarial, educación e investigación [22, p. 151].

Se emplean un conjunto definido de símbolos, tales como rectángulos, diamantes, óvalos y líneas de conexión para representar la interconexión de entidades, relaciones y sus atributos. Son un reflejo de la estructura gramatical y emplean entidades como sustantivos y relaciones como verbos. En la Figura 1.26 se muestran los elementos que conforman parte de un diagrama Entidad-Relación.

Figura 1.26. Diagrama Entidad-Relación

b. Pruebas

El objetivo de las pruebas es evaluar que el sistema desarrollado cumpla con los requerimientos definidos en la fase del diseño. Para este caso vamos a realizar los siguientes tipos de pruebas:

b.1 Pruebas Individuales de cada Módulo: Se verificará la funcionalidad de cada uno de los módulos definidos previamente, todo esto contrastando los requerimientos definidos en la fase de diseño.

b.2 Pruebas de Sistema: Una vez desarrollado y probado cada uno de los módulos se procederá a realizar una prueba del sistema completo, es decir funcionando todos los módulos a la vez.

b.3 Pruebas de Aceptación: Luego de realizar las pruebas individuales de los módulos y de todo el sistema se realizará pruebas en conjunto con las personas de la empresa Hieselat S.A, para verificar que se cumple con los requerimientos planteados.

2 METODOLOGÍA

2.1 Planteamiento del Tablero Kanban

En la Figura 2.1 se presenta las tareas y el proceso en el que se encuentran. En esta etapa de diseño toda tarea asociada con la fase de diseño y análisis de requerimientos se encuentra en la etapa de “tareas en proceso”, mientras que las demás tareas definidas para la fase de implementación y pruebas se encuentran en la columna de tareas por desarrollar.

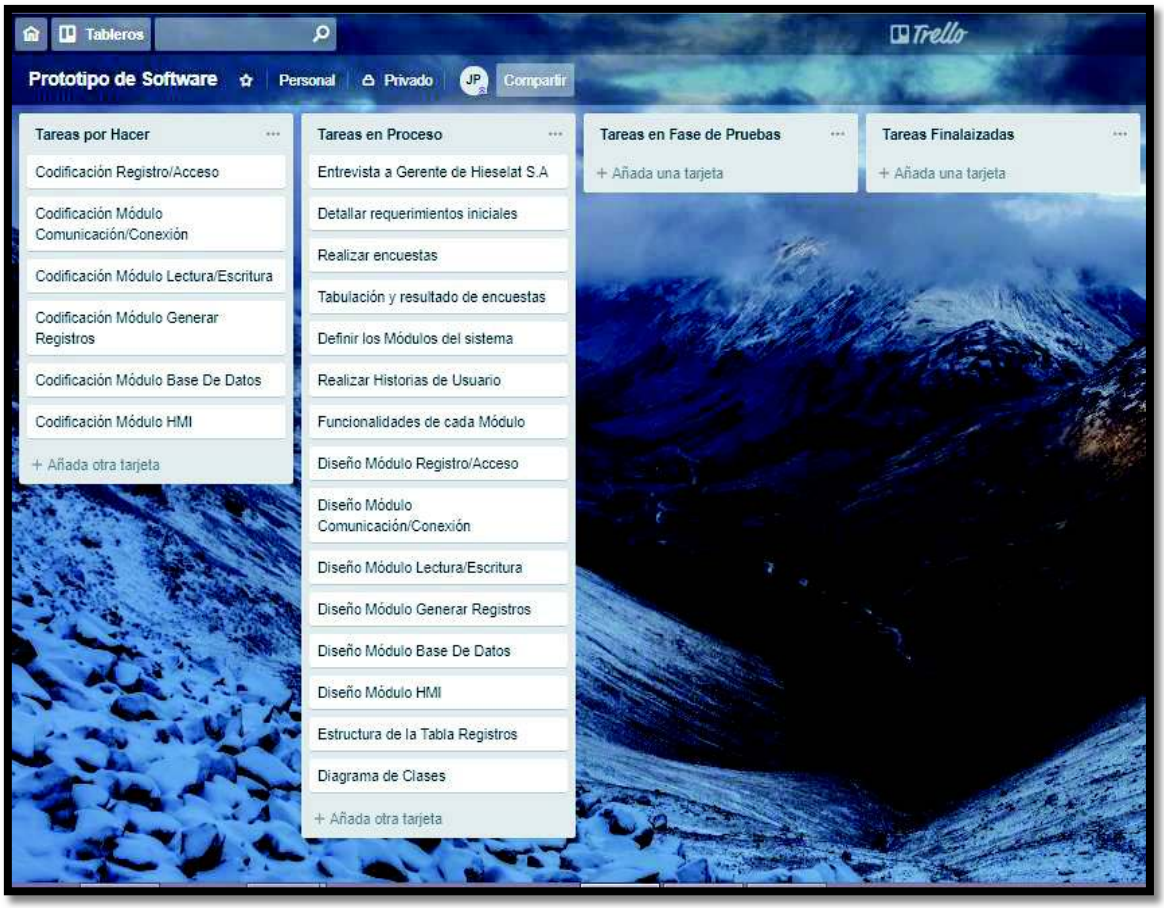


Figura 2.1. Planteamiento del Tablero Kanban

2.2 Requerimientos

2.2.1 Requerimientos Iniciales

Los requerimientos iniciales del sistema fueron planteados a partir de entrevistas, las cuales se realizaron en el mes de abril del 2018 al gerente general y equipo de ejecución de proyectos de la empresa HIESELAT S.A.

Después de haber analizado las entrevistas, los requerimientos obtenidos fueron.

a. Crear, modificar y eliminar usuarios, cada usuario creado debe tener su respectivo perfil para utilizar el prototipo de software; esta información estará almacenada en una base de datos. Se definieron dos perfiles.

   - **Administrador:** Este perfil de usuario tendrá acceso a todas las funcionalidades con las que cuenta el prototipo. Su rol consiste en, además de utilizar las funcionalidades de gestión de la información, administrar los usuarios y sus respectivos perfiles para manipular al prototipo.

   - **Operador:** Este perfil de usuario tendrá un acceso limitado ya que su principal trabajo consiste en el monitoreo (lectura de registros) y, si es el caso, la ejecución de secuencias de control que estén previamente configuradas.

b. Permitir el acceso al manejo y administración del prototipo mediante una interfaz de registro (Login), donde es indispensable contar con un nombre de usuario y una contraseña.

c. Contar con un sistema de recuperación de credenciales de usuario, a través del correo electrónico registrado al momento de crear el usuario.

d. Establecer la comunicación y conexión entre los dispositivos de la red y el prototipo de software mediante el protocolo industrial ModBus/TCP.

e. Leer los valores almacenados en los registros de cada dispositivo que forma parte de la red industrial.

f. Escribir un determinado valor en el o los registros de un dispositivo que este en la red industrial.

g. Generar registros que almacenen información importante acerca de los valores que poseen los registros de los dispositivos gestionados. En la Tabla 2.1 se muestra el formato inicial para almacenar la información dentro de la base de datos:

Tabla 2.1. Formato de información para registros

NOMBRE DE USUARIO	REGISTRO	VALOR DE REGISTRO	FECHA	HORA
Operador1	Nivel de agua	60	06/05/2017	10:30

La información será almacenada automáticamente en la base de datos con este formato, pasando como argumento los registros a monitorear: intervalo de muestreo y tiempo total de monitoreo.

h. Desarrollar una interfaz HMI para cubrir funciones específicas.
   Estas funciones son.
   - Control de temperatura
   - Controlar el nivel de un tanque
   - Controlar la velocidad de llenado del tanque (Caudal de entrada)
   - Visualizar el valor de las variables relacionadas en tiempo real

Al momento de realizar las historias de usuario el primer prototipo de la interfaz HMI es el que se muestra en la Figura 2.2.

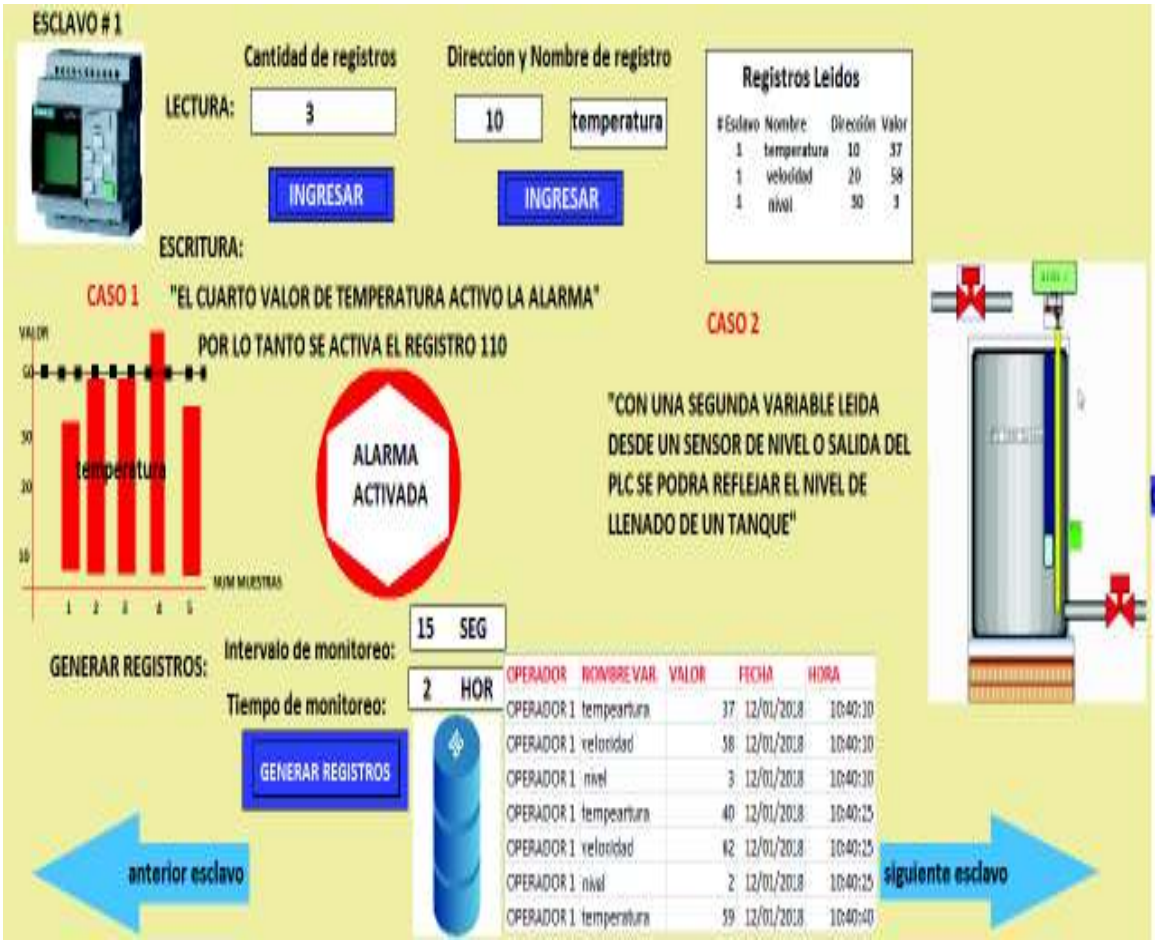


Figura 2.2. Prototipo Interfaz HMI

2.2.2 Análisis de Requerimientos Generales

Se elaboró una encuesta, la misma que fué realizada a 11 ingenieros; de la empresa Hieselat y a ingenieros que trabajan con software de control y monitoreo en el área ingeniería de control y automatización de procesos.

Con esto se obtuvo requerimientos más generales en comparación con los obtenidos en la primera entrevista con el ingeniero Jhonny Curimilma. Posterior a esto se definieron los

módulos con los que contará el sistema. Las 11 encuestas realizadas se encuentran en el Anexo A.

Una vez tabuladas las encuestas se obtiene los resultados que se presentan a continuación.

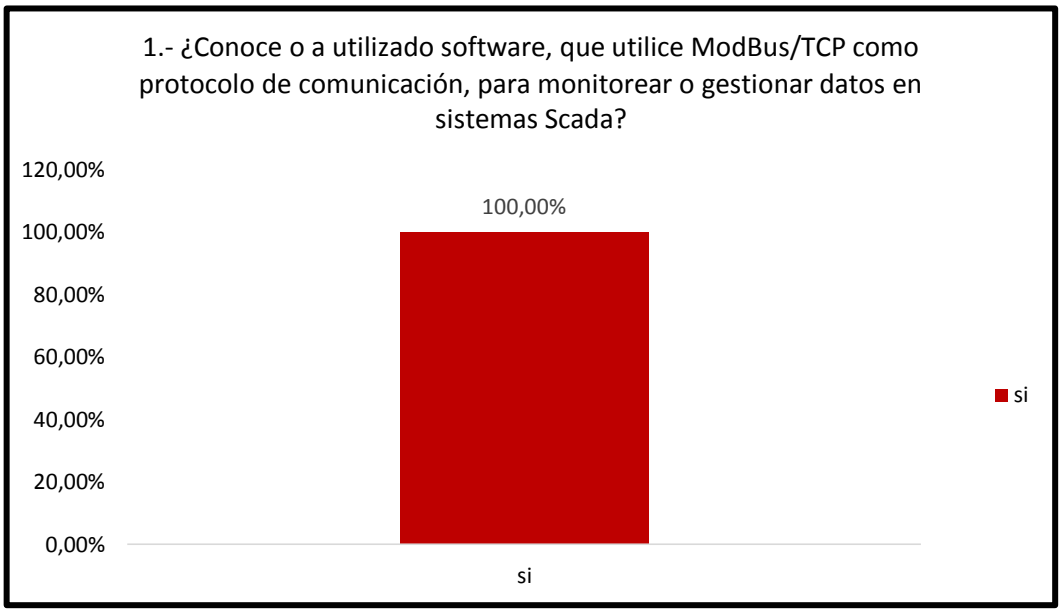


Figura 2.3. Resultados Pregunta 1

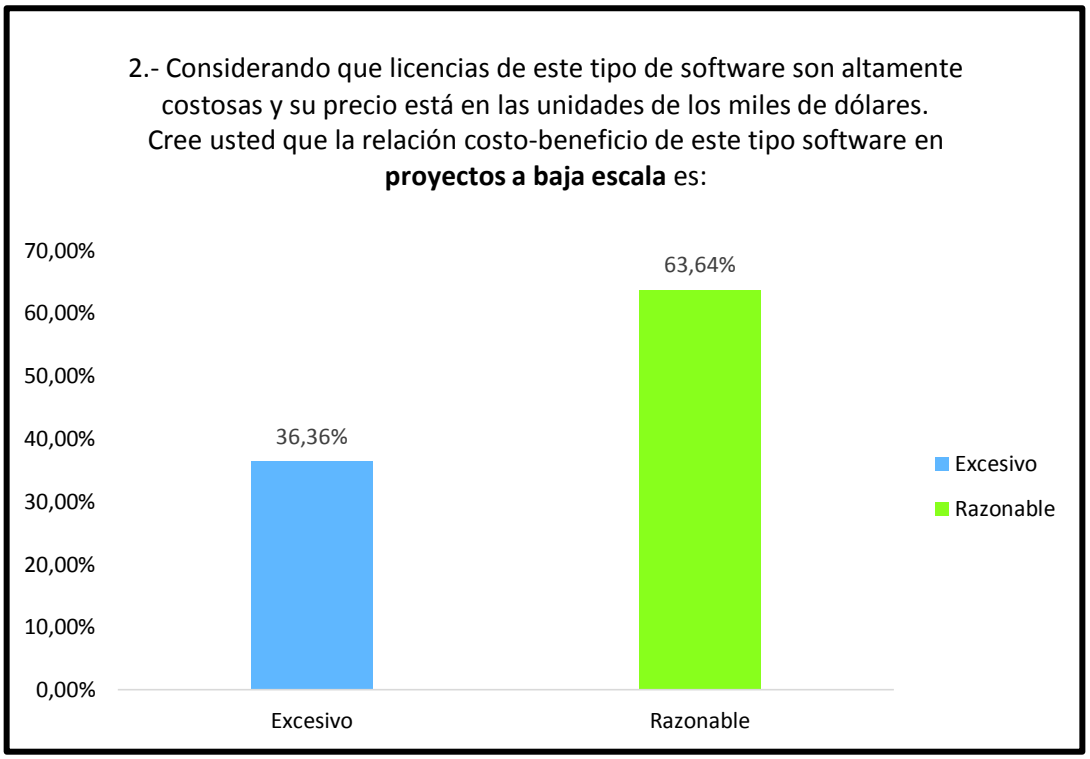


Figura 2.4. Resultados Pregunta 2

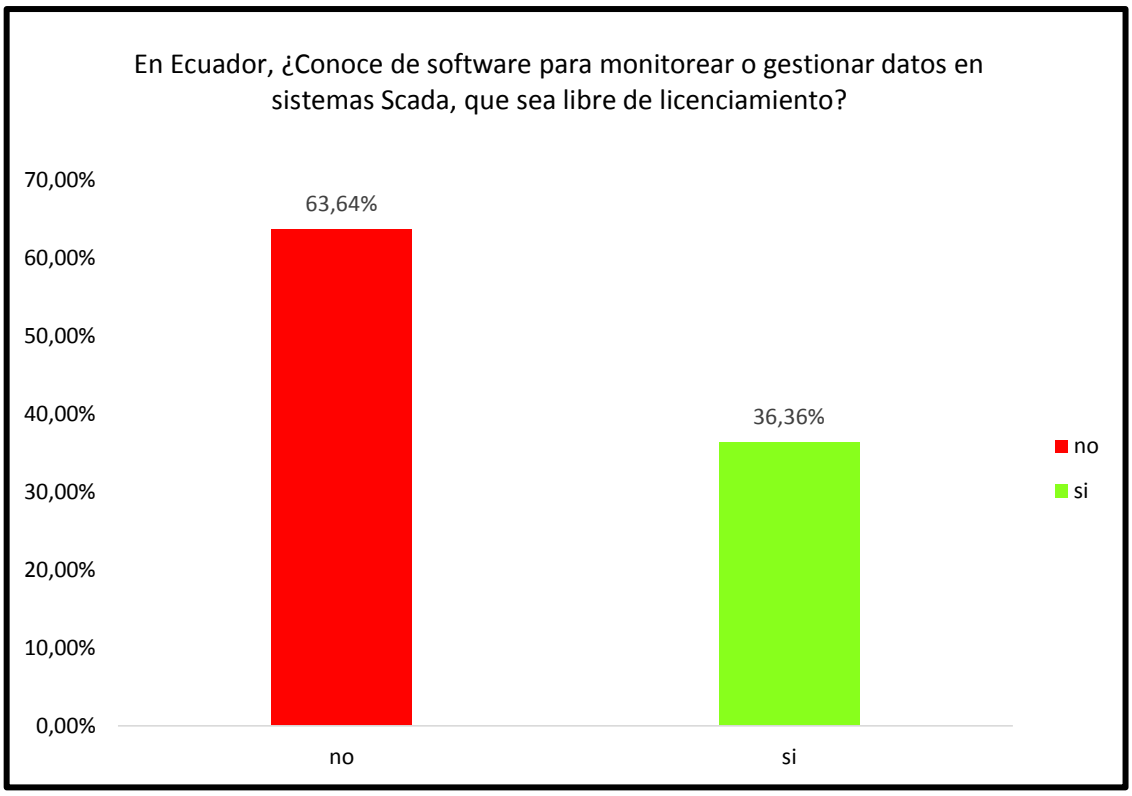


Figura 2.5. Resultados Pregunta 3

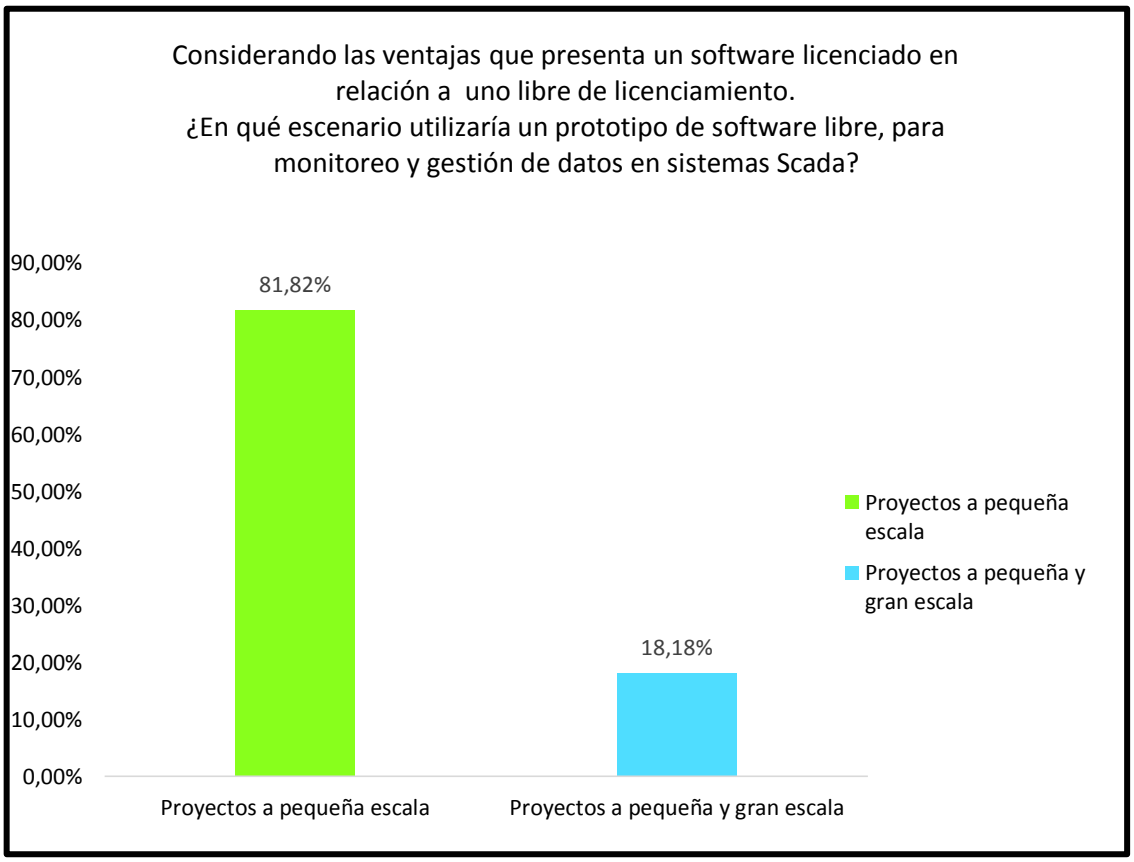


Figura 2.6. Resultados Pregunta 4

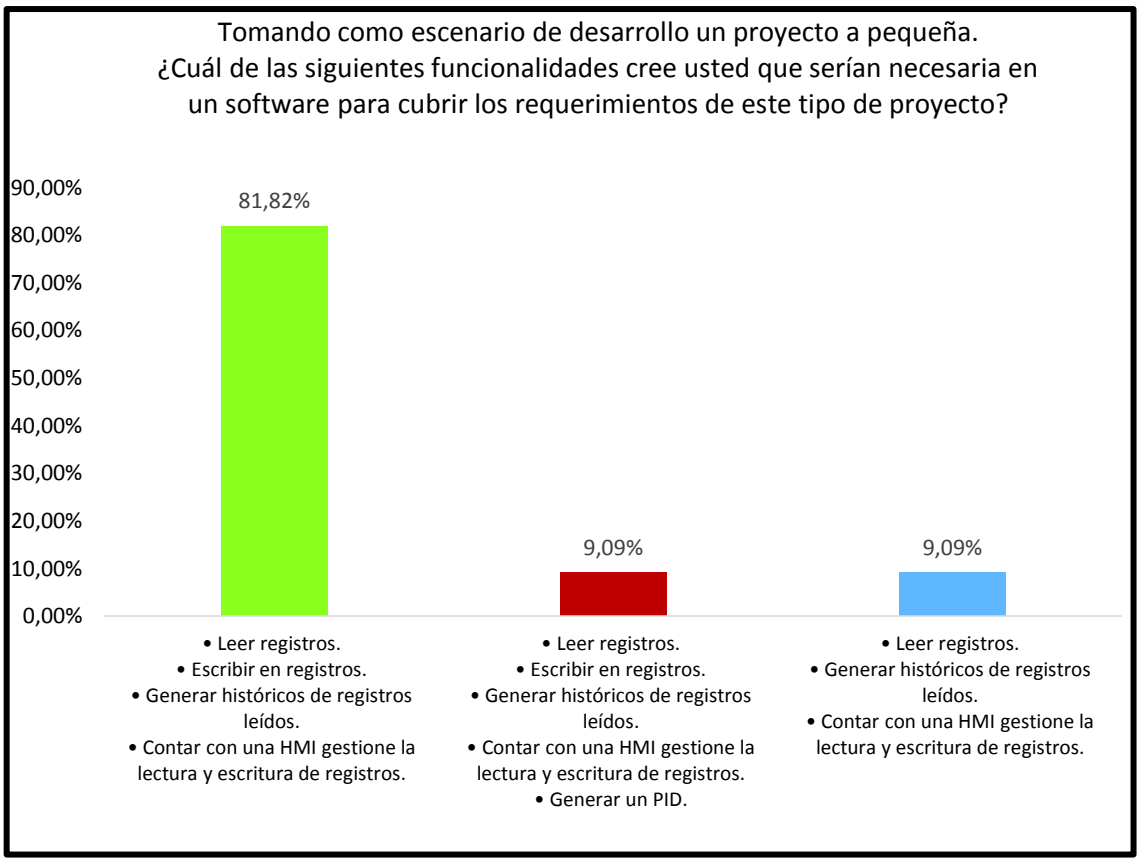


Figura 2.7. Resultados Pregunta 5

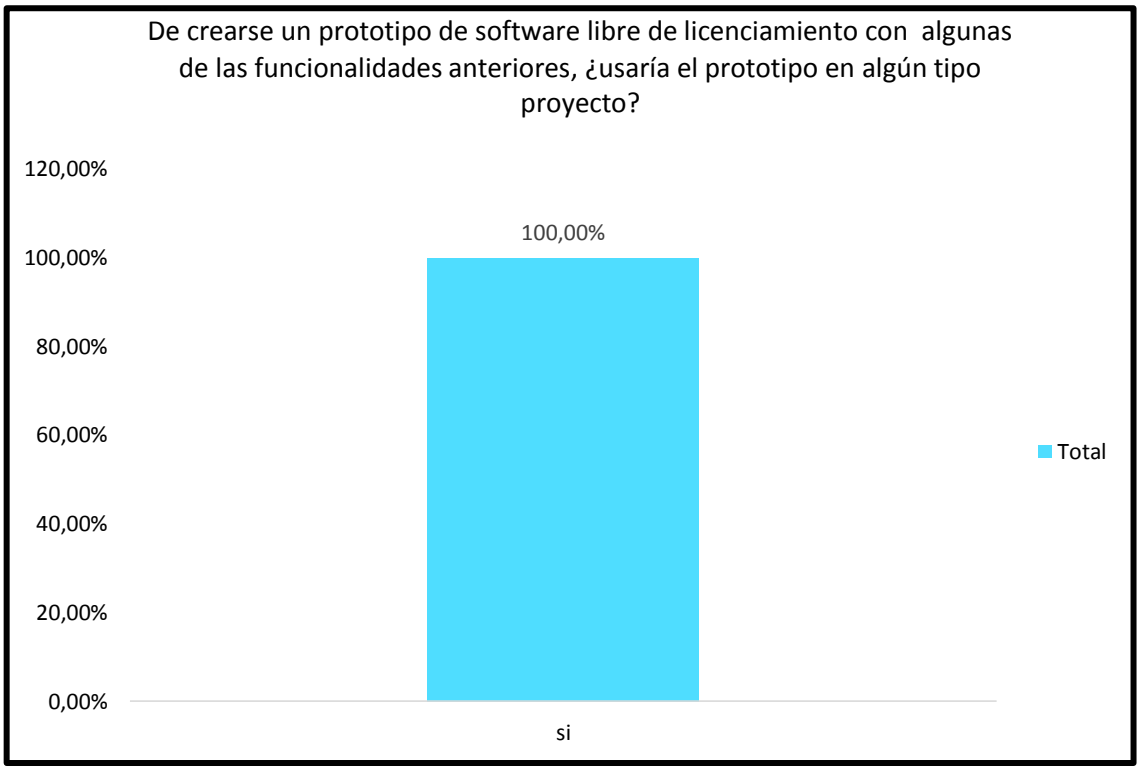


Figura 2.8. Resultados Pregunta 6

2.2.3 Módulos del Sistema

En base a los requerimientos obtenidos por las encuestas se definieron los módulos que se describen a continuación.

a. Módulo Registro/Acceso Usuarios

En este módulo se define un mecanismo de autenticación para acceder al prototipo y también la gestión de los roles de usuarios.

b. Módulo Comunicación/Conexión

Este módulo define la forma como se realizará la comunicación, mediante el protocolo ModBus/TCP, entre los dispositivos de la red y el prototipo. Además de definir los argumentos y secuencias de actividades para lograr establecer la conexión.

c. Módulo Lectura/Escritura

En este módulo se define las funciones para leer los registros de los equipos y también gestionar esta información mediante la escritura de los mismos.

d. Módulo Generar Registros

Este módulo permite definir una secuencia para generar registros de determinados valores obtenidos mediante la lectura desde los dispositivos. Además de definir el formato en que se guardará la información.

e. Módulo Base de Datos

En este módulo se define el modelo entidad/relación y la forma de conexión de la base de datos, donde se guardará tanto la información de los usuarios del sistema así como los registros generados.

f. Módulo HMI

Este módulo define la interfaz gráfica HMI la cual ayudará a los usuarios a poder gestionar la información que se obtiene de cada uno de los equipos de la red.

2.2.4 Historias de Usuario

Una vez establecidos los módulos, a partir de ellos se va a obtener los requerimientos específicos. Para esto se realizó el desarrollo de historias de usuario, en 6 reuniones diferentes: una para cada módulo, con el ingeniero Jhonny Curimilma.

Se elaboraron 36 historias de usuario para cumplir con todas las funcionalidades del prototipo. Una vez que se han realizado las historias de usuario, se puede considerar a cada una como una tarea por hacer dentro de los módulos.

Todas las historias de usuario realizadas se encuentran en el Anexo B. En la Figura 2.9 se presentan un resumen de todas las historias de usuario agrupadas por cada módulo.

Módulo Registro/Acceso Usuarios

1	HU01_01_OPERADORES
2	HU01_02_ROLES
3	HU01_03_AUTENTICACION
4	HU01_04_RECUPERAR_CONTRASEÑA
5	HU01_05_GUI_LOGIN
6	HU01_06_GUI_RECUPERAR_CONTRASEÑA
7	HU01_07_GUI_OPERADORES

Módulo Generar Registros

23	HU04_01_CREAR_REGISTRO
24	HU04_02_PARAMETROS_REGISTRO
25	HU04_03_EXITO_REGISTRO
26	HU04_04_FALLO_REGISTRO
27	HU04_05_GUARDAR_REGISTRO
28	HU04_06_GUI_REGISTROS

Módulo Comunicación/Conexión

8	HU02_01_PROCESO_CONEXION
9	HU02_02_EXITO_CONEXION
10	HU02_03_FALLO_CONEXION
11	HU02_04_DIAGRAMA_RED_MODBUS
12	HU02_05_GUI_CONEXION
13	HU02_06_GUI_DIAGRAMA_RED

Módulo Base de Datos

29	HU05_01_OPERADORES
30	HU05_02_PERFILES
31	HU05_03_ESCLAVOS
32	HU05_04_REGISTROS

Módulo Lectura/Escritura

14	HU03_01_CREAR_LECTURA
15	HU03_02_EXITO_LECTURA
16	HU03_03_FALLO_LECTURA
17	HU03_04_RESULTADO_LECTURA
18	HU03_05_CREAR_ESCRITURA
19	HU03_06_EXITO_ESCRITURA
20	HU03_07_FALLO_ESCRITURA
21	HU03_08_RESULTADO_ESCRITURA
22	HU03_09_GUI_LECTURA/ESCRITURA

Módulo HMI

33	HU06_01_CONTROL_TEMPERATURA
34	HU06_02_CONTROL_NIVEL_TANQUE
35	HU06_03_CUADRO_RESUMEN_VARIABLES
36	HU06_04_GUI_HMI_ESPECIFICA

Figura 2.9. Historias de Usuario

Todas las historias de usuario tienen un identificador único. Este identificador tiene la siguiente estructura.

HU##_X_Y_Z

HU##: Historia de Usuario número ##.

X: Corresponde al número de modulo al que está asociada la historia de usuario.

Y: Corresponde al número de la historia dentro de un determinado módulo.

Z: Corresponde al nombre de la historia de usuario.

A continuación se muestran los resultados de las historias de usuario tomando únicamente el nombre y el enunciado de las mismas y dividiéndolas por módulo.

En la Tabla 2.2 se muestran las historias de usuario del módulo Registro/Acceso de usuarios. En este módulo el administrador podrá realizar la creación y modificación de operadores del sistema además de llevar un control de accesos

Tabla 2.2. Módulo Registro/Acceso Usuarios

ID de la Historia de Usuario	Enunciado de la Historia
HU01_01_OPERADORES	Como administrador se quiere crear, eliminar y actualizar operadores para permitir solo a los operadores registrados manejar el software. Y cada que se realice una operación, esta estará asociada al operador registrado.
HU01_02_ROLES	Como administrador se quiere crear operadores con dos perfiles diferentes. Un perfil de administrador y un perfil de operador para signar todos los privilegios al perfil de administrador y restringir algunos para los operadores.
HU01_03_AUTENTICACION	Como operador se quiere ingresar un usuario y contraseña al momento de utilizar el software, para saber que todas las acciones que se ejecutaron en el transcurso de tiempo que dure mi turno son mi responsabilidad.
HU01_04_RECUPERAR_CONTRASEÑA	Como operador se quiere tener una opción de credenciales para recuperar mi usuario y contraseña a través de mi correo electrónico.
HU01_05_GUI_LOGIN	Como operador se quiere tener una interfaz gráfica de login donde ingrese mi usuario y mi contraseña para ingresar en el sistema y queden registradas las acciones realizadas en mi sesión.
HU01_06_GUI_RECUPERAR_CONTRASEÑA	Como operador se quiere tener una interfaz GUI LOGIN donde se muestre una pestaña que me envíe a otra interfaz GUI RECUPERAR CONTRASEÑA y me pida un correo electrónico para recuperar mi usuario o contraseña y poder ingresar al sistema.
HU01_07_GUI_OPERADORES	Como administrador se quiere contar con una interfaz GUI OPERADORES, donde ingrese el nombre, teléfono, correo electrónico y sobre todo un rol Operador/Administrador para crear, actualizar y eliminar operadores.

En la Tabla 2.3 se tiene las historias de usuario para el módulo Comunicación/Conexión en el cual se desarrollará todo el proceso de establecimiento de la comunicación y conexión con los equipos dentro de la red. Además de mostrar el estado de tal proceso, es decir si se realizó o no con éxito.

Tabla 2.3. Módulo Comunicación/Conexión

ID de la Historia de Usuario	Enunciado de la Historia
HU02_01_PROCESO_CONEXION	Como operador se quiere ingresar la dirección IP de un dispositivo que se encuentra en la red para establecer la conexión vía ModBus/TCP y posteriormente gestionar estos equipos.
HU02_02_EXITO_CONEXION	Como operador se quiere visualizar una pantalla emergente con un aviso de que la conexión se realizó con éxito para asegurar ejecutar las siguientes funcionalidades.
HU02_03_FALLO_CONEXION	Como operador se quiere observar una pantalla emergente con un aviso de que la conexión no se realizó, para avisar al operador que la conexión falló.
HU02_04_DIAGRAMA_RED_MODBUS	Como operador se quiere visualizar un diagrama de red donde se muestre gráficamente la conexión entre los dispositivos y el prototipo de software para mostrar el estado de la conexión: si está activo o inactivo.
HU02_05_GUI_CONEXION	Como operador se quiere visualizar una interfaz GUI donde ingrese el nombre del dispositivo y su dirección IP para establecer la conexión mediante el protocolo ModBus/TCP entre el dispositivo y el prototipo de software.
HU02_06_GUI_DIAGRAMA_RED	Como operador se quiere visualizar en una interfaz GUI ESQUEMA DE RED el esquema de red que tengo entre los dispositivos y el prototipo de software para verificar como se encuentra el estado de la conexión en todo momento.

En la Tabla 2.4 se muestran las historias de usuario del módulo Lectura/Escritura. Aquí se desarrolló tanto el proceso para la funcionalidad de leer, como de escribir en los registros que poseen los dispositivos que forman parte de la red. Además de mostrar el estado de cada uno de los procesos, es decir si se realizaron o no con éxito.

Tabla 2.4. Módulo Lectura/Escritura

ID de la Historia de Usuario	Enunciado de la Historia
HU03_01_CREAR_LECTURA	Como operador se quiere ingresar la o las direcciones de los registros que se quiere leer, esto para visualizar el valor que está guardando en esa o esas direcciones de registros.
HU03_02_EXITO_LECTURA	Como operador se quiere observar una pantalla emergente con un aviso indicándome que la lectura se realizó con éxito para asegurar que se realizó la lectura correctamente.
HU03_03_FALLO_LECTURA	Como operador se quiere observar una pantalla emergente con un aviso indicándome que la lectura fallo, para identificar que la lectura no se realizó correctamente y cuál es la causa de esto.
HU03_04_RESULTADO_LECTURA	Como operador se quiere observar el resultado de la lectura mostrándome el valor para verificar que la lectura se está realizando de forma correcta.
HU03_05_CREAR_ESCRITURA	Como operador con privilegios de Administrador se quiere elegir la dirección y el valor que voy a escribir en el o los registros para realizar el proceso de escritura.
HU03_06_EXITO_ESCRITURA	Como operador con privilegios de Administrador se quiere observar una pantalla emergente con un aviso indicándome que la escritura se realizó con éxito, para asegurar que se realizó la escritura correctamente.
HU03_07_FALLO_ESCRITURA	Como operador con privilegios de Administrador se quiere observar una pantalla emergente con un aviso indicándome que la escritura fallo. Para identificar que la escritura no se realizó correctamente y cuál es la causa de esto.
HU03_08_RESULTADO_ESCRITURA	Como operador con privilegios de Administrador se quiere observar el resultado de la escritura, mostrándome que el proceso se realizó correctamente.
HU03_09_GUI_LECTURA/ESCRITURA	Como operador se quiere tener una interfaz GUI LECTURA/ESCRITURA donde pueda ingresar los distintos parámetros definidos en las historias de usuario previas para ejecutar las funcionalidades de lectura y escritura para cada esclavo.

En la Tabla 2.5 se observan las historias de usuario del módulo Generar Registros. En este módulo se describe tanto el procedimiento como los parámetros necesarios para generar

históricos de los registros de las variables que son leídas de un dispositivo en particular. Además de mostrar el estado de tal proceso, es decir, si se realizó o no con éxito.

Tabla 2.5. Módulo Generar Registros

ID de la Historia de Usuario	Enunciado de la Historia
HU04_01_CREAR_REGISTRO	Como operador se quiere seleccionar uno varios de los registros de los cuales estoy realizando la lectura, seleccionar un intervalo de muestreo y el tiempo total en el que se realizara la creación de registro para generar automáticamente la creación de históricos con los valores de los registros seleccionados y guardarlos en la base de datos.
HU04_02_PARAMETROS_REGISTRO	Como operador se quiere visualizar mediante una interfaz gráfica como se están guardando los registros dentro del histórico para asegurarme que se está conservando el formato contemplado en la fase del diseño.
HU04_03_EXITO_REGISTRO	Como operador se quiere observar una pantalla emergente con un aviso indicando que la creación del histórico se hizo con éxito para asegurar que se realizó la creación del histórico correctamente.
HU04_04_FALLO_REGISTRO	Como operador se quiere observar una pantalla emergente con un aviso indicando que la creación del histórico ha fallado para identificar que la creación del histórico no se realizó correctamente y cuál es la causa de esto.
HU04_05_GUARDAR_REGISTRO	Como operador se quiere guardar el histórico en la base de datos automáticamente para facilitar el proceso ya que este podría demorar varios minutos y horas.
HU04_06_GUI_REGISTROS	Como operador se quiere contar con una interfaz GUI donde pueda ingresar los parámetros necesarios para generar el histórico de los registros seleccionados.

En la Tabla 2.6 se tiene las historias de usuario del módulo Base de Datos. El cual describe las dos principales tablas que va a tener la base de datos. Estas tablas son OPERADORES y REGISTROS. En estas tablas se llevará el registro de los operadores del sistema, así como su respectivo perfil. Y también se almacenará los históricos que se generen dentro del sistema.

Tabla 2.6. Módulo Base de Datos

ID de la Historia de Usuario	Enunciado de la Historia
HU05_01_OPERADORES	Como administrador se quiere crear una tabla OPERADORES en la base de datos para llevar el registro de los operadores y el perfil de cada uno de ellos.
HU05_02_PERFILES	Como administrador se quiere crear una tabla PERFILES en la base de datos para llevar el registro de los perfiles.
HU05_03_ESCLAVOS	Como administrador se quiere crear una tabla ESCLAVOS en la base de datos para llevar el registro de los esclavos.
HU05_04_REGISTROS	Como administrador se quiere crear una tabla REGISTROS en la base de datos para guardar los históricos que se vayan creando conforme sea solicitado.

En la Tabla 2.7 se observa las historias de usuario del módulo HMI, en este módulo se define todas las interfaces gráficas necesarias para interactuar con cada una de las funcionalidades que tiene el prototipo de software.

Tabla 2.7. Módulo HMI

ID de la Historia de Usuario	Enunciado de la Historia
HU06_01_CONTROL_TEMPERATURA	Como operador se quiere visualizar de forma gráfica el valor de la variable que representa a la temperatura de ese instante y de instantes anteriores y también contar con un umbral de temperatura configurable para visualizar el cómo está variando la temperatura.
HU06_02_CONTROL_NIVEL_TANQUE	Como operador se quiere visualizar de forma gráfica el valor de la variable que representa el nivel del tanque, visualizar si el tanque se está llenando o vaciando para visualizar el cómo está variando el nivel del tanque y también activar automáticamente el proceso de llenar o vaciar el tanque dependiendo del nivel.
HU06_03_RESUMEN_VARIABLES	Como operador se quiere visualizar un cuadro en el que se muestren el valor de todas las variables que se estén utilizando para el desarrollo del HMI para visualizar de forma conjunta todas las variables.
HU06_04_GUI_HMI_ESPECIFICA	Como operador se quiere contar con una interfaz GUI donde cense el nivel de un tanque, la velocidad de llenado del tanque y el nivel de temperatura para activar un sensor en caso de que uno de los parámetros configurados ha pasado un umbral.

2.3 Diseño de Módulos

En este sub módulo se describen y muestran los distintos diagramas planteados para cada uno de los módulos. Los diagramas ayudarán a visualizar la interacción entre las clases que conforman el sistema, así como la interacción entre los operadores (Usuarios), el prototipo de software, la base de datos y los dispositivos en red (Esclavos).

2.3.1 StarUML

StarUML es una herramienta de modelado de software de código abierto que admite UML (*Unified Modeling Language*). Se basa en la versión 1.4 de UML, proporciona once tipos diferentes de diagramas y acepta la notación UML 2.0. Apoya activamente el enfoque MDA (*Model Driven Architecture*) al admitir el concepto de perfil UML y permite generar código para múltiples idiomas[23].

StarUML es un sofisticado modelador de software destinado a admitir modelos ágiles y concisos[24]. En la Figura 2.10 se muestra el entorno de desarrollo de StarUML.

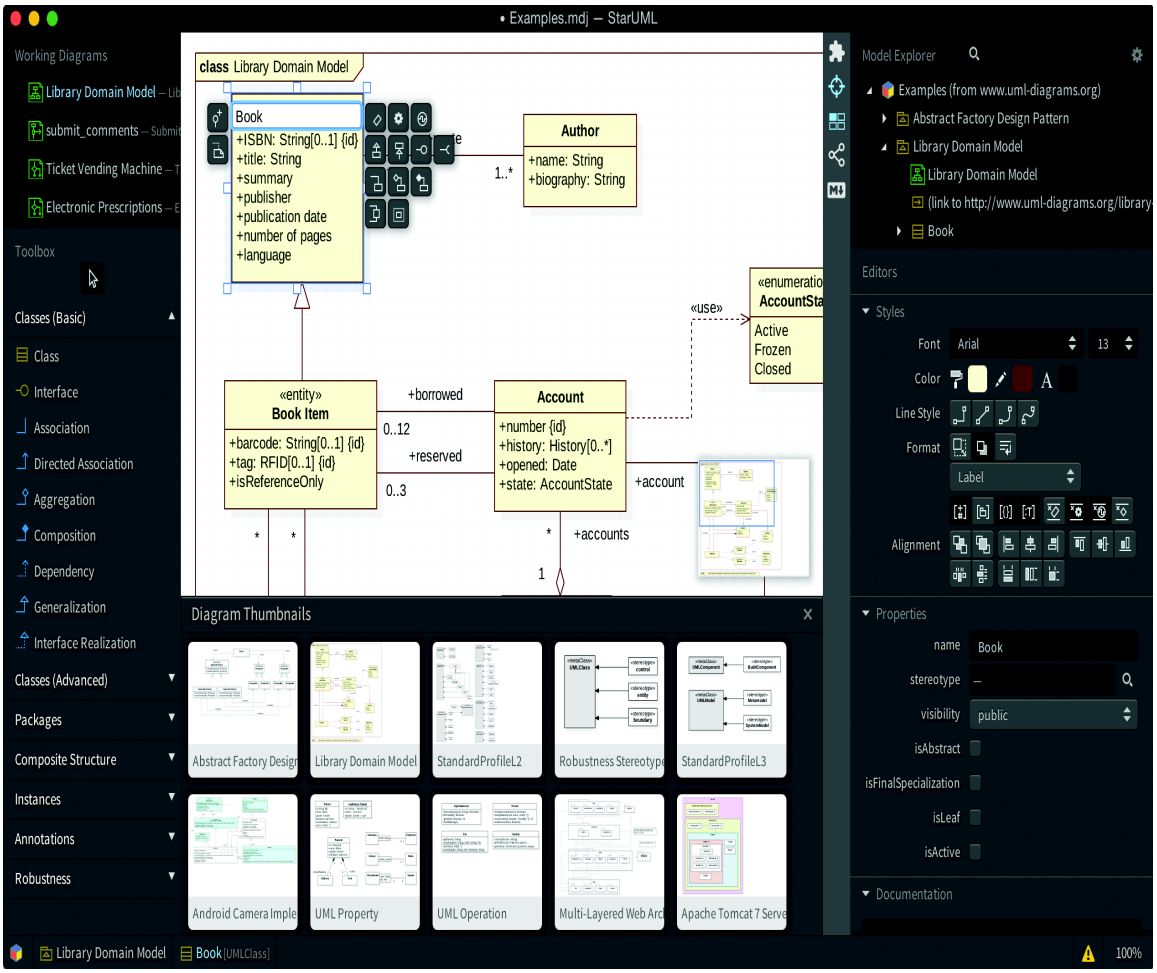


Figura 2.10. Entorno de Desarrollo de StarUML

2.3.2 Diseño Módulo Registro/Acceso Usuarios

a. Diagramas Módulo Registro/Acceso Usuarios

Los diagramas presentados en este módulo representan los procesos, actividades y la interacción entre las clases, para que el usuario pueda realizar las tareas de registrar, modificar y eliminar usuarios, así como verificar las credenciales de inicio de sesión (login).

b. Diagrama de Casos de Uso Módulo Registro/Acceso Usuarios

En la Figura 2.11 se muestra el proceso que ejecuta un usuario, teniendo un perfil de "Operador" o de "Administrador", con el prototipo de software para realizar un inicio de sesión en el sistema o a su vez crear, actualizar y eliminar operadores. En este módulo el usuario interactúa indirectamente con la base de datos para verificar las credenciales, así como para registrar cambios en la tabla "Usuario".

El diagrama de casos de uso muestra que el usuario realiza la acción "Acceder Login", luego el prototipo de software solicita el ingreso de las credenciales para el inicio de sesión y finalmente las compara con las credenciales almacenadas en la base de datos. Dependiendo del resultado de esta verificación, el sistema mostrará un mensaje a través de una ventana emergente indicando el inicio de sesión exitoso o fallido.

Si el mensaje muestra un inicio de sesión fallida el usuario podrá realizar una recuperación de credenciales a través del ingreso de su correo electrónico, el cual será verificado en la base de datos y posteriormente se realizará el envío de un *e-mail* con sus respectivas credenciales.

En este diagrama el usuario puede elegir crear un nuevo usuario, para lo cual deberá ingresar los atributos del nuevo usuario. Si quiere actualizar o eliminar determinado usuario deberá seleccionarlo mediante su nombre de usuario. Luego de cualquiera de estas tres acciones el sistema deberá realizar una actualización de la base de datos.

c. Diagrama de Actividades Módulo Registro/Acceso Usuarios

En la Figura 2.12 se observa que el diagrama de actividades se ha dividido en dos procesos, esto debido a que son dos flujos diferentes tanto la parte de iniciar sesión como en la parte de registrar, actualizar y eliminar operadores.

El diagrama de actividades de inicio de sesión muestra como el usuario deberá primero ingresar dos credenciales las cuales son el nombre de usuario y su *password*. Una vez realizado este proceso el prototipo de software realizará una consulta a la base de datos verificando la existencia del nombre de usuario en la tabla "USUARIO". De existir ese

nombre de usuario entonces se compara el *password* proporcionado con el almacenado en la tabla. Si coinciden se presentará una ventana emergente mostrando un mensaje de inicio de sesión exitoso; de lo contrario, al no existir el nombre de usuario o no coincidir el password proporcionado se presentará una ventana donde se solicita el ingreso de un correo electrónico para realizar la recuperación de las credenciales nombre de usuario y password.

Para el proceso de recuperar credenciales, una vez ingresado el correo electrónico se procederá a realizar una consulta en la base de datos para verificar su autenticidad y luego realizar él envio de un e-mail a ese usuario recordándole sus credenciales. Si al realizar la consulta del correo este no existe se muestra una ventana emergente con un mensaje de credenciales erróneas y se termina el proceso.

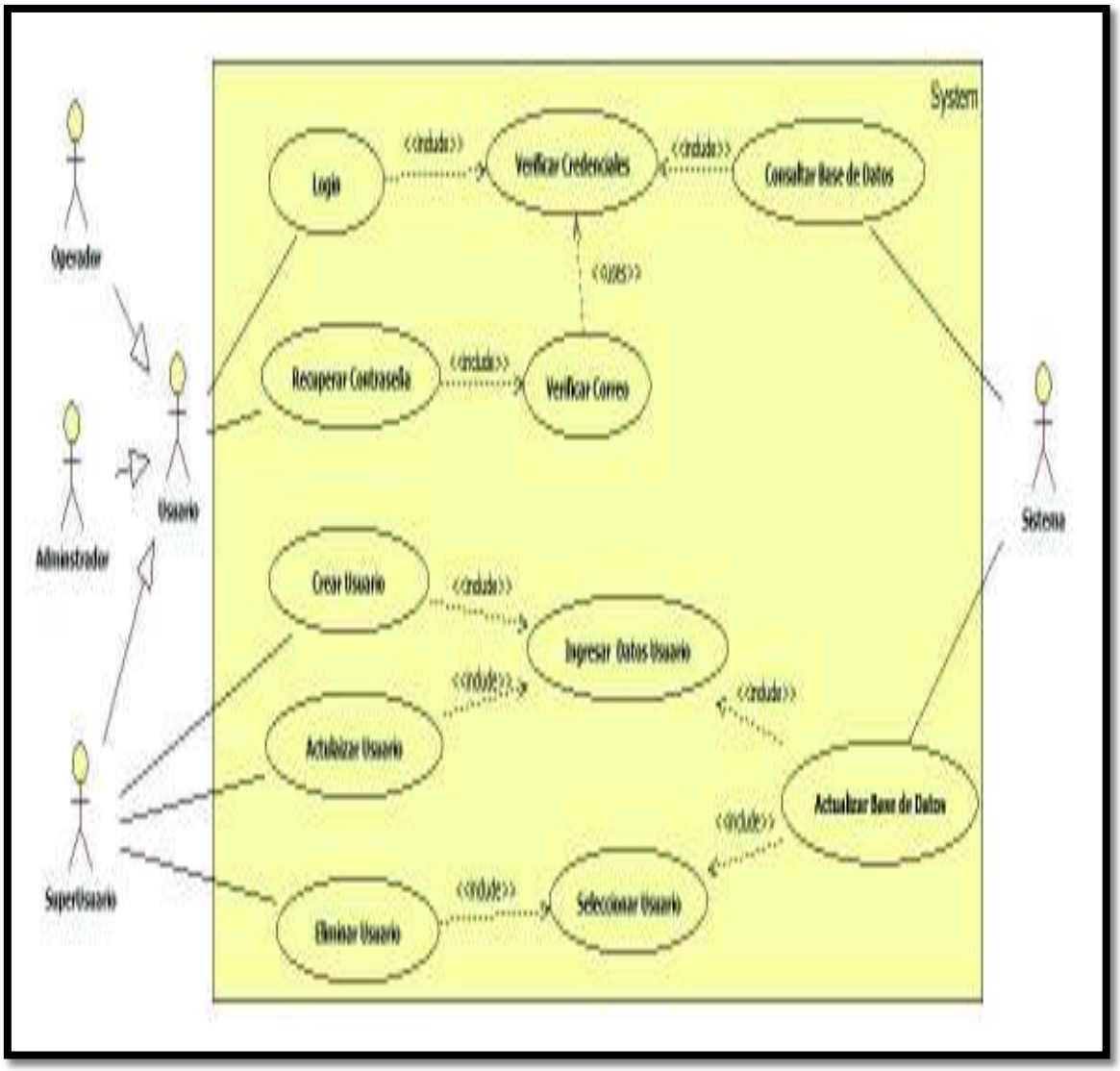


Figura 2.11. Diagrama de Casos de Uso Módulo Registro/Acceso Usuarios

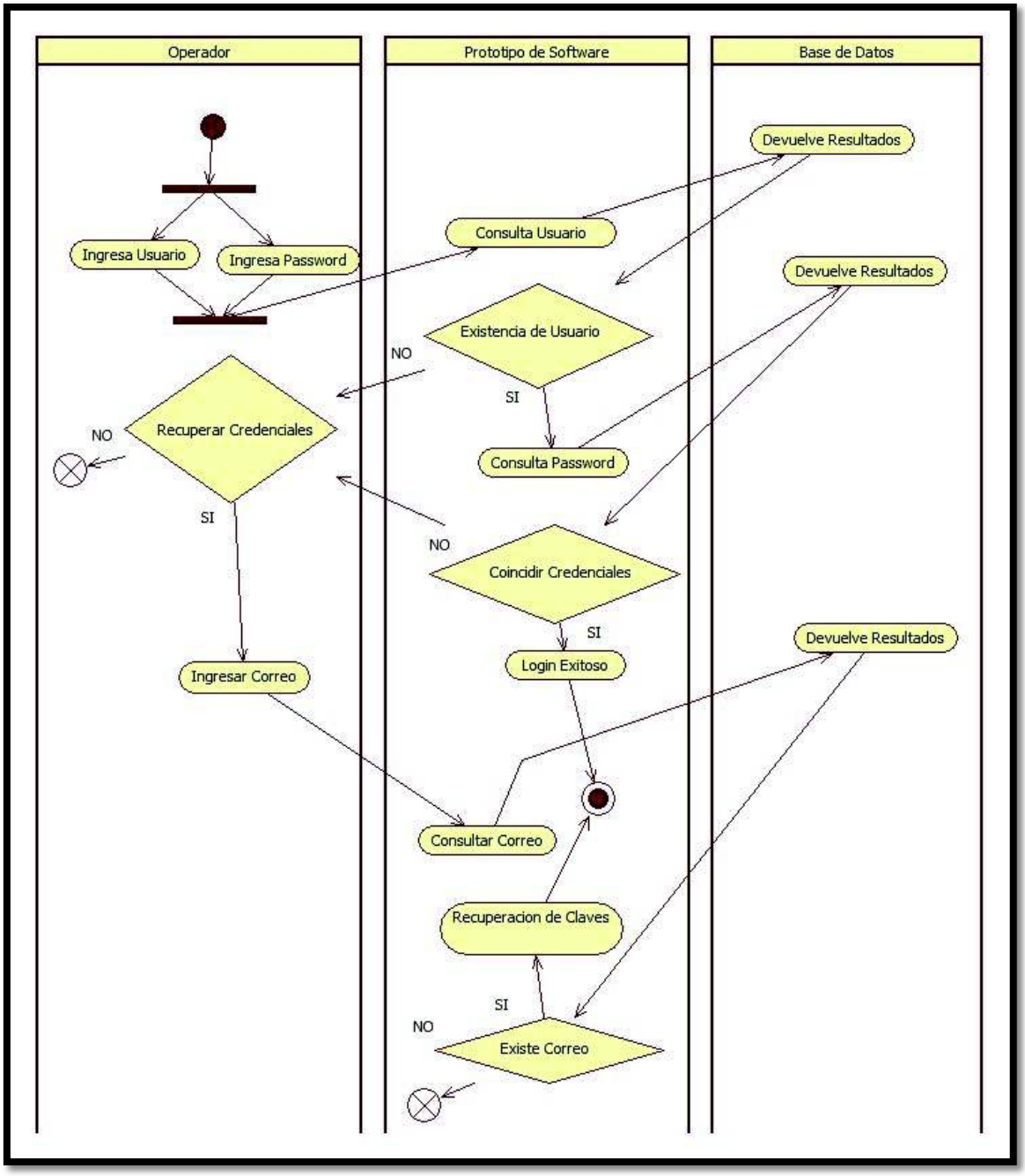


Figura 2.12. Diagrama de Actividades Módulo Registro/Acceso Usuarios (Login)

En la segunda parte, en la Figura 2.13, el diagrama de actividades muestra que el usuario empieza realizando una de estas tres actividades: Crear Usuario, Modificar Usuario o Eliminar Usuario.

Si el usuario elije la opcion de crear un nuevo usuario el sistema le pedirá ingresar los atributos: id de usuario, id perfil, nombre de usuario, *password* y correo. Luego de esto el sistema realizará una consulta a la base de datos para verificar si el id de usuario o el nombre de usuario ya existen. De existir, se pide ingresar un id de usuario o un nombre de

usuario diferente, de lo contrario el sistema crea el nuevo usuario en la base de datos y la actualiza.

Para la parte de modificar un usuario se pedirá primero identificarlo y con esto el sistema obtendra los datos de ese operador y los mostrará al operador para que se actualice todos o algunos de sus atributos. Una vez que el usuario solicite la actualización de los datos el sistema realizará los cambios en la base de datos.

Antes de actualizar los datos en la base, se verificará que el id de usuario y el nombre de usuario no exista en la tabla o de lo contario de repetira la rutina anterior de nombre de usuario existente y se solicitara un cambio.

Para la parte de eliminacion de un operador el sistema pedirá identificarlo y será borrado de la base de datos.

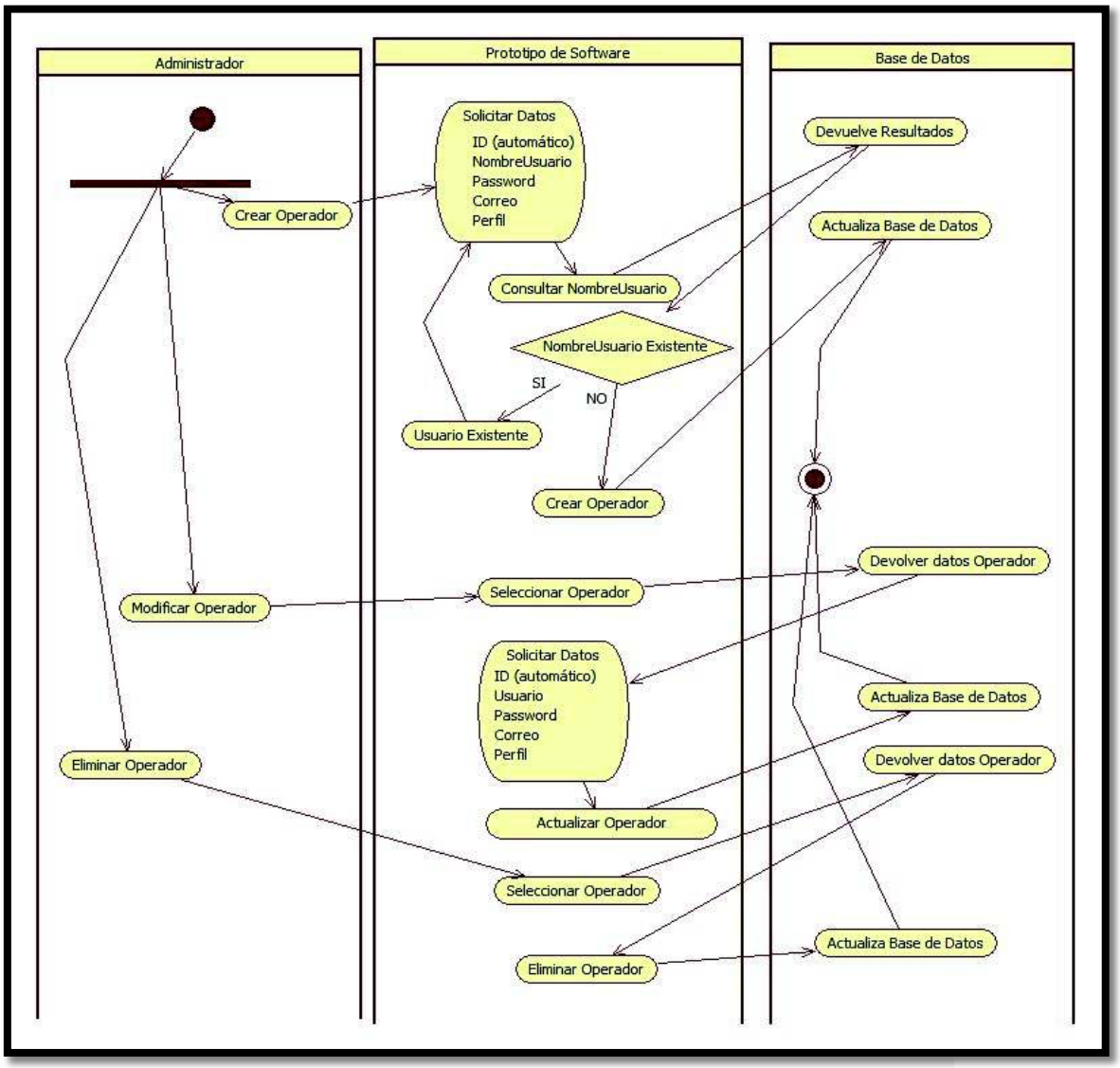


Figura 2.13. Diagrama de Actividades Módulo Registro/Acceso

d. Interfaz Gráfica Módulo Registro/Acceso Usuarios

En el módulo de Registro/Acceso de usuarios se debe considerar las tres funcionalidades principales con las que se va a contar, éstas son.

- Acceder al menú de funcionalidades registrando su usuario y contraseña (login)
- Recuperar las credenciales de inicio de sesión mediante correo electrónico
- CRUD de usuarios

Para la primera funcionalidad “login” se tendrá una interfaz como se muestra en la Figura 2.14.

Figura 2.14. Diseño Interfaz Gráfica “LOGIN”

Esta interfaz contará con un cuadro de texto (*JTextField*), en el cual el usuario deberá ingresar su nombre de usuario, también tendrá un cuadro de texto para ingresar el password (*JPasswordField*), este tipo de componente tiene la particularidad de ocultar el texto ingresado al reemplazarlo por un carácter especial.

Esta interfaz tendrá tres botones (*JButton*). El primero servirá para ingresar las credenciales y compararlas con las almacenadas en la base de datos, si las credenciales son correctas automáticamente reconocerá al usuario registrado y presentará el menú de funcionalidades de acuerdo a su nivel de acceso (Perfil); el segundo botón cancelará la operación es decir que no validará las credenciales y borrará el texto escrito dentro de los dos cuadros de texto anteriormente descritos; el tercer botón tendrá la funcionalidad de re direccionar al usuario a una segunda ventana donde podrá realizar la acción de recuperar las credenciales a través de correo electrónico. Este botón al igual que el *label* (*JLabel*), que indica que las credenciales son incorrectas, se activará solo si las credenciales fueron validadas con la base de datos, ya que por defecto estarán ocultas.

En la Figura 2.15 se muestra el diseño de la interfaz de la ventana que se mostrará al usuario cuando se genere la funcionalidad de recuperar las credenciales, después de haber ingresado el nombre de usuario o contraseña.

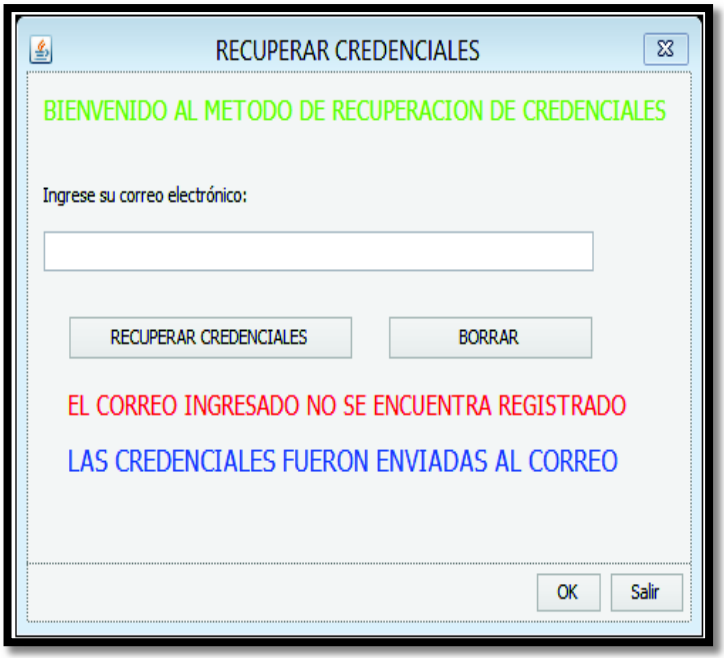


Figura 2.15. Diseño de Interfaz Gráfica “RECUPERAR CREDENCIALES”

Esta interfaz cuenta con una caja de texto (*JTextField*) en la cual se ingresará el correo del usuario del cual se desea recuperar las credenciales.

Además tendrá dos botones (*JButton*), el primer botón validará el correo electrónico ingresado en la caja de texto con los correos guardados en la base de datos, y de ser validados exitosamente se enviará a ese correo sus credenciales es decir nombre de usuario y contraseña. El segundo botón tendrá la funcionalidad de cancelar la operación de recuperar credenciales y limpiar la caja de texto con el correo que se haya ingresado.

Además contará con dos *labels* (*JLabel*) para indicar al usuario si la recuperación de credenciales se realizó o no con éxito.

En este módulo también se tendrá una interfaz gráfica en la cual se realice el proceso de creación, modificación y eliminación de usuarios (Operadores). En la Figura 2.16 se muestra el diseño de la interfaz para realizar esta funcionalidad.

La interfaz gráfica para recuperar credenciales contará principalmente con una tabla (*JTable*) en la cual se refleje los usuarios (Operadores) que se encuentren registrados en la base de datos cada uno con sus respectivos valores de id de usuario, id de perfil, nombre

de usuario, password y correo. Además, tendrá 5 cajas de texto (*JTextField*) en la cuales al seleccionar un determinado usuario y sus atributos se verán reflejados en cada caja.

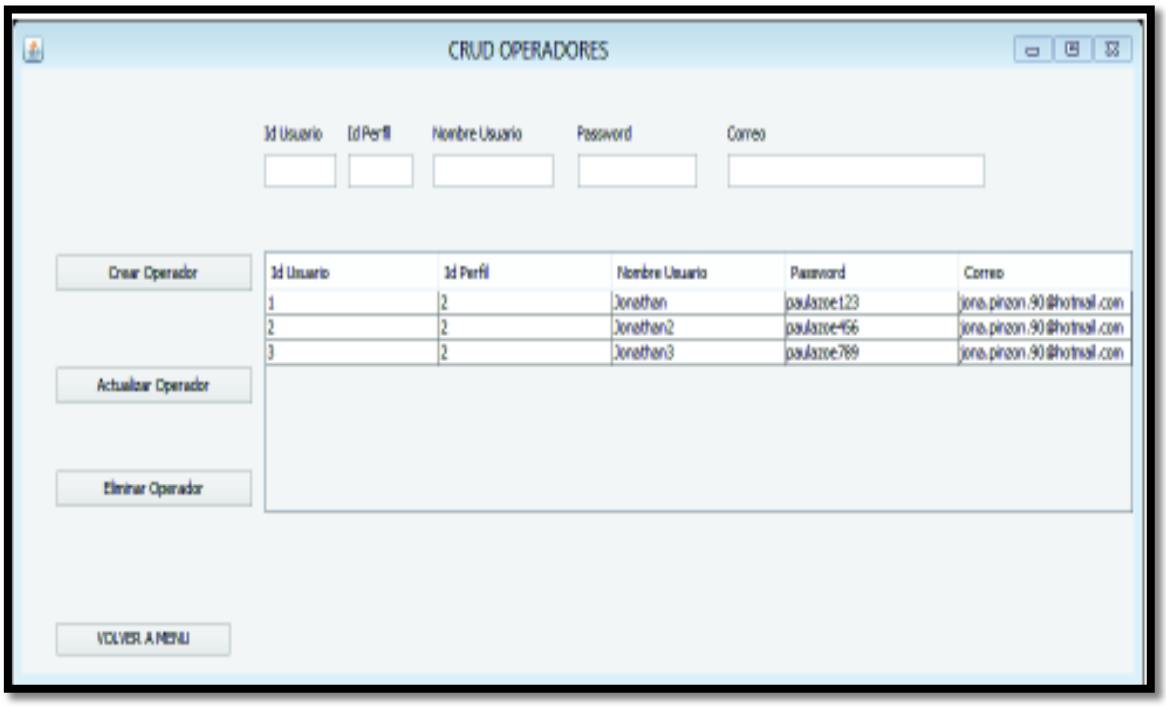


Figura 2.16. Diseño Interfaz Gráfica “CRUD OPERADORES”

La interfaz también contará con 4 botones (*JButton*) de los cuales el primer botón realizará la funcionalidad de crear un nuevo operador, usando como atributos del operador los valores que se encuentran en las cajas de texto. El segundo botón actualizará a un operador existente, el cual será seleccionado previamente y se lo actualizará con los valores cargados en las cajas de texto. El tercer botón realizará la funcionalidad de eliminar un operador el cual es seleccionado previamente desde la tabla de operadores.

El cuarto botón realizará la funcionalidad de regresar al menú principal, tomando en cuenta que en este punto se ha realizado el proceso de registro correctamente.

2.3.3 Diseño Módulo Comunicación/Conexión

a. Diagramas Módulo Comunicación/Conexión

Los diagramas presentados en este módulo representan los procesos, actividades y la interacción entre las clases, para que el usuario pueda realizar las tareas de conectarse y establecer una comunicación con los dispositivos (Esclavos) que están conectados a la red y que utilizan el protocolo ModBus/TCP como protocolo de comunicación.

b. Diagrama de Casos de Uso Módulo Comunicación/Conexión

La Figura 2.17 muestra la interacción que realiza el operador para establecer la conexión y realizar la comunicación entre el prototipo de software y los esclavos.

El operador tendrá que ingresar las direcciones IP de los esclavos con los que quiera establecer conexión y mantener una comunicación. El prototipo de software ejecutará los métodos para solicitar la conexión con los esclavos.

El esclavo devolverá un mensaje respondiendo a la solicitud del prototipo de software (Maestro) de establecer la conexión. Y con esto se mostrará al operador si la conexión con los esclavos se realizó exitosamente o no.

Además, el operador puede ejecutar la funcionalidad de visualizar el diagrama de red. Aquí se podrá observar el estado de las conexiones con cada uno de los esclavos. En este caso, el estado de la conexión con los dos esclavos virtuales y el esclavo físico.

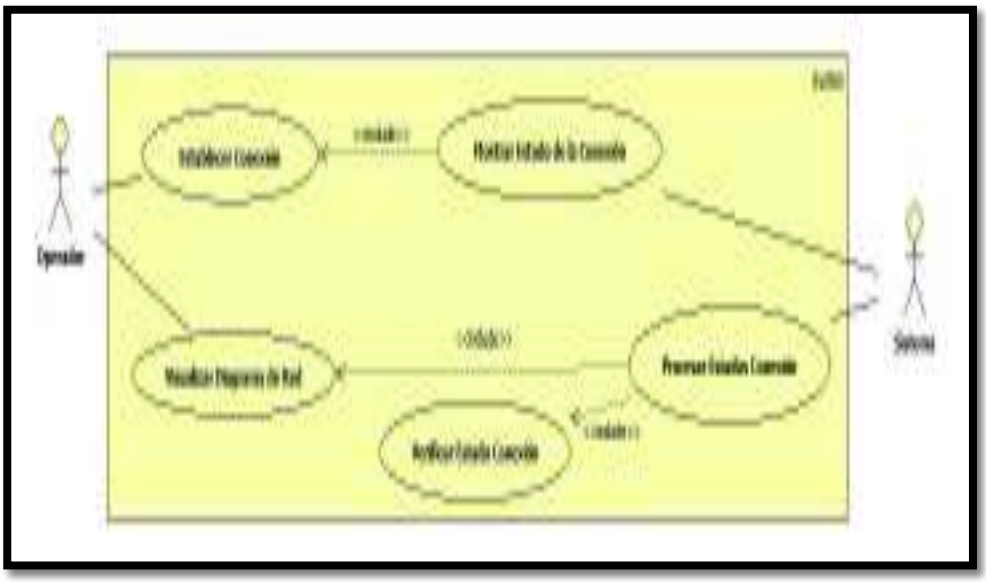


Figura 2.17. Diagrama de Casos de Uso Módulo Comunicación/Conexión

c. Diagrama de Actividades Módulo Comunicación/Conexión

En la Figura 2.18 se muestra el orden del flujo que se realiza cuando un operador requiere establecer la conexión con los esclavos, y posteriormente visualizar un diagrama donde se observa el estado de conexión con los dispositivos de la red.

Para establecer la conexión, primero el operador ingresa la dirección IP del dispositivo con el que quiere conectarse, luego el prototipo ejecutará la función “abrir_conexion” y enviará la solicitud al dispositivo, este a su vez responderá con un código. Este código será interpretado por el prototipo de software y mostrará si la conexión es exitosa o fallida.

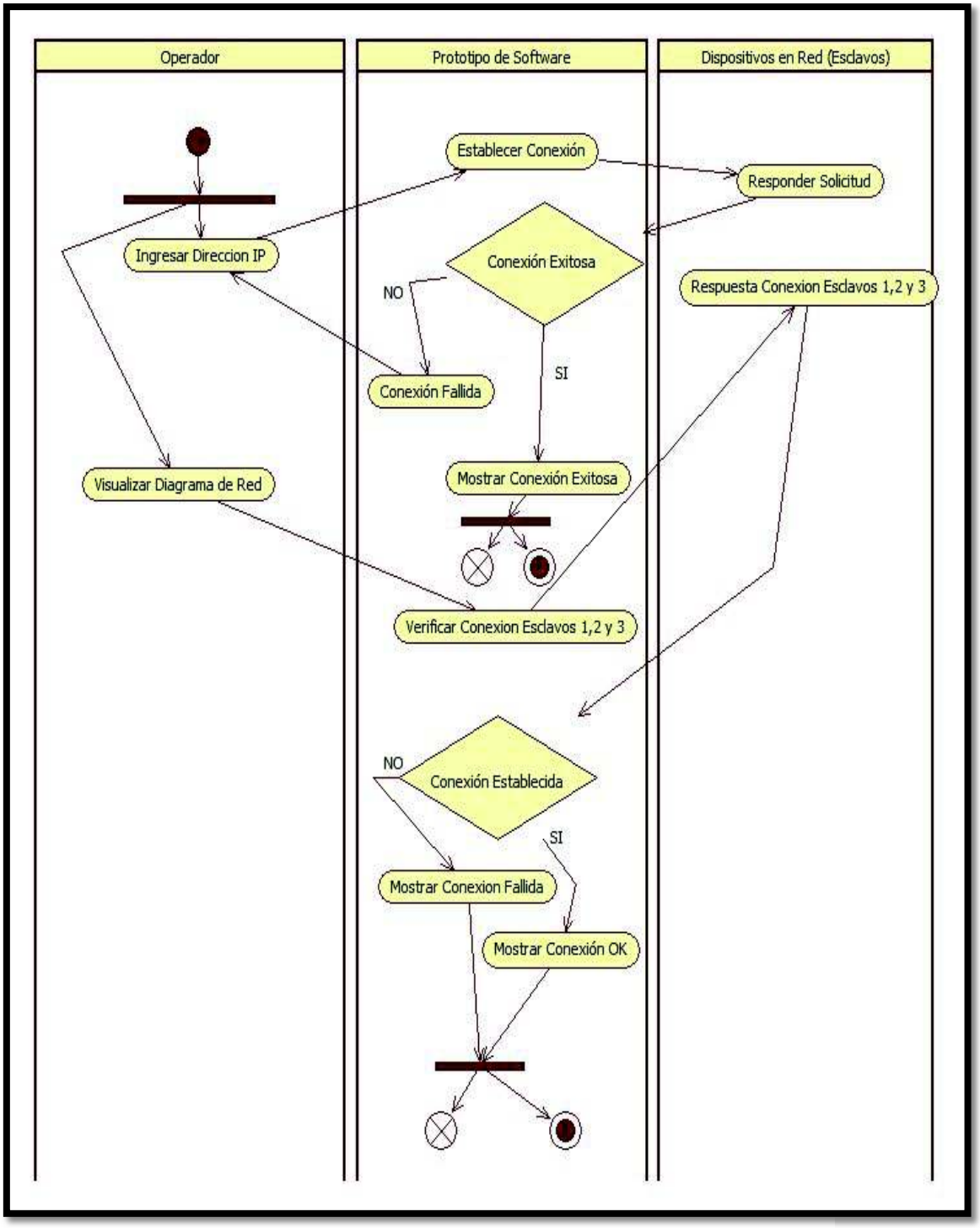


Figura 2.18. Diagrama de Actividades Módulo Comunicación/Conexión

d. Interfaz Gráfica Módulo Comunicación/Conexión

Para este módulo se diseñará dos interfaces gráficas ya que en el alcance del prototipo se cuenta con dos funcionalidades. La primera es el establecimiento de la conexión y mantener la comunicación con los tres diferentes esclavos y, además, visualizar en tiempo real el estado de la conexión con cada uno de los esclavos.

En la Figura 2.19 se muestra la interfaz gráfica para establecer la conexión con cada uno de los esclavos.

Figura 2.19. Diseño Interfaz Gráfica “ESTABLECER CONEXION”

Esta interfaz contará con tres cajas de texto (*JTextField*) en los cuales se ingresará la dirección IP de cada uno de los esclavos con los que se va a establecer la conexión.

También tendrá 3 pares de botones (*JButton*) un par para cada esclavo. Uno de los botones realizará el proceso de crear un socket con el puerto 502 (puerto ModBus TCP) y la dirección IP que se encuentra en la caja de texto y posteriormente establecerá la conexión.

El segundo botón cerrará la conexión que se estableció al ejecutar la funcionalidad de establecer conexión.

La interfaz contará con 3 pares de *labels* (*JLabel*) los cuales indicaran el estado de la conexión ya que estará asociado a la funcionalidad de diagrama de red, donde se estará constantemente revisando el estado de la conexión con cada uno de los esclavos.

El *label* de color verde mostrará el estado de “conectado” mientras que el *label* de color rojo mostrará el estado de “desconectado”.

Al igual que en las interfaces anteriores tendrá un botón (*JButton*) para regresar al menú principal.

Como se mencionó anteriormente este módulo también contará con una interfaz gráfica para realizar la funcionalidad de visualizar el estado de red, es decir cómo se encuentra la conexión entre el Maestro (Prototipo de Software) y cada uno de los esclavos.

En la Figura 2.20, que se muestra a continuación, se detalla el diseño de la interfaz gráfica para constantemente visualizar el estado de la conexión con cada uno de los esclavos.

Figura 2.20. Diseño Interfaz Gráfica “DIAGRAMA DE RED”

La interfaz gráfica tendrá 4 *labels* (*JLabel*) asociados a un icono específico, esto gracias a su propiedad “*icon*”, propiedad que tiene un *JLabel*, con ello se podrá representar con una imagen a cada uno de los tres esclavos y al maestro.

Adicional a esto la interfaz gráfica contará con tres pares de *labels* de los cuales: el primero estará representado por la flecha verde, se mostrará siempre y cuando la conexión esta establecida, el segundo representado por una X se mostrará cuando la conexión se haya perdido.

Esta interfaz además cuenta con un botón (*JButton*) con el cual podrá regresar al menú principal.

2.3.4 Diseño Módulo Lectura/Escritura

a. Diagramas Módulo Lectura/Escritura

Los diagramas presentados en este módulo representan los procesos, actividades y la interacción entre las clases, para que el usuario pueda realizar las tareas de lectura y escritura sobre los registros de los dispositivos en red (Esclavos).

b. Diagrama de Casos de Uso Módulo Lectura/Escritura

El diagrama de casos de uso se ha dividido en dos partes tal como se muestra en la Figura 2.21 y 2.22.

La primera parte muestra la secuencia que debe realizar un operador para leer los valores que se encuentran almacenados en los registros de los dispositivos esclavos.

En el proceso de lectura, el operador deberá seleccionar la interfaz gráfica del esclavo que quiere obtener los valores de lectura. El prototipo de software realizará, como primer paso, la conectividad con el esclavo. De tener un canal de comunicación establecido, el operador procederá a seleccionar el tipo de registros que quiere leer. Estos tipos de registros son.

- *Coil Status*: Registros binarios (1/0) de salida.
- *Input Status*: Registros binarios (1/0) de entrada.
- *Holding Registers*: Registros de dos palabras (octetos) de salida.
- *Input Registers*: Registros de dos palabras (octetos) de entrada.

Luego de esto se ejecutará el método para obtener los valores de lectura, si el proceso es exitoso el dispositivo esclavo devolverá los valores de los registros correctamente. De lo contrario el prototipo de software únicamente devolverá un mensaje de error indicando que existió un problema al momento de obtener los valores de los registros del dispositivo.

En la segunda parte se muestra el proceso de escritura, el cual es muy similar al proceso de lectura. Primero se elige el dispositivo sobre el cual se va a escribir el valor de los registros, pero antes se debe verificar si está establecida una conexión con dicho dispositivo. En el proceso de escritura se debe ingresar la cantidad y dirección de los registros a escribir y también el valor con el que sobrescribirá esos registros.

En el proceso de escritura se podrá elegir únicamente entre dos de los cuatro tipos de registros que se mencionaron en el proceso de lectura, estos registros son:

- *Coil Status*: Registros binarios (1/0) de salida.
- *Holding Registers*: Registros de dos palabras (octetos) de salida.

Esto debido a que los registros *Input Registers* e *Input Status* como su nombre lo dice son del tipo “input”, por lo cual ModBus no desarrolló un número de función para escribir en este tipo de registros.

Para terminar el proceso de escritura se ejecutará el método para escribir los valores, el prototipo de software devolverá un mensaje en el cual indique si el proceso de escritura se realizó con éxito o no.

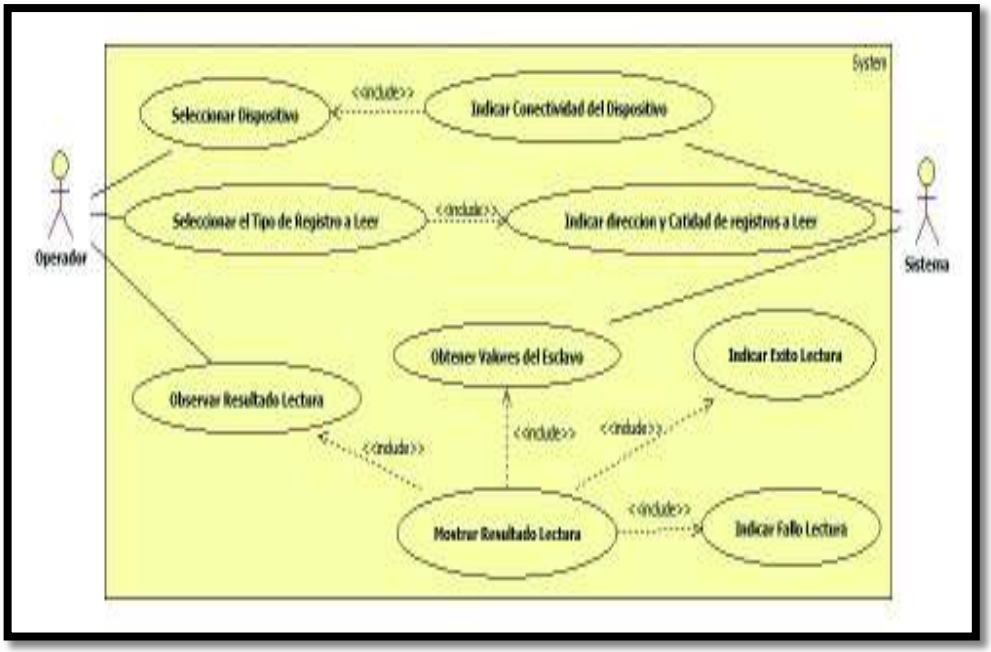


Figura 2.21. Diagrama de Casos de Uso Módulo Lectura/Escritura (Lectura)

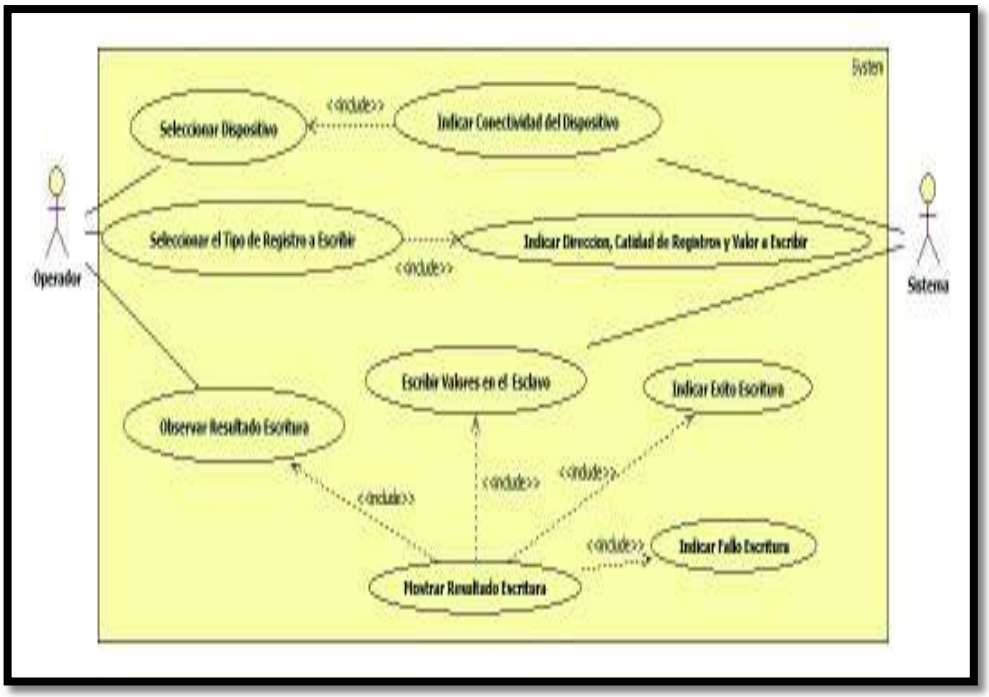


Figura 2.22. Diagrama de Casos de Uso Módulo Lectura/Escritura (Escritura)

c. Diagrama de Actividades Módulo Lectura/Escritura

En las Figuras 2.23 y 2.24 se muestra la forma en que el diagrama de actividades se ha dividido en dos partes al igual que en el diagrama de los casos de uso.

La primera parte muestra la secuencia que ejecuta el prototipo de software para realizar las acciones de lectura. En primer lugar el usuario deberá seleccionar el dispositivo del cual va a visualizar los valores almacenados en sus registros. Luego, el prototipo mostrará la interfaz gráfica del dispositivo en donde puede realizar el proceso de lectura y escritura. Al abrir esta interfaz gráfica el prototipo verifica si está establecida o no una conexión con dicho dispositivo.

Si la conexión está establecida, se elegirá cuál de los 4 tipos de registros con los que cuentan los dispositivos, se va a realizar la lectura. Los registros son los detallados anteriormente en el diagrama de casos de uso: *Coil Status, Input Status, Holding Registers* e *Input Registers*.

Luego de seleccionar el tipo de registro a leer, se ejecutará el método de leer registros.
Finalmente el prototipo mostrará al operador una ventana emergente con un mensaje de lectura exitosa o de lo contrario un mensaje de error, además de mostrar los valores de lectura solicitados.

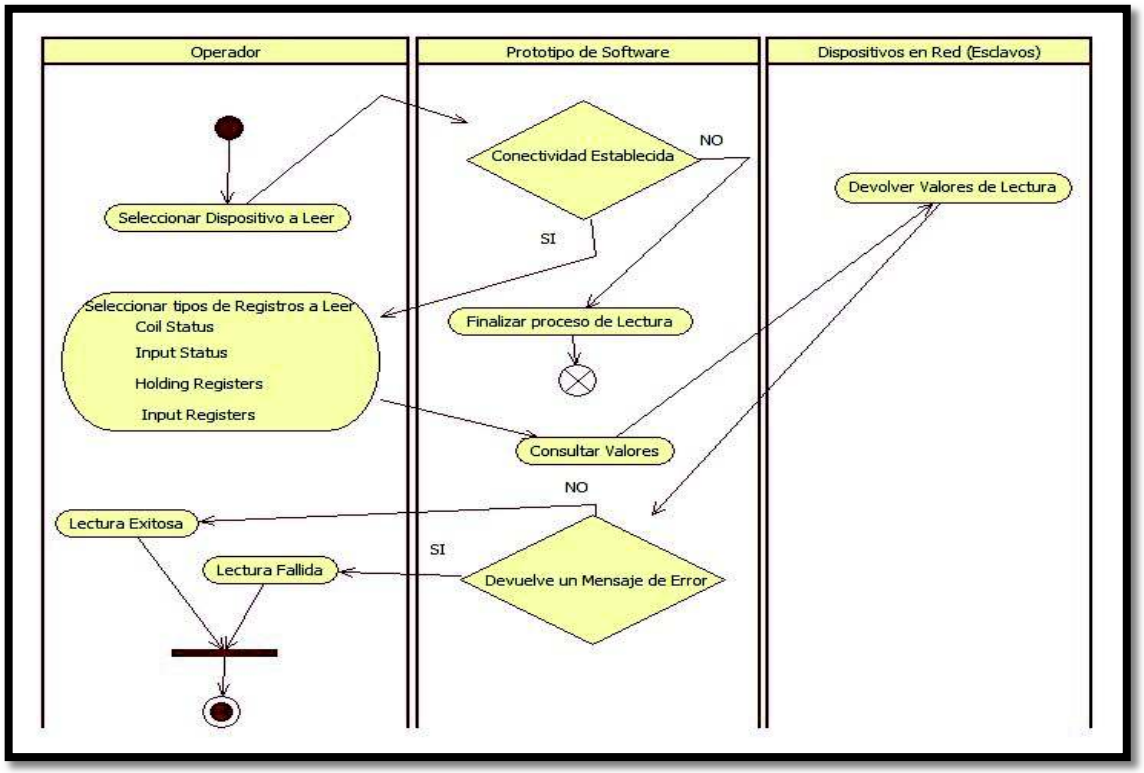


Figura 2.23. Diagrama de Actividades Módulo Lectura/Escritura (Lectura)

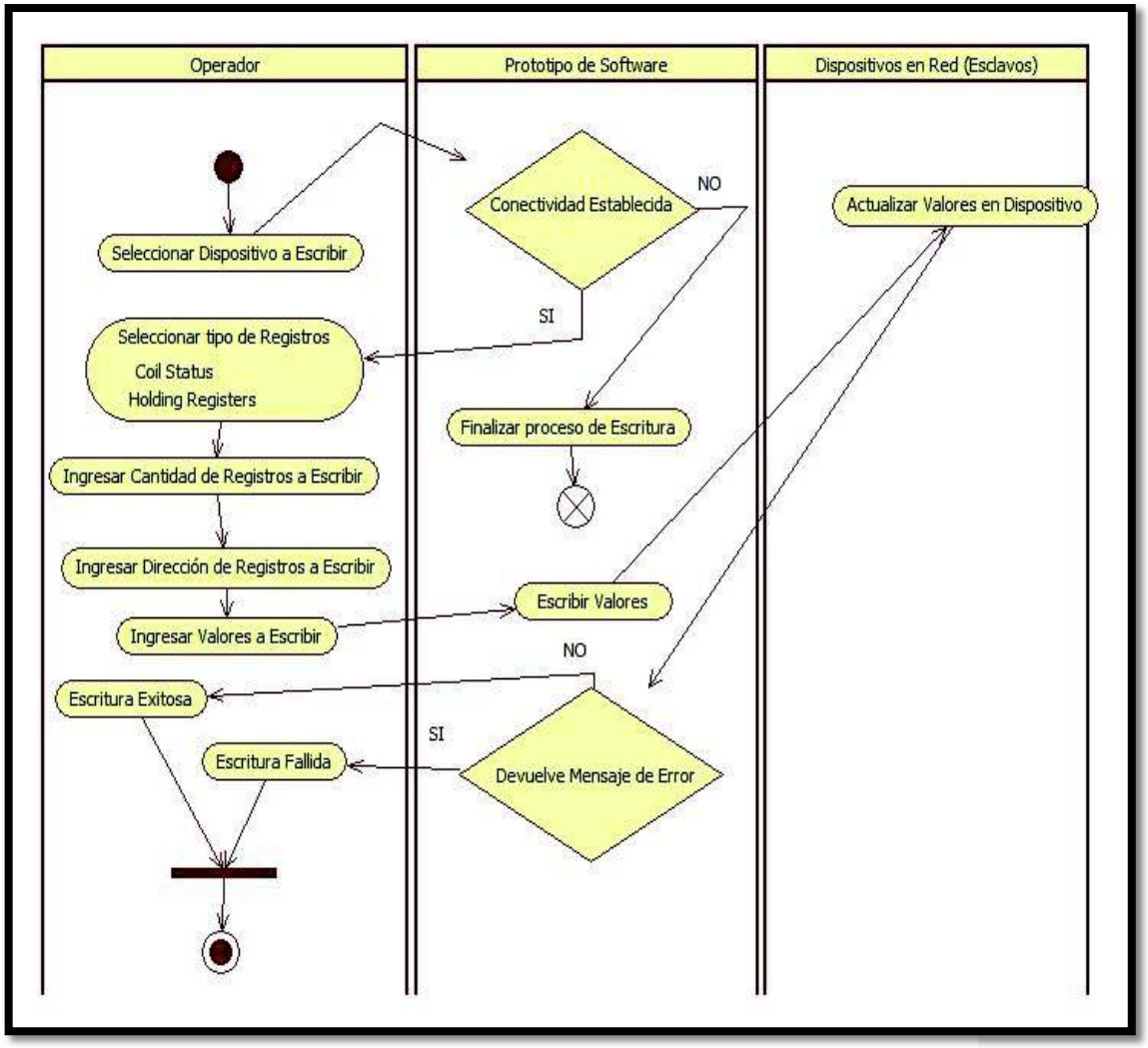


Figura 2.24. Diagrama de Actividades Módulo Lectura/Escritura (Escritura)

d. Interfaz Gráfica Módulo Lectura/Escritura

En este módulo se diseñara una interfaz gráfica para facilitar al usuario el uso de las funcionalidades de lectura y escritura de registros, con las que cuenta el software. Se tendrá tres interfaces gráficas idénticas una para cada esclavo ya que cada uno de los esclavos realizará los procesos de lectura y escritura en canales de conexión diferentes.

En un alcance inicial se pretendía únicamente leer y escribir el tipo de registro "*Holding Register*"; sin embargo, al momento de realizar las historias de usuario se determinó la importancia de los cuatro principales de registros, los cuales son.

- Coil Register
- Input Status
- Input Register
- Holding Register

Por tal motivo, la interfaz gráfica deberá permitir tanto la lectura de los cuatro tipos de registros anteriormente mencionados y la escritura en los dos tipos de registros que el protocolo me permite, es decir “COIL REGISTER” e “INPUT REGISTER”.

En la Figura 2.25 se presenta el diseño de la interfaz gráfica para realizar la funcionalidad de lectura y escritura de los registros que posee cada uno de los esclavos.

Figura 2.25. Diseño Interfaz Gráfica “LECTURA/ESCRITURA” Esclavo 2

La interfaz gráfica tendrá 6 botones (*JButton*) uno para cada funcionalidad. Los primeros cuatro botones servirán para realizar la funcionalidad de lectura sobre los cuatro tipos de registros, mientras que los otros dos ayudarán a realizar el proceso de escritura.

Para la funcionalidad de lectura de los tipos de registros: “COIL STATUS” e “INPUT STATUS” se tendrá un arreglo de 16 *radio button* (*JRadioButton*) para los dos tipos de lectura, tomando en cuenta que estos dos tipos de registros son del tipo binario, es decir 1/0, por lo tanto se consideró que la lectura podría ser mejor representada por un *radio button* donde se refleje el estado binario de 1 o 0.

Para la funcionalidad de escribir en un tipo de registro “COIL REGISTER” se tendrá una caja de texto (*JTextField*)) en la cual se ingresará la dirección de “*coil*” a realizar la escritura, es decir, cambiar su estado de 1 a 0 o viceversa. También, se tendrán dos radio *button*

(*JRadioButton*) en los cuales se elegirá uno de los dos estados a escribir en la dirección de registro ingresada en la caja de texto anteriormente descrita.

Para el proceso de escritura de *holding registers* se va a utilizar la función para escribir múltiples registros. El proceso de escritura se realizará en dos partes.

- La primera, donde se cargará la cantidad de registros a escribir y la dirección de inicio desde donde se va a empezar a escribir los registros. Para esta primera parte la interfaz contará con dos cajas de texto (*JTextField*) en la cuales ingresará la cantidad de registros a escribir y la dirección del registro de inicio desde donde se escribirá. Además de un botón (*JButton*), el cual creará la matriz en donde se almacenarán los valores con los cuales se va a escribir los registros.
- Para la segunda parte del proceso de escritura en los *holdings registers*, se tendrá una caja de texto (*JTextField*) en la cual se ingresarán uno por uno los valores a escribir en los registros. También se tendrá un botón (*JButton*) con el cual se ingresarán los valores de escritura en la matriz creada en el paso anterior. Al terminar de ingresar todos los valores el botón automáticamente se deshabilitará se podrá usar el botón de escribir *holding registers*.

2.3.5 Diseño Módulo Generar Registros

a. Diagramas Módulo Generar Registros

Los diagramas de este módulo representan los procesos, actividades y la interacción entre las clases para que el usuario pueda realizar las tareas de generar históricos con variables específicas y almacenarlas en la base de datos.

b. Diagrama de Casos de Uso Módulo Generar Registros

En la Figura 2.26 se tiene la interacción que efectuará un operador para realizar la funcionalidad de generar históricos de un determinado grupo de variables del cual se está realizando la lectura constantemente.

El operador seleccionará los registros con los cuales quiere generar el histórico, estos registros deben estar previamente leídos con la funcionalidad del prototipo “LeerRegistros”, una vez seleccionados deberá ingresar un tiempo de muestreo es decir cada cuanto tiempo quiere tomar el valor de dichos registros y también un tiempo total por el cual quiere que se ejecute esta funcionalidad.

Siempre el tiempo total de historial debe ser mayor que el tiempo de muestreo total.

Una vez establecidos los tiempos de muestreo y total, entonces se procede a generar el histórico, el prototipo guardará automáticamente esto en la base de datos.

De existir o no algún error al guardar la información, el prototipo mostrará un mensaje de guardado exitoso o fallido dependiendo cuál sea el caso.

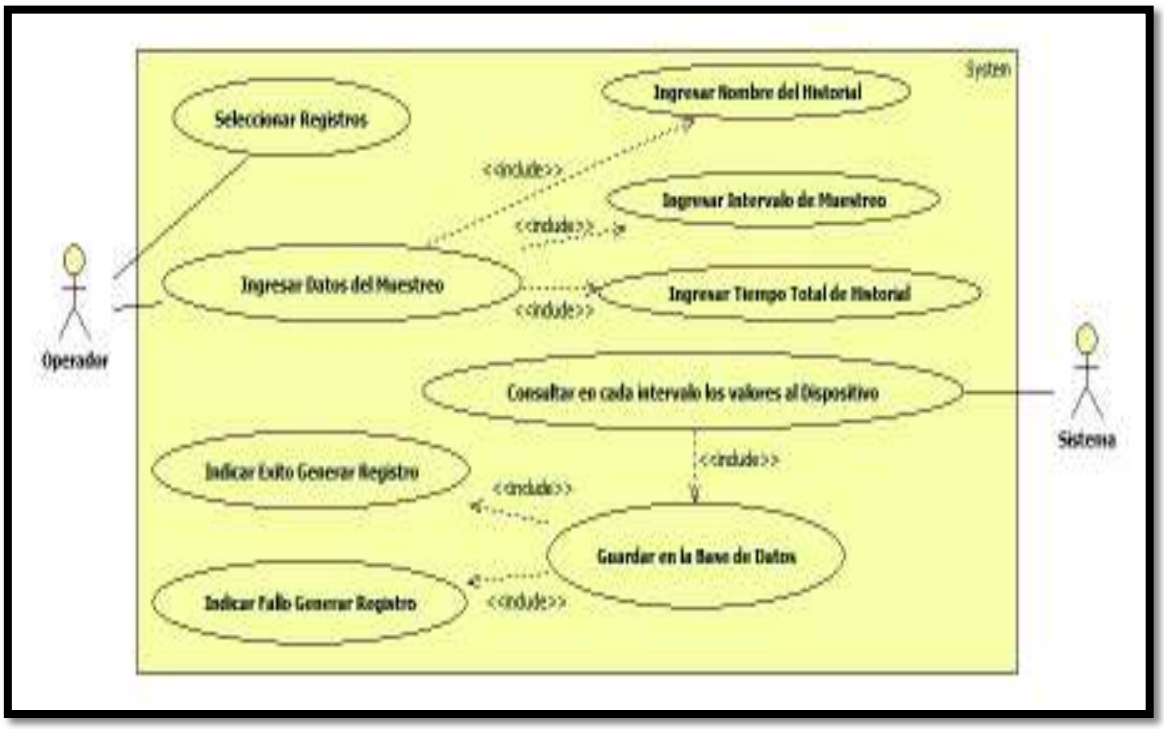


Figura 2.26. Diagrama de Casos de Uso Módulo Generar Registros

c. Diagrama de Actividades Módulo Generar Registros

El diagrama de actividades de este módulo se muestra en la Figura 2.27, aquí se puede observar que el operador debe primero seleccionar los registros de los cuales quiere generara el histórico, luego de eso el prototipo le solicitará al operador que ingrese los tiempos de muestreo y tiempo total del histórico.

Posteriormente el operador ejecutará el método para empezar a generar el histórico, entonces el prototipo de software entrará en un doble bucle, el primero se ejecuta si y solo si se ha cumplido el intervalo de muestreo. Si el intervalo de muestreo se ha cumplido, entonces realiza el proceso de lectura y el valor del contador que se compara con el tiempo de muestreo vuelve a cero.

Además, el prototipo verifica si se ha cumplido con el intervalo total del historial, de cumplirse con esta condición, entonces se finalizará el almacenamiento de los valores en la base de datos, o de lo contrario el bucle vuelve a la primera condición donde se verifica el cumplimiento del intervalo de muestreo.

De no cumplirse la primera condición, el prototipo regresa al mismo punto, verificando si la condición se ha cumplido y el contador de muestreo seguirá incrementando.

Una vez que se cumpla el intervalo del tiempo total entonces se termina el almacenamiento de datos y se actualizará la base de datos.

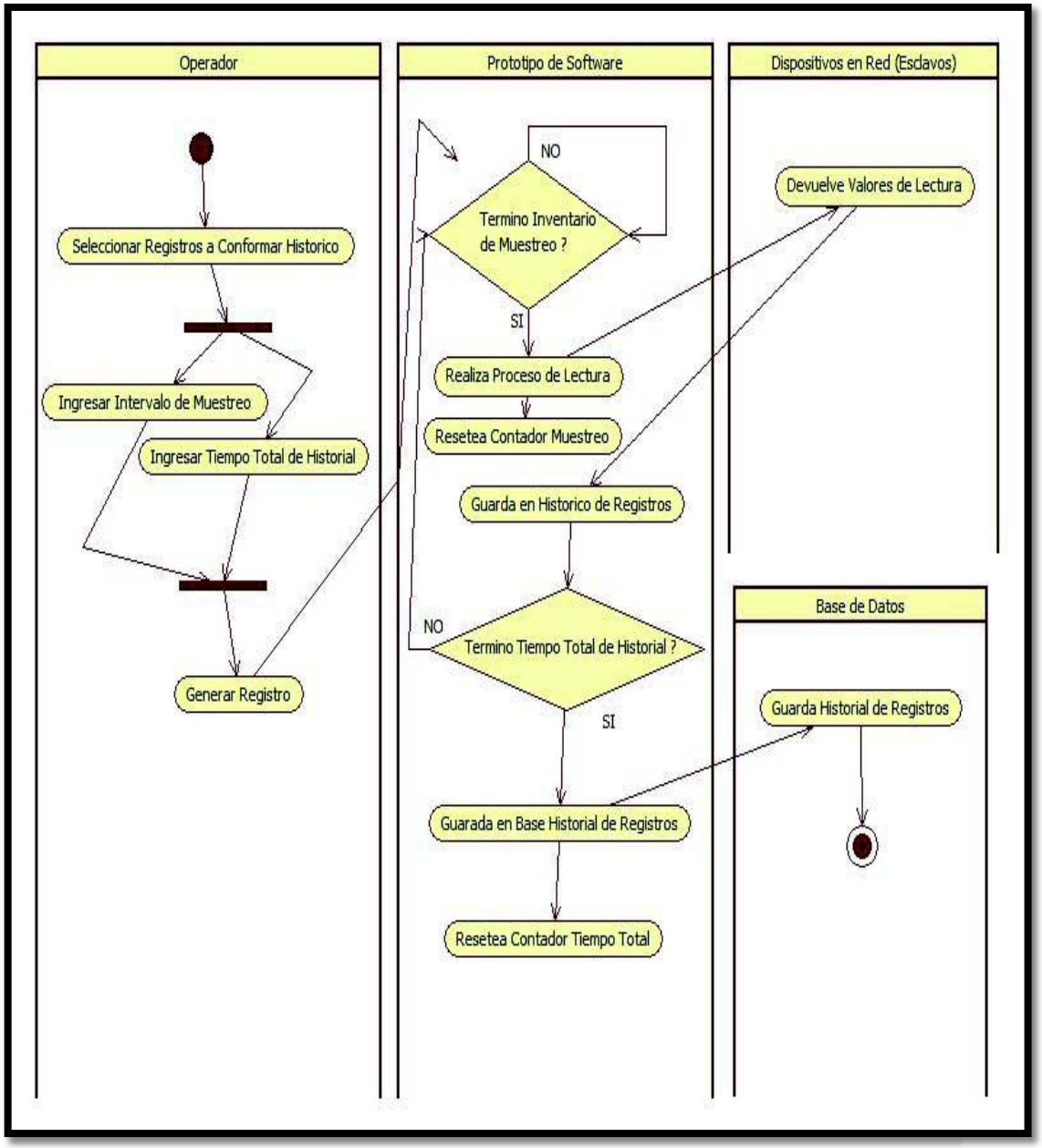


Figura 2.27. Diagrama de Actividades Módulo Generar Registros

d. Interfaz Gráfica Módulo Generar Registros

El diseño de la interfaz gráfica para el módulo de generación de registros (Historiales) se muestra en la Figura 2.28.

Esta interfaz proporcionará a los usuarios la opción de utilizar la funcionalidad de crear un registro de tres variables de interés (que para este caso son: temperatura, nivel del tanque y velocidad de llenado) por un periodo de tiempo.

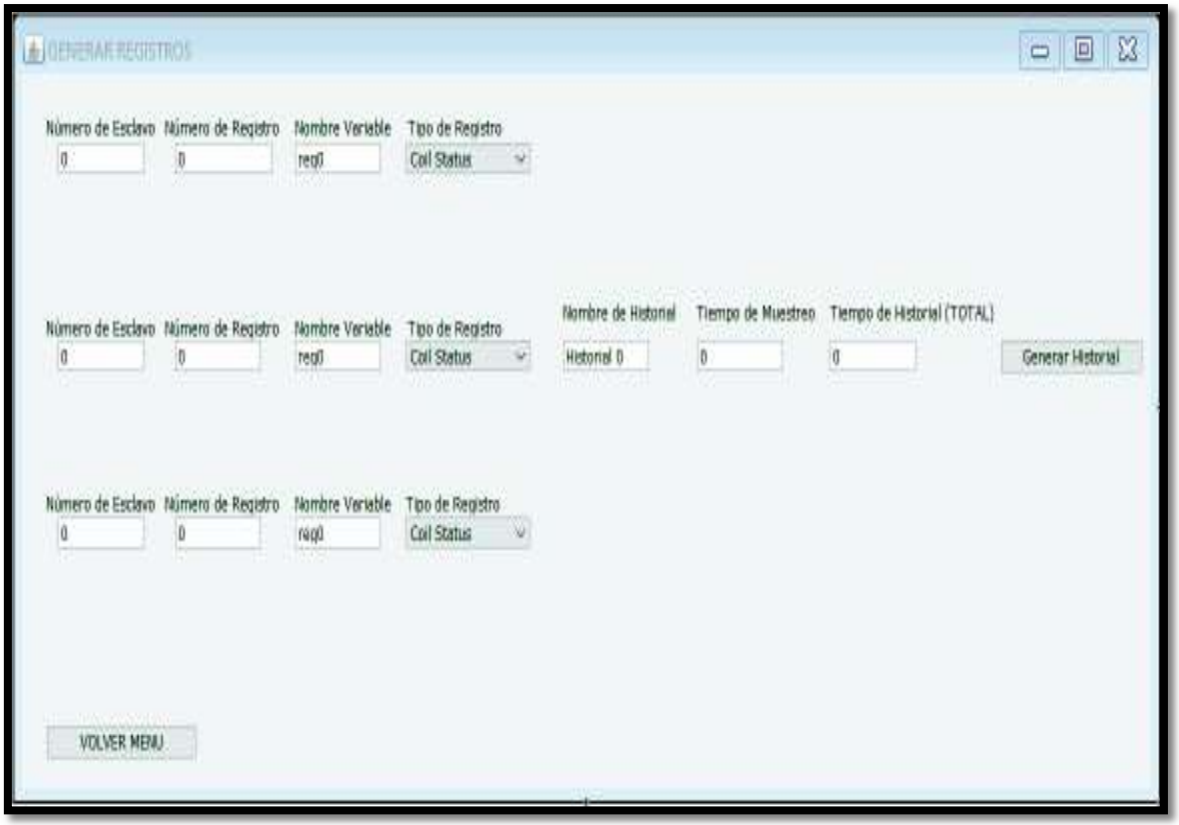


Figura 2.28. Diseño Interfaz Gráfica “GENERAR REGISTROS”

En esta interfaz se tendrán tres pares de elementos gráficos, uno para cada variable, en donde se especificará en tres cajas de texto (*JTextField*) el número de esclavo, número de registro y nombre de variable, en este grupo de elementos también se tendrá un *combo box* (*JComboBox*) en el cual se elegirá cuál de los cuatro tipos de registros quiere leer con los parámetros anteriormente ingresados en las cajas de texto. Esto se repite para cada uno de las tres variables de las cuales se requiere generar el histórico de los registros.

La interfaz también contará con otras tres cajas de texto en las cuales se ingresará el nombre del historial, el tiempo de muestreo y el tiempo total que se va a generar los registros. Además de un botón (*JButton*) con el cual los registros empezarán a guardarse en la base de datos.

Como en interfaces anteriores también cuenta con un botón el cual le permite regresar al menú principal.

2.3.6 Diseño Módulo Base de Datos

a. Interfaz Gráfica Módulo Base de Datos

En la Figura 2.29 se presenta el diseño para realizar la funcionalidad de visualizar las tres tablas “Perfiles”, “Operadores” y “Registros”.

Esta interfaz gráfica contará con tres tablas (*JTable*) principales las cuales mostrarán la información que se encuentra en la base de datos. Además contará con un *combo box* (*JComboBox*) con el cual se selecciona la tabla actualizada que se quiere visualizar, y el botón (*JButton*) para regresar al menú principal.

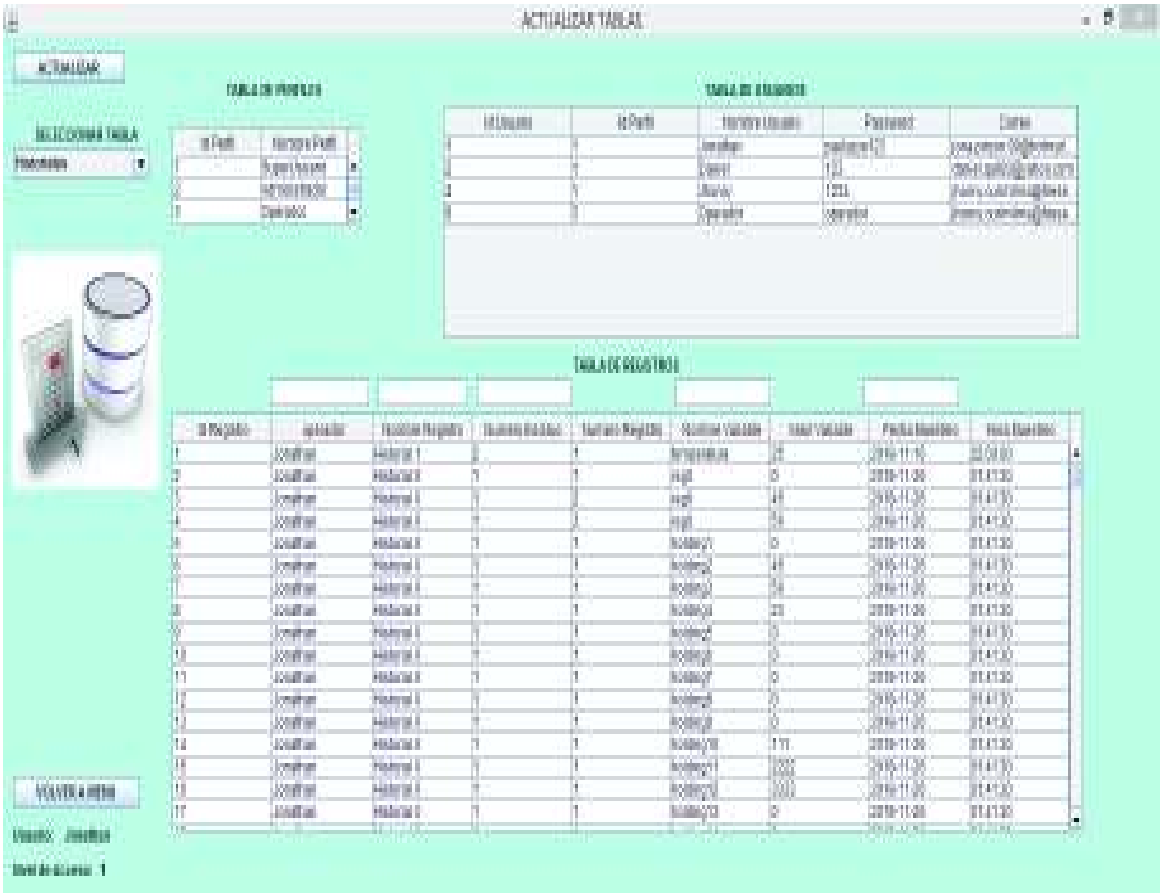


Figura 2.29. Diseño Interfaz Gráfica “ACTUALIZAR TABLAS”

b. Diagrama Relacional de la Base de Datos

A continuación se tiene el diagrama relacional de la base de datos, en la Figura 2.30, en el cual se registrarán los operadores y además se guardarán los históricos de los registros generados con el prototipo de software.

Este diagrama consta de 3 tablas, donde dos son las tablas principales “Usuario” y “Registro”. La tabla “Usuario” está asociada con la tabla “Perfiles” ya que cada usuario debe

contar con un perfil para poder tener un nivel de acceso a las funcionalidades. De igual manera los registros deben ser realizados por un usuario.

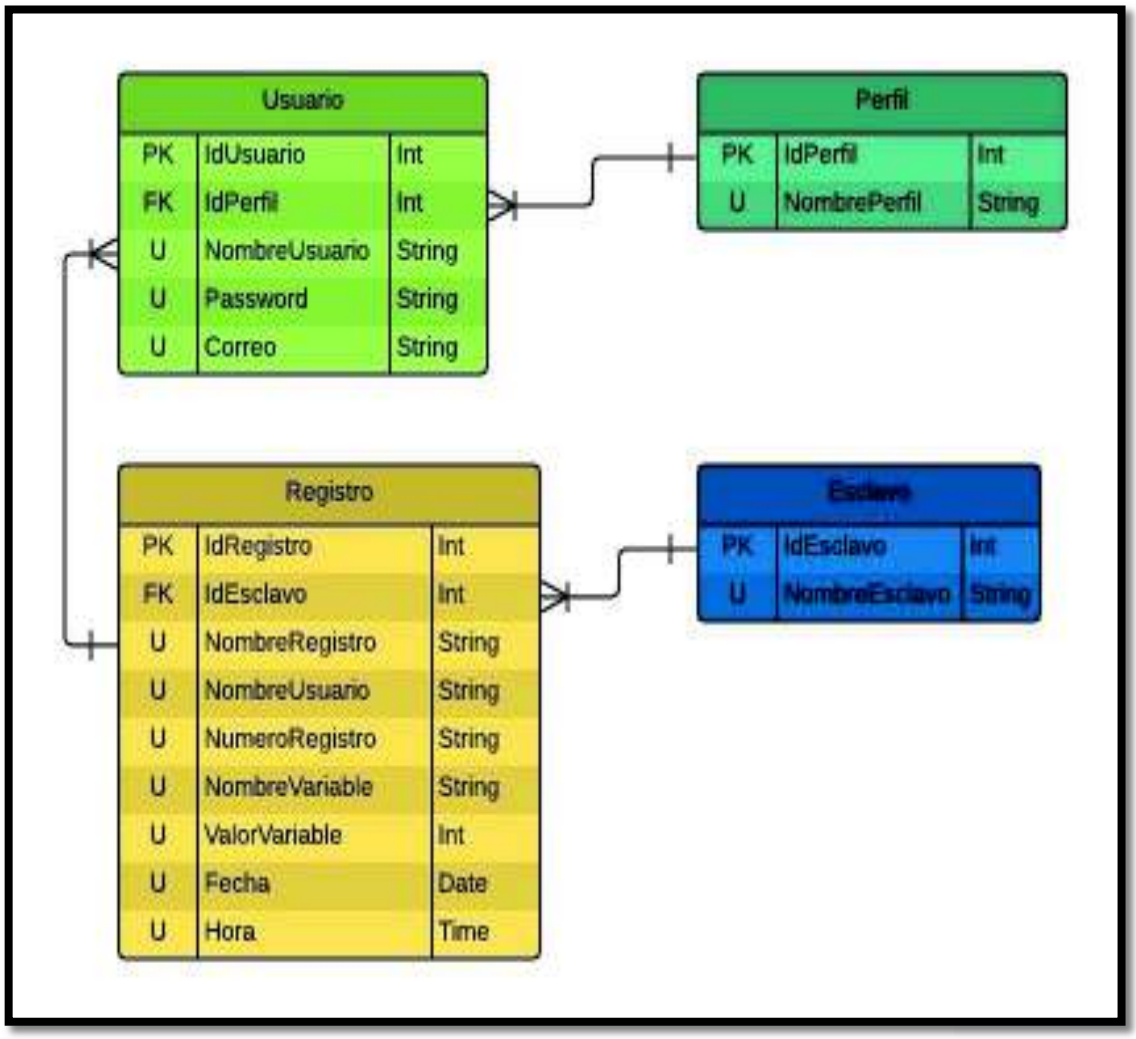


Figura 2.30. Diagrama Relacional

2.3.7 Diseño Módulo HMI

En la Figura 2.31, se detalla el diseño de la interfaz con la cual se cumple la funcionalidad de contar con un HMI especifico en donde se encuentren relacionadas las variables con las cuales se está generando el registro en la base de datos.

Esta interfaz contará con varios *labels* (*JLabel*) a los cuales en su propiedad "*icon*" se cargaran imágenes específicas para construir un historial de temperatura, el cual al pasar un umbral habilite y haga visible a un botón (*JButton*) el cual saltará en señal de que existe o existieron problemas en la temperatura.

En la parte de representar el nivel del tanque se utilizaran *labels* (*JLabel*) para mostrar como el nivel del tanque va subiendo o bajando de acuerdo a la velocidad variable de llenado del tanque y a la velocidad fija de vaciado del tanque.

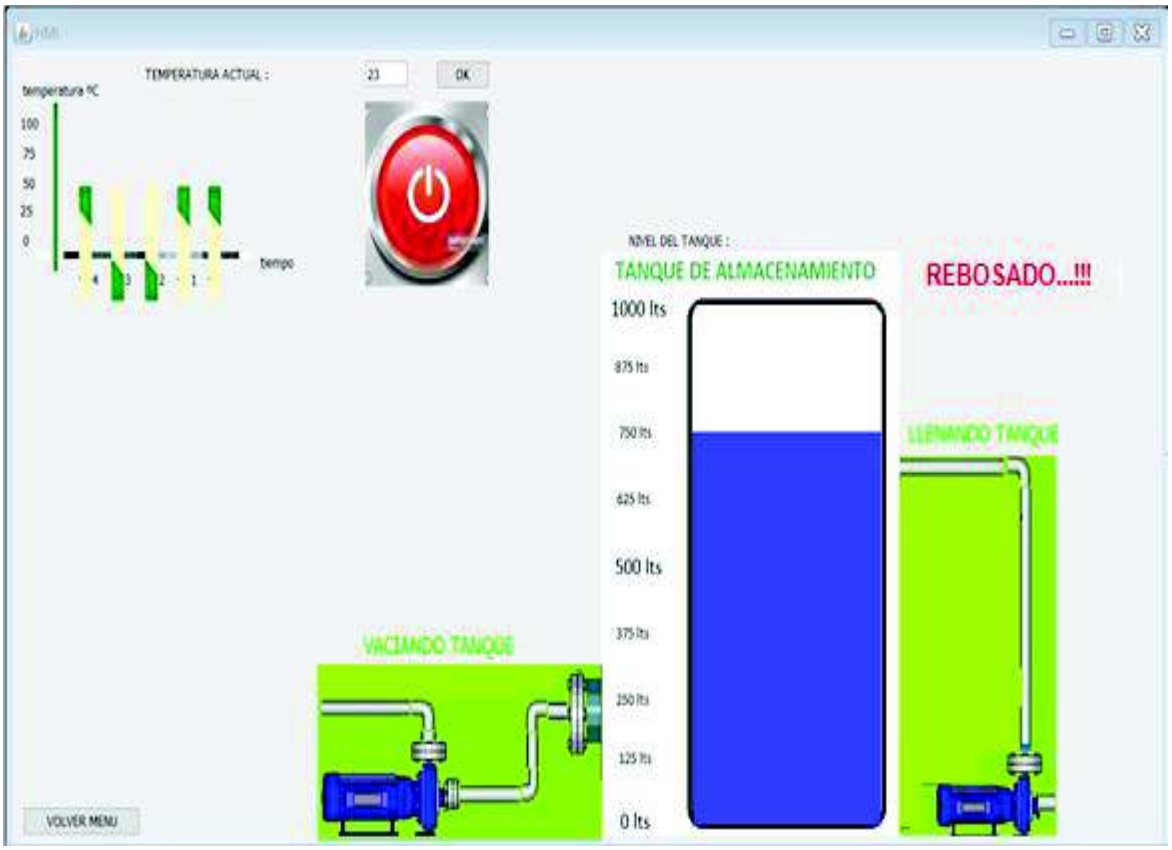


Figura 2.31. Diseño Interfaz Gráfica "HMI"

La interfaz también contará con una caja de texto y un botón con los cuales se pueda modificar el valor del umbral de temperatura.

Adicional a esto se presentará una tabla resumen de la medidas tanto de temperatura, umbral de temperatura, velocidad de llenado del tanque, velocidad de vaciado del tanque y nivel del tanque en tiempo real

2.3.8 Estructura de la Tabla Registros

Previamente con el ING. Jhonny Curimilma se estableció la estructura de la tabla Registros (históricos). La estructura que se muestra en la Figura 2.32 es la siguiente.

- Id Registro: el identificador único de cada uno de los registros que se van a guardar.
- Nombre Registro: Nombre con el que vamos a guardar el historial.
- Número Esclavo: de que esclavo estamos tomando el valor a guardar.
- Número Registro: la dirección del registro que se está guardando.
- Nombre Variable: el nombre que toma cada una de las variables que se guardan en la base de datos.
- Valor Variable: el valor de la variable.

- Fecha de Muestreo: la fecha en que fue tomada la lectura de esa variable.
- Hora de Muestreo: hora en la que la variable fue leída.

IdRegistro	nombreRegistro	numeroEsclavo	numeroRegistro	nombreVariable	valorVariable	fechaMuestreo	horamuestra
1	reg1	1	10	temperatura	100	2018-10-25	0000-00-00 00:00:00
2	HISTORIAL	2	1	temperatura	100	2018-11-03	02:59:18

Figura 2.32. Formato de Información Históricos

2.3.9 Diagrama de Clases

En la figura 2.33 se tiene el diagrama de clases completo, en este se muestra las interacciones entre todas las clases que conforman el prototipo de software, además de las relaciones entre las mismas.

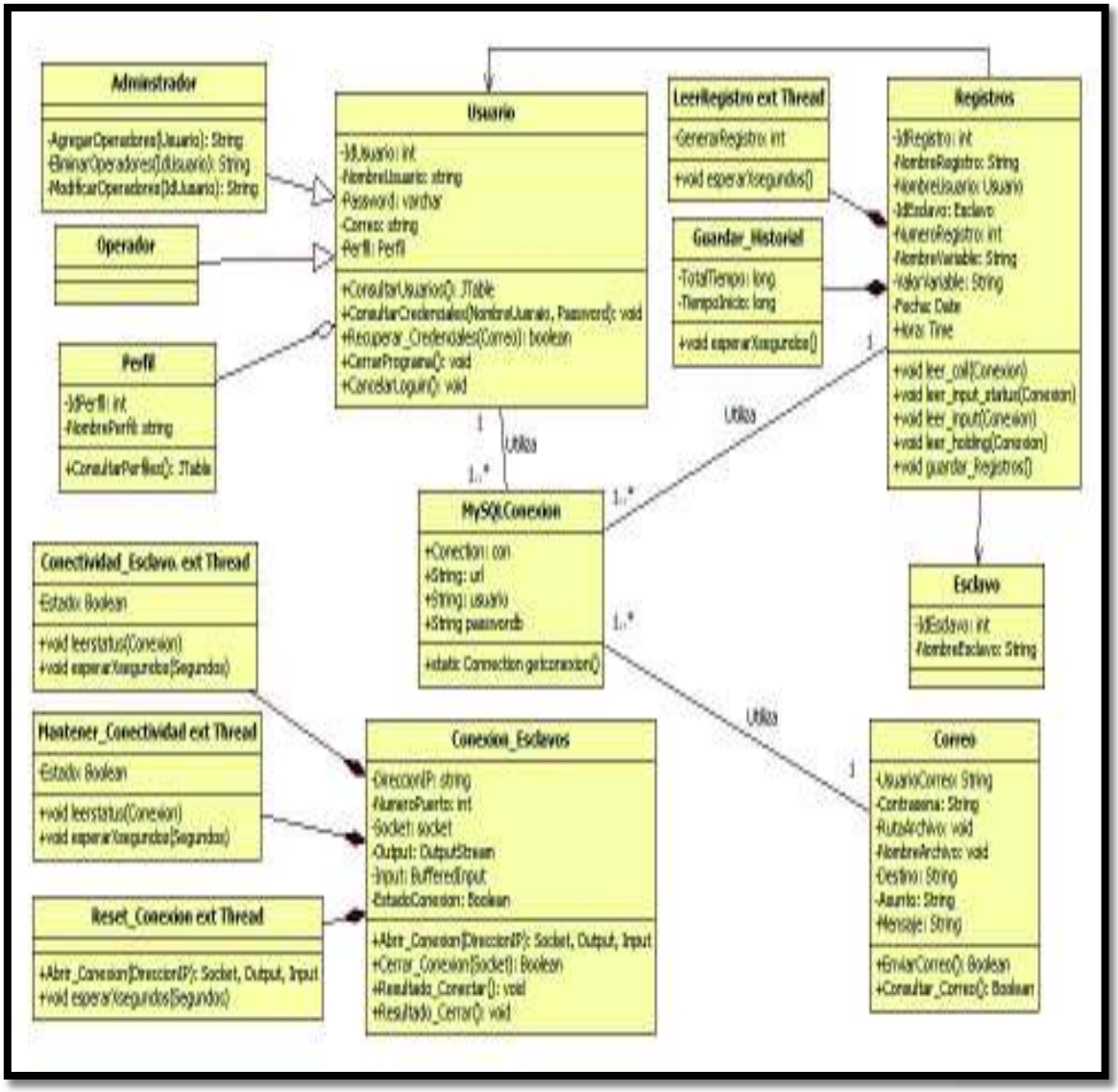


Figura 2.33. Diagrama de Clases Genera

3 IMPLEMENTACIÓN

3.1 Actualización del Tablero Kanban

Una vez terminada la fase de diseño las tareas del tablero cambian de columna, es decir que las tareas asociadas con la fase de diseño pasan de estar en el estado “Tareas en Proceso” al estado “Tareas Realizadas”. Mientras que las tareas asociadas con la fase de implementación ahora están en el estado “Tareas en Proceso”, tal como se muestra en la Figura 3.1.

Figura 3.1. Actualización del Tablero Kanban

3.2 Implementación del Prototipo de Software

En este subcapítulo se mostrará cómo se desarrolló la codificación de todas las clases, métodos e interfaces gráficas que forman parte de cada uno de los módulos que componen al prototipo de software.

A continuación se detallará las codificaciones de los módulos en el mismo orden pero con la diferencia que primero será el Módulo Base de Datos. Esto debido a que algunas de las funcionalidades realizan consultas a la base de datos, por tal motivo el orden de codificación de los módulos sería el siguiente.

- Módulo Base de Datos
- Módulo Registro/Acceso Usuarios
- Módulo Comunicación/Conexión
- Módulo Lectura/Escritura
- Módulo Generar Registros
- Módulo GUI

Como una pequeña introducción se debe mencionar que el prototipo de software está codificado usando el lenguaje JAVA y el entorno de desarrollo integrado (IDE) ECLIPSE.

Dentro de la utilización del IDE ECLIPSE se utiliza el *plugin WindowBuilder*, el cual permite crear interfaces gráficas de manera más rápida y fácil.

WindowBuilder[25]

WindowBuilder es un potente y fácil de usar diseñador bidireccional de GUI de Java.

Está compuesto por *SWT Designer*[10] y *Swing Designer*[11] que facilitan mucho la creación de aplicaciones Java GUI sin perder mucho tiempo escribiendo código.

Se usa el diseñador visual y las herramientas de diseño *WYSIWYG*[12] para crear formularios simples para ventanas complejas; el código de Java se generará automáticamente.

Permite agregar fácilmente controles con la función de arrastrar y soltar, agregar controladores de eventos a sus controles, cambiar varias propiedades de los controles con un editor de propiedades, y mucho más.

WindowBuilder está construido como un complemento para Eclipse y los diversos IDE basados en Eclipse (*RAD, RSA, MyEclipse, JBuilder,* etc.). El complemento crea un árbol

[10] SWT Designer es un editor visual que se utiliza para crear interfaces gráficas de usuario. Es un analizador bidireccional, por ejemplo, puede editar el código fuente o usar un editor gráfico para modificar la interfaz de usuario. SWT Designer sincroniza entre ambas representaciones[26].

[11] Visual Swing Designer for Eclipse, es una herramienta de diseño de GUI, que consiste en un conjunto de complementos de Eclipse. Su objetivo es proporcionar un diseñador Swing para Eclipse para desarrolladores de escritorio Java.

[12] WYSIWYG, acrónimo de What You See Is What You Get (en español, "lo que ves es lo que obtienes"), es una frase aplicada a los procesadores de texto y otros editores de texto con formato (como los editores de HTML) que permiten escribir un documento mostrando directamente el resultado final, frecuentemente el resultado impreso.

de sintaxis abstracta (AST) para navegar por el código fuente y usa *GEF*[13] para mostrar y administrar la presentación visual. En la Figura 3.2 se muestra un entorno de desarrollo de WindowBuider.

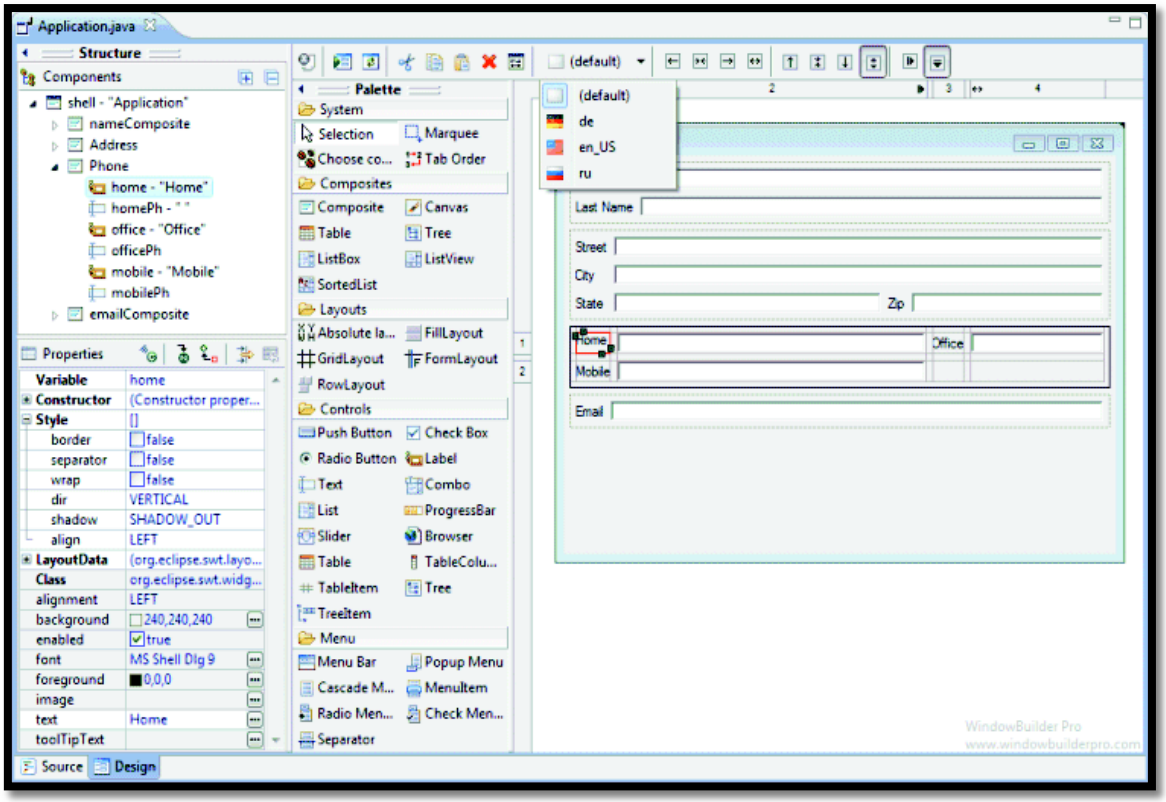


Figura 3.2. Entorno de Desarrollo WindowBuilder

En la parte de la base de datos se utilizó el sistema de gestión de base de datos relacional MySQL en conjunto con la herramienta visual de diseño *MySQL WorkBench* [27] para tener un entorno gráfico de desarrollo y sea más fácil la implementación de la base de datos.

3.2.1 Codificación Módulo Base de Datos

En la codificación de este módulo se muestra las partes más importantes en cuanto a la codificación y creación de la base de datos, las tablas y la clase que proporciona la conexión desde el prototipo de software hacia la base de datos.

a. Creación de la Base de Datos

En la Figura 3.3 se muestra el primer paso para crear la base de datos. Primero, ejecutar *MySQL Workbench* y abrir la instancia de conexión local. Para el alcance de este trabajo

[13] El Marco de Edición Gráfica (GEF) de Eclipse proporciona herramientas integradas para el usuario final de Eclipse en términos de creación de Graphviz (editor DOT, vista de gráfico DOT) y un entorno de representación de nube de palabras (vista de nube de etiquetas), así como componentes de marco (común , Geometría , FX , MVC , Graph , Layout ,Zest , DOT y Cloudio) para crear aplicaciones cliente gráficas Java enriquecidas, integradas con Eclipse o independientes.

de titulación no es necesario contar con instancias de conexión externas. Aquí se ingresa la contraseña, establecida al momento de instalar *MySQL Workbench*, se ingresa al entorno de desarrollo donde se crea la base de datos y las tablas.

Figura 3.3. Página de Inicio de *MySQL Workbench*

Una vez dentro del entorno de desarrollo de *MySQL Workbench* se crea un nuevo *schema*, es decir una nueva base de datos, como se muestra en la Figura 3.4

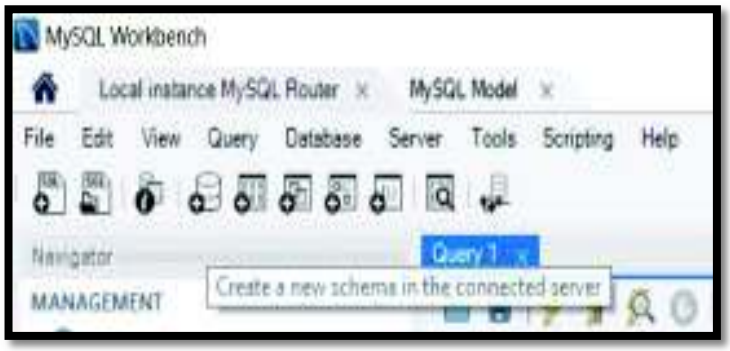


Figura 3.4. Creación *Schema* (Base de Datos)

Seleccionando la opción de crear una nueva base de datos (*Schema*) automáticamente se tiene una base de datos con el nombre por defecto "*mydb*", a continuación se cambia ese nombre por el que se quiere y presionamos aplicar (*Apply*) como se muestra en la Figura 3.5. En la parte izquierda dentro de *Schemas* se puede observar la base de datos que acabamos de crear junto con las bases de datos que vienen creadas por defecto para ejemplos.

b. Creación de Tablas Usuario, Perfil y Registro

Para realizar la creación de una nueva tabla se realiza los pasos que se muestran en la Figura 3.6.

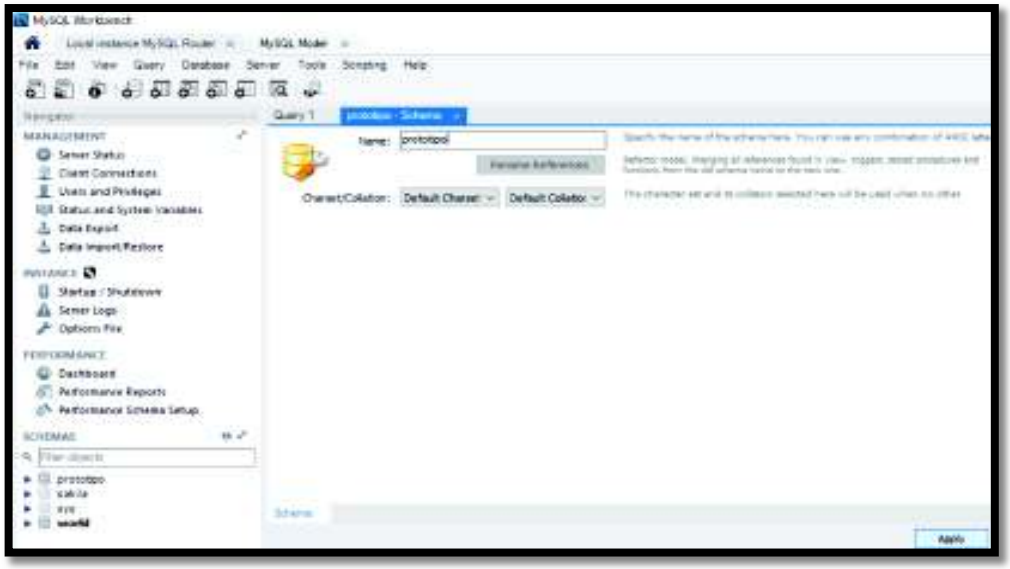


Figura 3.5. Creación de la Base de Datos “Prototipo”

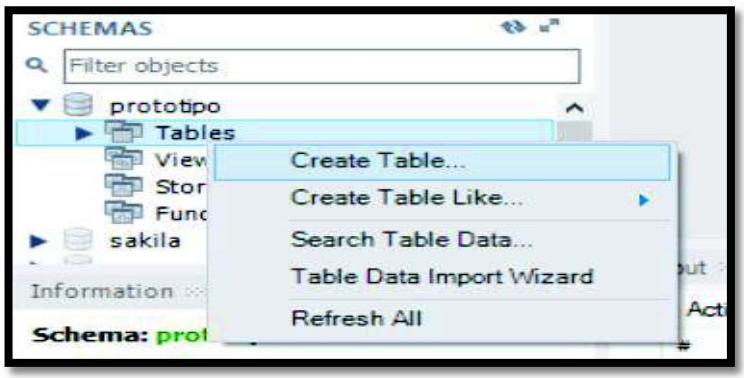


Figura 3.6. Creación Tabla

Se selecciona la base que se acabo de crear, luego *click* derecho sobre tablas, aquí se selecciona la opción de crear tabla y se crea las tres tablas anteriormente mencionadas.

En la Figura 3.7 se muestra la creación de la tabla Perfil, la creación de perfiles permite tener un nivel de acceso de los usuarios hacia las funcionalidades del Prototipo de Software. Esta tabla tiene un Id de perfil, el cual es la clave primaria y también será clave foránea en la tabla usuario adicional a esto se tiene un nombre de perfil. Ambos atributos deben ser *not null*, es decir no aceptan un *null* como valor de entrada.

En la Figura 3.8 se muestra la creación de la tabla Registro. En esta tabla se almacenan los registros creados (Historiales) de las variables: temperatura, nivel de tanque y caudal de entrada del tanque.

Esta tabla cuenta con Id de registro el cual es la clave primaria, el nombre de registro a crear, número de esclavo de donde se está tomando la lectura del registro, número de

registro que se está leyendo, nombre de la variable que se va a guardar, el valor de la variable, fecha y hora de muestreo.

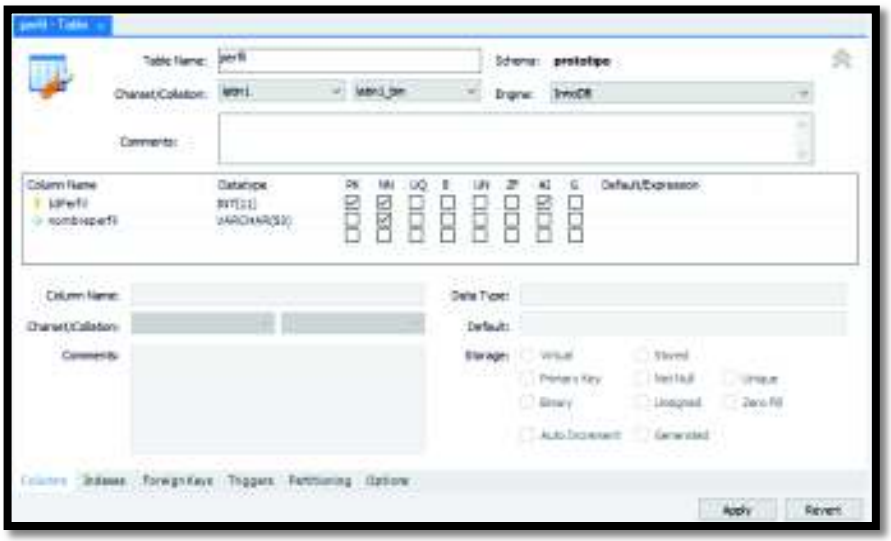


Figura 3.7. Creación Tabla Perfil

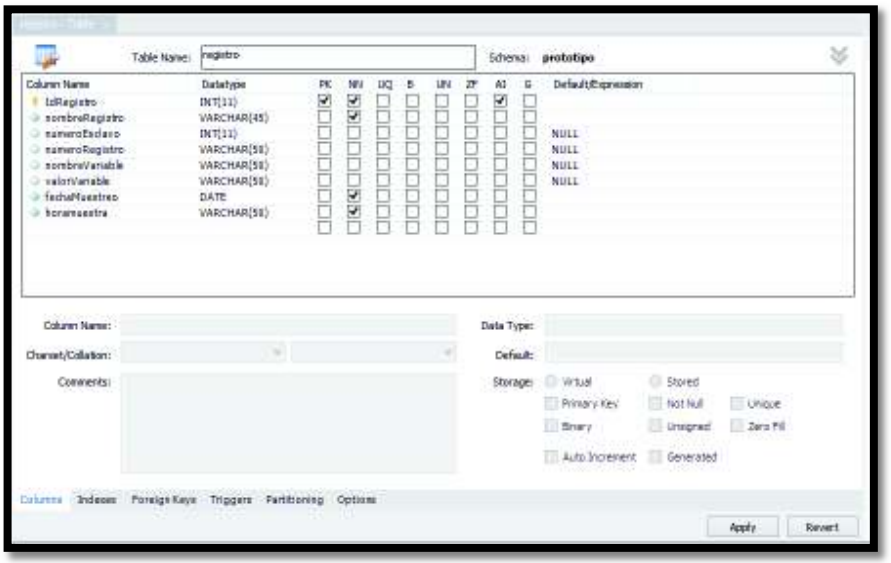


Figura 3.8. Creación Tabla Registro

En la Figura 3.9 se muestra la creación de la tabla Usuario. En esta tabla se almacena información de los operadores que van a utilizar el prototipo de software, y de acuerdo a un determinado perfil podrán acceder a todas o algunas funcionalidades.

Esta tabla cuenta con IdUsuario que es la clave primaria; un id de Perfil que es una clave foránea debido a la relación con la tabla Perfiles; un nombre de usuario que debe ser del tipo *unique*, es decir no pueden existir dos usuarios con el mismo nombre de usuario,

password y un correo electrónico el cual será indispensable para utilizar la funcionalidad de recuperar credenciales.

Todos los atributos de la tabla son *not null*, es decir que no aceptan un valor nulo como parámetro de entrada.

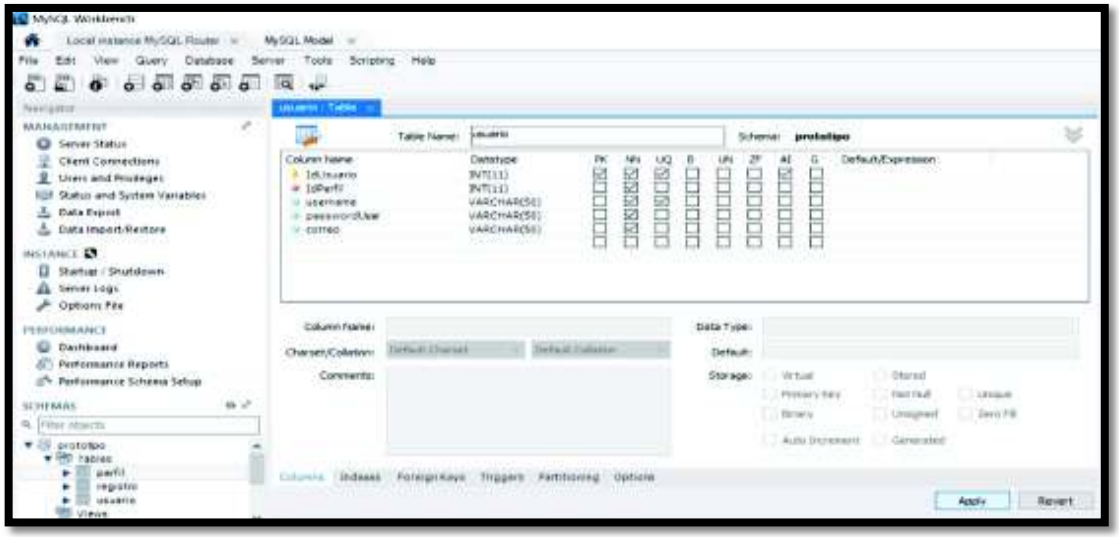


Figura 3.9. Creación Tabla Usuario

Como resultado se tiene la base de datos creada con cada una de las tablas, tal como se muestra en la Figura 3.10.

Figura 3.10. Resumen Creación Base de Datos

c. Codificación Clase *MySQLConexion*

En la Código 3.2 se muestra la codificación utilizada para establecer la conexión con la base de datos.

Para empezar se debe mencionar los paquetes de clases utilizados dentro de esta aplicación: *java.sql* y *javax.sql*, los cuales están dentro de la librería *mysql-connector-java-8.0.12.jar*. Esta librería está añadida y cargada en el proyecto PrototipoSoftware tal como se muestra en la Figura 3.11.

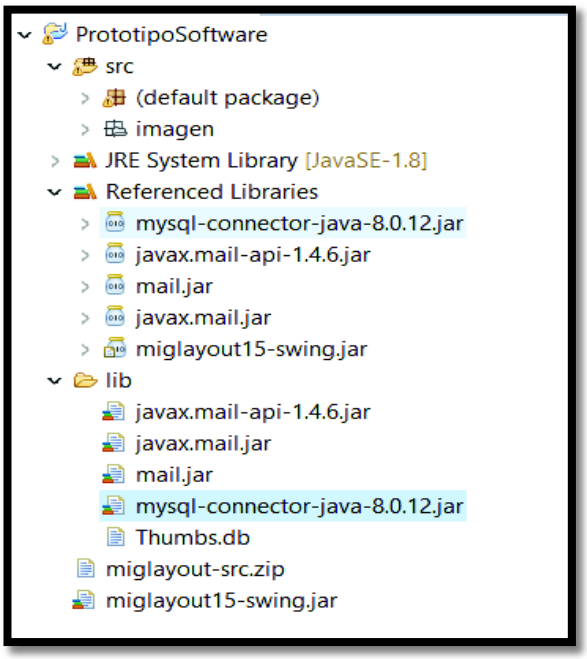


Figura 3.11. Añadir Librería mysql-connector-java-8.0.12.jar

Como se muestra en el Código 3.2, para conectarse a la base de datos es necesario cargar el driver el cual es el encargado de realizar ésta función, para ello se utiliza *Class.forName*.

En este caso el *driver JDBC-MySQL* es: *com.mysql.jdbc.Driver* [28].

Una vez cargado el *driver* es necesario crear un objeto del tipo *Connection*, para administrar la conexión. Se utiliza *DriverManager* para obtener un objeto del tipo *Connection* con una base de datos. Y se pasa los parámetros: URL, Nombre de Usuario y password. Como se muestra en el Código 3.1.

```
Connection con = DriverManager.getConnection(URL, "root", "********");
```

Código 3.1. Creación de Objeto *Connection*

Si el *driver* reconoce la URL suministrada por el método *DriverManager.getConnection*, dicho driver establecerá una conexión con el controlador de base de datos especificado en la URL del JDBC. La clase *DriverManager* como su nombre indica, maneja todos los detalles del establecimiento de la conexión por detrás de la escena.

La conexión devuelta por el método *DriverManager.getConnection* es una conexión abierta que se puede utilizar para crear sentencias JDBC que pasen las sentencias SQL al controlador de la base de datos.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import javax.swing.JOptionPane;
public class MySQLConexion {

    public static Connection getconexion(){
        Connection con = null;
        try {
            Class.forName("com.mysql.cj.jdbc.Driver"); // Driver para generar la conexion con la base de datos

            //Parametros para pasar al objeto del tipo  Connection
            String url = "jdbc:mysql://localhost/Prototipo"+"?useTimezone=true&serverTimezone=UTC&useSSL=false";
            String usuario = "root";
            String passwordb ="paulazoe123";

            con = DriverManager.getConnection(url,usuario,passwordb); // Generar la conexion

        } catch (ClassNotFoundException e) {
            JOptionPane.showMessageDialog(null, "NO SE HA PODIDO CONECTAR A LA BASE DE DATOS");
            e.printStackTrace();
        } catch (SQLException e) {
            JOptionPane.showMessageDialog(null, "NO SE HA PODIDO CONECTAR A LA BASE DE DATOS");

            e.printStackTrace();
        }
        return con;
    }

    }
```

Código 3.2. Clase *MySQLConexion*

d. Codificación Clase Actualizar Tablas

La interfaz gráfica Actualizar_Tablas hereda de *JFrame*, es la encargada de mostrar al usuario las tres tablas que conforman la base de datos. Cuenta con un combo box (*JComboBox*) en el cual se elige que tabla debe ser actualizada y visualizada.

Como se muestra en el Código 3.3, dependiendo de la opción elegida desde el combo box se actualizará y mostrará una tabla en particular, llamando a cada uno de los métodos Actualizar_Perfiles, Actualizar_Operadores o Actualizar_Historiales.

En el Código 3.4 se muestra la codificación de la función Actualizar_Operadores.

Se realiza una instancia de la clase Usuario y se llama al método ConsultarUsuarios(); este método devuelve un objeto del tipo tabla (*JTable*).

```
JComboBox comboBox = new JComboBox();
comboBox.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {
        String opcion = comboBox.getSelectedItem().toString();
        if (opcion == "Perfiles"){
        Actualizar_Perfiles();
        } else if (opcion == "Usuarios"){
        Actualizar_Operadores();

        } else if(opcion == "Historiales"){
        Actualizar_Historiales();
        }

    }
```

Código 3.3. Codificación del *ComboBox*

Una vez obtenida esta tabla se le añade a un *JScrollPane* y al panel principal.

```
public void Actualizar_Operadores() {
    Usuario buscar_usuario = new Usuario();
    JTable tab_usuario = buscar_usuario.ConsultarUsuarios();
    JScrollPane scrollPaneusuario = new JScrollPane(tab_usuario);
    scrollPaneusuario.setBounds(520, 50, 750, 168);
    getContentPane().add(scrollPaneusuario);
}
```

Código 3.4. Función Actualizar_Operadores de la Clase Actualizar_Tablas

En el Código 3.5 se muestra la codificación del método ConsultarUsuarios(). Este método crea una instancia del método *MySQLConexion*, explicado anteriormente, y sobre esa conexión realizar la consulta.

```
"SELECT * FROM usuario"
```

Código 3.5. Solicitar Todos los Valores de la Tabla Usuario

Posteriormente se crea una matriz para almacenar cada uno de los valores devueltos al realizar la consulta y guardar esta matriz en una tabla la cual será retornada al ejecutar este método como se muestra en el código 3.6.

De no ejecutarse con éxito la consulta o el guardar la información en la tabla, se muestra un mensaje informando que la operación no se realizó con éxito.

```
public JTable ConsultarUsuarios() {
    Connection con_usuario =null;
    JTable tabla_usuario = new JTable();
    try {
        String IdUsusario,IdPerfil,username,passwordUser,correo;
        con_usuario = MySQLConexion.getconexion();
        Statement st_usuario = con_usuario.createStatement();
        ResultSet rs_usuario = st_usuario.executeQuery("SELECT * FROM usuario");
        String columnas_usuario [] = {"Id Usuario","Id Perfil", "Nombre Usuario", "Password", "Correo"};

        DefaultTableModel modelo_usuario = new DefaultTableModel();
        modelo_usuario.setColumnIdentifiers(columnas_usuario);
        tabla_usuario.setModel(modelo_usuario);
        while(rs_usuario.next()){
            IdUsusario= rs_usuario.getString("IdUsuario");
            IdPerfil = rs_usuario.getString("IdPerfil");
            username= rs_usuario.getString("username");
            passwordUser = rs_usuario.getString("passwordUser");
            correo = rs_usuario.getString("correo");
        modelo_usuario.addRow(new Object[]{IdUsusario,IdPerfil,username,passwordUser,correo});
        }
    } catch (SQLException e) {
        JOptionPane.showMessageDialog(null,"NO SE PUDO REALIZAR LA OPERACION DE CONSULTAR USUARIOS");
        e.printStackTrace();
    }
    return tabla_usuario;

    }
```

Código 3.6. Codificación Método ConsultarUsuarios

Tanto las funciones Actualizar_Perfiles y Actualizar_Historiales, así como los métodos ConsultarPerfiles y ConsultarHistoriales realizan procesos idénticos a Actualizar_Operadores y ConsultarOperadores respectivamente; por tal motivo el detalle de sus codificaciones se omitirá y la codificación completa de todas estas funciones se presentarán en el ANEXO C.

3.2.2 Codificación Módulo Registro/Acceso Usuarios

A continuación se muestra la codificación de las dos interfaces gráficas que conforman el módulo Registro/Acceso de Usuarios: Login y RecuperarContraseña.

a. Codificación Clase Login

La clase Login hereda de *JFrame* ya que es una interfaz gráfica en la cual el usuario registra su nombre de usuario y password para acceder a las funcionalidades del prototipo.

Esta clase debe importar la librería mysql-connector-java-8.0.12.jar, ya que se realizarán consultas en la base de datos. En el Código 3.7 se muestran los paquetes que son importados.

```
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
```

Código 3.7. Paquetes de *java.sql* Clase Login

La clase Login es la única en todo el prototipo que tiene el método *main* ya que el prototipo siempre iniciará desde esta clase. La funcionalidad del método *void main* es sencillo, muestra la interfaz gráfica para que el usuario registre sus credenciales y utilice el prototipo de software.

La codificación de la clase *void main* se muestra en el Código 3.8.

```
public static void main(String[] args) {
	EventQueue.invokeLater(new Runnable() {
		public void run() {
			try {
				Login frame = new Login();
				frame.setVisible(true);

			} catch (Exception e) {
				e.printStackTrace();
			}
		}
	});
}
```

Código 3.8. Codificación Método *main*

Como se definió en la fase de diseño, la interfaz LOGIN cuenta con dos cajas de texto donde se ingresa el nombre de usuario y password, el proceso para validar estos valores en la base de datos está dentro del botón “INGRESAR”, en caso de querer suspender el proceso de login se lo realiza con el botón “CANCELAR”.

Dentro del botón “INGRESAR” se hace el llamado a la función consultar_credenciales(), su codificación se muestra en el Código 3.10. Aquí se tiene los principales procesos que se realizan dentro de esta función.

El primer proceso es la preparación de la conexión, para esto se declara la variable “con_login” del tipo *Connection* sobre la cual se realizará una consulta hacia la base de datos.

También se elabora la consulta hacia la base de datos la cual tiene la finalidad de obtener el *password* que corresponda al nombre de usuario ingresado.

La consulta se muestra en el Código 3.9.

```
"SELECT passwordUser FROM usuario WHERE username = ?"
```

Código 3.9. Consulta Password

Una vez estructurada la consulta se la envía sobre la conexión a la base de datos previamente creada, y se compara el password devuelto de la consulta con el password ingresado en la caja de texto.

Si los dos password coinciden, se accede exitosamente a las funcionalidades del prototipo de software y se lanza la ventana del menú principal. De no coincidir o de obtener un valor *null* al momento de realizar la consulta se muestra la opción para recuperar credenciales a través de correo electrónico y un mensaje informando que las credenciales son incorrectas.

Este proceso se lo muestra en los códigos 3.10 y 3.11.

El botón “CANCELAR” realiza el proceso de abortar la validación de credenciales al limpiar las cajas de texto en las cuales se ingresó el nombre de usuario y *password.*

```
public void consultar_credenciales() {
    //Creacion de la conexion: con_login de la clase MySQLConexion
    Connection con_login =null;
    String nombreusuario, password, consulta, passworddl, perfil;
    PreparedStatement st_login =null;
    ResultSet rs_login = null;
    try {
        //Elaboracion y ejecucion de la consulta sobre la conexion creada
        nombreusuario = txtnombreusuario.getText();
        password = txtpassword.getText();
        con_login = MySQLConexion.getconexion();
        //La consulta que se realiza hacia la base obtiene el password del nombre de usuario que se
        // ingreso en el cuadro de texto
        consulta = "SELECT passwordUser, IdPerfil FROM usuario WHERE username = ?";
        st_login = con_login.prepareStatement(consulta);
      st_login.setString(1,nombreusuario);
        rs_login = st_login.executeQuery();
```

Código 3.10. Codificación Función consultar_credenciales()

```
      if (rs_login.next()== true){
                        password1 = rs_login.getString("passwordUser");
      //Si los dos password coinciden, entonces se lanza la ventana del menu principal
                        if(password1.equals(password)){
                              log=true;
                              perfilu = rs_login.getString("IdPerfil");
                    useru = nombreusuario;
                              Menu ventana_menu = new Menu ();
                              ventana_menu.setVisible(true);
                              Login.this.dispose();
            //En caso de que las credenciales no coincidan se habilita la opcion para recuperar las
            // credenciales Y aparece un mensaje de que las credenciales ingresadas son incorrectas

                        }else {
                              lblerroru.setVisible(true);
                              btnRecuperarCredenciales.setVisible(true);
                        }
      }else {
            lblerroru.setVisible(true);
            btnRecuperarCredenciales.setVisible(true);
      }
} catch (SQLException e1) {
      e1.printStackTrace();
} }
```

Código 3.11. Codificación Función consultar_credenciales() Validación

b. Codificación Clase Recuperar_password

Al activar el botón para recuperar credenciales se muestra una interfaz gráfica donde se ingresa el correo de usuario, del cual se quiere recuperar las credenciales, y se llama a la función Recuperar_contrasena() la cual realiza los procesos para validar el correo electrónico dentro de la base de datos y enviar las credenciales al correo con el que la validación se realizó exitosamente.

En los Códigos 3.12 y 3.13 se observa la codificación de la función Recuperar_contrasena() la cual realiza el proceso de elaborar el formato (*body*) del correo electrónico y enviarlo.

```
public void Recuperar_contrasena() {
//Se prepara la conexion a la base de datos
Connection con_rec =null;
   String consulta,correo,resultadousername, resultadopassword;
   PreparedStatement st_rec =null;
   ResultSet rs_rec = null;
   //Se obtiene el correo a validar
   correo= txtcorreo.getText();
   try {
         // Se establece la conexion y se realiza la consulta para que devuelva el nombre de usuario y
         //password asociados a ese correo
         con_rec = MySQLConexion.getconexion();
         consulta = "SELECT  username,passwordUser FROM usuario WHERE correo = ?";
         st_rec = con_rec.prepareStatement(consulta);
         st_rec.setString(1,correo);
               rs_rec = st_rec.executeQuery();
```

Código 3.12. Codificación Función Recuperar_contrasena

```
            //Se obtiene los valores devueltos en la consulta y se crea una instancia de la clase
            // Correo esto con el motivo de utilizar su metodo Enviar_Correo y pasarle como
            // parametros:El correo validado
            //Y las credenciales recuperadas desde la base de datos
            if(rs_rec.next()==true){
                  resultadousername= rs_rec.getString("username");
                  resultadopassword= rs_rec.getString("passwordUser");
                  Correo c = new Correo();
                  respuesta_correo= c.Enviar_Correo(correo, resultadousername, resultadopassword);
                  if (respuesta_correo==true){
                        lblLasCredencialesFueron.setVisible(true);
            //Si es el caso de no tener ese correo registrado en la base de datos o de no haber
      // podido realizar el envío del correo electrónico se mostrará un mensaje de advertencia
                  }else {
                        JOptionPane.showMessageDialog(null, "EL CORREO DE RECUPERACION NO HA PODIDO
            SER ENVIADO");
                  }
            }else{
            lblElCorreoIngresado.setVisible(true);
      JOptionPane.showMessageDialog(null, "EL CORREO INGRESADO NO EXISTE EN LA BASE DE DATOS");
            }
} catch (SQLException e1) {

} }
```

Código 3.13. Codificación Función Recuperar_contrasena Validación

Como se observa en el código anterior, se realiza una serie de procesos para implementar la funcionalidad de recuperar las credenciales a través del correo electrónico.

El primero proceso es crear una instancia de la clase *MySQLConexion* sobre la cual se realiza la consulta para recuperar las credenciales, segundo se estable la conexión y sobre esta se realiza la consulta que se muestra en el Código 3.14.

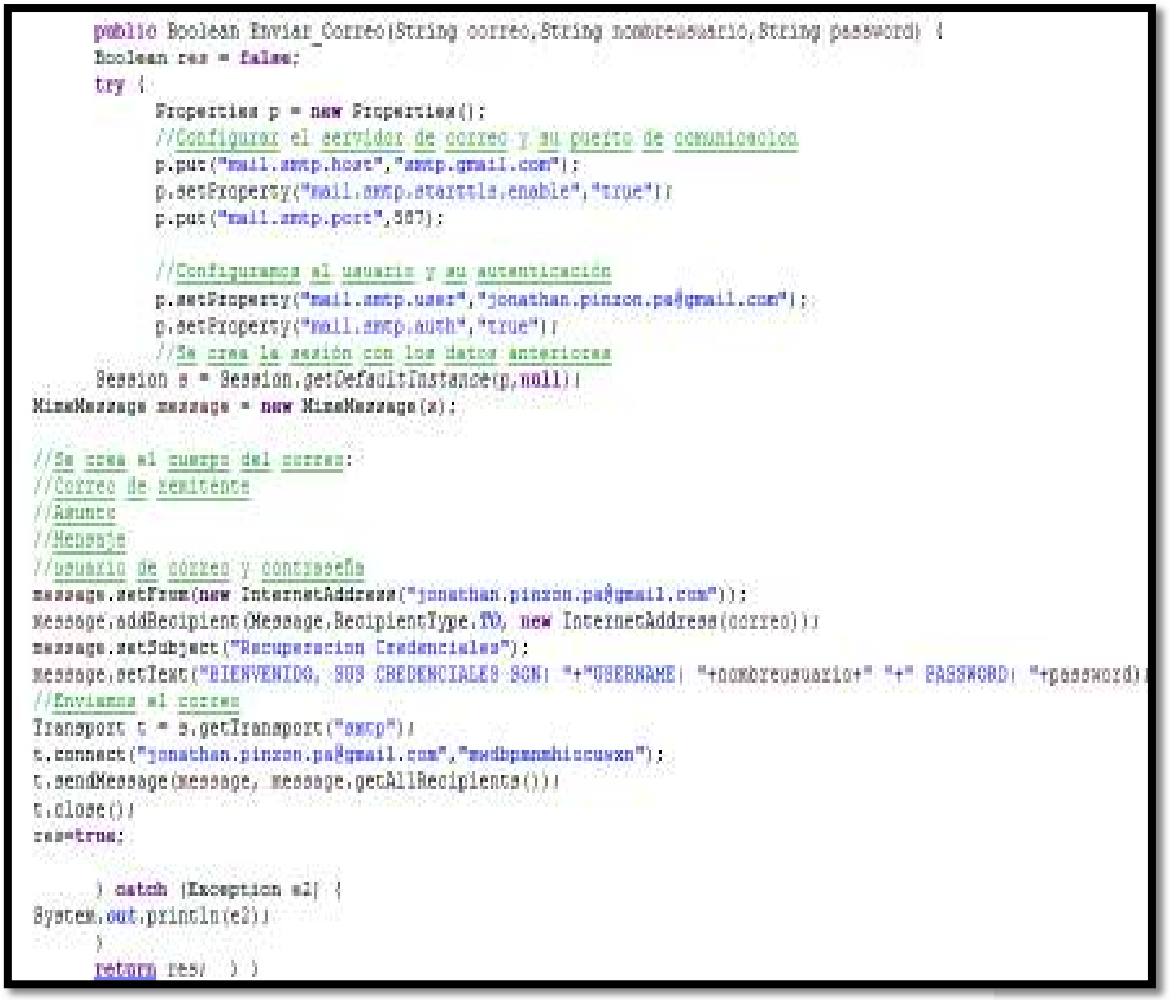
```
"SELECT  username,passwordUser FROM usuario WHERE correo = ?"
```

Código 3.14. Consulta de Credenciales

Con esta consulta se solicita el nombre de usuario (*username*) y password (*passwordUser*) del usuario que tiene el correo ingresado.

Una vez obtenidos estos valores se realiza una instancia de la clase Correo para posteriormente llamar a su método Enviar_Correo, a este método se pasa como parámetros el correo y las credenciales recuperadas desde la base de datos. De no obtener las credenciales al momento de realizar la consulta o de no poder enviar el correo se presenta un mensaje que indica que el proceso de recuperación de credenciales ha fallado.

En el Código 3.15 se muestra la codificación del método de Enviar_Correo, método que pertenece a la clase Correo.



```
public Boolean Enviar_Correo(String correo,String nombreusuario,String password) {
Boolean res = false;
try {
      Properties p = new Properties();
      //Configurar el servidor de correo y su puerto de comunicacion
      p.put("mail.smtp.host","smtp.gmail.com");
      p.setProperty("mail.smtp.starttls.enable","true");
      p.put("mail.smtp.port",587);

      //Configuramos al usuario y su autenticación
      p.setProperty("mail.smtp.user","jonathan.pinzon.pa@gmail.com");
      p.setProperty("mail.smtp.auth","true");
      //Se crea la sesión con los datos anteriores
    Session s = Session.getDefaultInstance(p,null);
MimeMessage message = new MimeMessage(s);

//Se crea el cuerpo del correo:
//Correo de remitente
//Asunto
//Mensaje
//usuario de correo y contraseña
message.setFrom(new InternetAddress("jonathan.pinzon.pa@gmail.com"));
Message.addRecipient(Message.RecipientType.TO, new InternetAddress(correo));
message.setSubject("Recuperacion Credenciales");
message.setText("BIENVENIDO, SUS CREDENCIALES SON: "+"USERNAME: "+nombreusuario+" "+" PASSWORD: "+password);
//Enviamos el correo
Transport t = s.getTransport("smtp");
t.connect("jonathan.pinzon.pa@gmail.com","medbpamhiscuwxn");
t.sendMessage(message, message.getAllRecipients());
t.close();
res=true;

    } catch (Exception e2) {
System.out.println(e2);
    }
    return res;  } }
```

Código 3.15. Codificación Método Enviar_Correo

Hay tres pasos para enviar un correo electrónico utilizando la librería *JavaMail* estos pasos se muestran en los Códigos 3.16, 3.17 y 3.18 respectivamente.

- Obtener el objeto de sesión

```
Properties p = new Properties();
            //Configurar el servidor de correo y su puerto de comunicacion
            p.put("mail.smtp.host","smtp.gmail.com");
            p.setProperty("mail.smtp.starttls.enable","true");
            p.put("mail.smtp.port",587);
//Configuramos al usuario y su autenticación
            p.setProperty("mail.smtp.user","jonathan.pinzon.pa@gmail.com");
            p.setProperty("mail.smtp.auth","true");
      //Se crea la sesión con los datos anteriores
     Session s = Session.getDefaultInstance(p,null);
```

Código 3.16. Objeto de Sesión

- Crear el cuerpo del mensaje

```
//Se crea el cuerpo del correo:
//Correo de remitente
//Asunto
//Mensaje
//usuario de correo y contraseña
message.setFrom(new InternetAddress("jonathan.pinzon.pa@gmail.com"));
message.addRecipient(Message.RecipientType.TO, new InternetAddress(correo));
message.setSubject("Recuperacion Credenciales");
message.setText("BIENVENIDO,    SUS    CREDENCIALES    SON:    "+"USERNAME:
"+nombreusuario+" "+" PASSWORD: "+password);
```

Código 3.17. Cuerpo del Mensaje

- Se envía el mensaje

```
//Enviamos el correo
Transport t = s.getTransport("smtp");
t.connect("jonathan.pinzon.pa@gmail.com","mwdbpmnmhiocuwxn");
    t.sendMessage(message, message.getAllRecipients());
```

Código 3.18. Envio de Mensaje

En el último paso cuando se realiza la conexión se observa que se pasa como parámetros el correo y contraseña. Se observa una particularidad en la contraseña ya que es de este tipo: *mwdbpmnmhiocuwxn*, debido a que es una contraseña aleatoria generada por Gmail.

Es necesario generar una clave adicional para aplicaciones externas en Gmail, con esta clave se puede acceder desde otras aplicaciones que quieran crear sesiones de este usuario.

Para generar esta clave primero se abre la cuenta de Gmail, allí se selecciona “Acceso y seguridad” y en el apartado de “Contraseñas de las aplicaciones”. Como se muestra en la Figura 3.12.

Luego se ingresa el tipo de gestor o aplicación que va a solicitar la clave. De no existir entre las opciones se puede ingresar un nombre propio como es en este caso. Como se muestra en la Figura 3.13.

El último paso es generar la clave y utilizar esta en la aplicación para iniciar sesión. Esto necesario generar esta contraseña adicional con el objetivo de no vulnerar la seguridad de la cuenta de Gmail. Como se muestra en la Figura 3.14.

Figura 3.12. Creación de Contraseña para Aplicaciones Paso 1

Figura 3.13. Creación de Contraseña para Aplicaciones Paso 2

Figura 3.14. Creación de Contraseña para Aplicaciones Paso 3

Con la codificación de las dos clases mencionadas anteriormente, se cumple con las funcionalidades del módulo 1 del Prototipo de Software. La codificación completa de las clases se encuentra adjunta en el ANEXO C.

3.2.3 Codificación Módulo Comunicación/Conexión

A continuación se muestra la codificación de las clases que conforman el módulo de Comunicación/Conexión: Establecer_Conexion, Conexión, Conectividad_Esclavo (1,2 y 3), Reset (1,2 y 3) y Diagrama_Red.

a. Codificación Clase Establecer_Conexion

Establecer_Conexion hereda de *JFrame* ya que es una interfaz gráfica a través de la cual el usuario establece la conexión con cualquiera de los tres esclavos que se encuentre visibles en la red industrial. En el alcance del prototipo se definió un cantidad máximo de 3 esclavos, sin embrago se podría realizar conexiones simultaneas con muchos más.

En esta interfaz se tiene un botón por cada esclavo para establecer la conexión y cerrarla.

A continuación en el Código 3.19 se muestra la codificación sobre las funciones que se ejecutan al presionar este botón abrir conexión.

```
 btnabrir1 = new JButton("Abrir conexion");
btnabrir1.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {

    dir_ip = txtdir1.getText();
try {
                socket_1 = new Socket (dir_ip,502);
                output_1 = socket_1.getOutputStream();
                input_1 = new BufferedInputStream(socket_1.getInputStream());
                con1=true;
                socket_11 = new Socket (dir_ip,502);
                output_11 = socket_11.getOutputStream();
                input_11 = new BufferedInputStream(socket_11.getInputStream());
                con11=true;
                socket_111 = new Socket (dir_ip,502);
                output_111 = socket_111.getOutputStream();
                input_111 = new BufferedInputStream(socket_111.getInputStream());
                con111=true;
                Conectividad_Esclavo1 conectividad1 = new Conectividad_Esclavo1();
                conectividad1.start();
                Reset1 r = new Reset1();
                r.start();
                d1=dir_ip;
         }
        catch (Exception e) {
        con1=false;
        d1="";
         }
    resultado_conectar(con1);
    label_conectividad();
    }
});
```

Código 3.19. Codificación Botón Abrir Conexión

Como se pudo observar en la figura anterior, para el proceso de comunicación se está utilizando sockets, cada socket se conforma por un número de puerto y una dirección IP, en este caso el número de puerto es 502 (número de puerto exclusivo para ModBus TCP) y la dirección IP del esclavo con el que se quiere comunicar.

Por cada esclavo se genera 3 sockets. Esto debido a que se necesita un primer socket para la funcionalidad de Lectura/Escritura, un segundo socket para implementar la funcionalidad de Diagrama de Red el cual necesita un flujo independiente ya que está constantemente realizando consultas hacia el esclavo para determinar si la conectividad está establecida o no. Y un tercer socket es utilizado para la funcionalidad de generar el histórico de registros; esta funcionalidad corre por detrás y debe ser independiente del socket utilizado para la funcionalidad de Lectura/Escritura.

Por estas razones se genera tres sockets diferentes, flujos de salida y flujos de entrada.

Además por cada socket se genera una variable del tipo *boolean* la cual actualiza el label que indica el estado de la conexión en esta interfaz gráfica.

También se crea una instancia de los Hilos Conectividad_Esclavo1 y Reset1 y se las inicializa.

En el botón Cerrar Conexión se realiza el cierre del *socket* creado al momento de establecer la conexión. Para esto instancia a la clase conexión y se llama al método cerrar conexión como se muestra en el Código 3.20.

```
Conexion cierre1=new Conexion();
cerrar1=cierre1.cerrar_conexion(socket_1);
```

Código 3.20. Codificación Cerrar Conexión

b. Codificación Hilo Conectividad_Esclavo

En el Código 3.21 se muestra la codificación del hilo Conectividad_Esclavo1. Este hilo es el encargado de verificar en cada intervalo de tiempo la conectividad con el esclavo 1. Se realiza la lectura del flujo de entrada en este caso “input_1” al perderse la conexión con el esclavo1 el valor leído de este flujo de entrada es -1, entonces se puede interpretar esto como que la conexión con el esclavo se ha perdido.

Esta es una propiedad del objeto del tipo *read(BufferedInputStream)* como se muestra en la Figura 3.15.

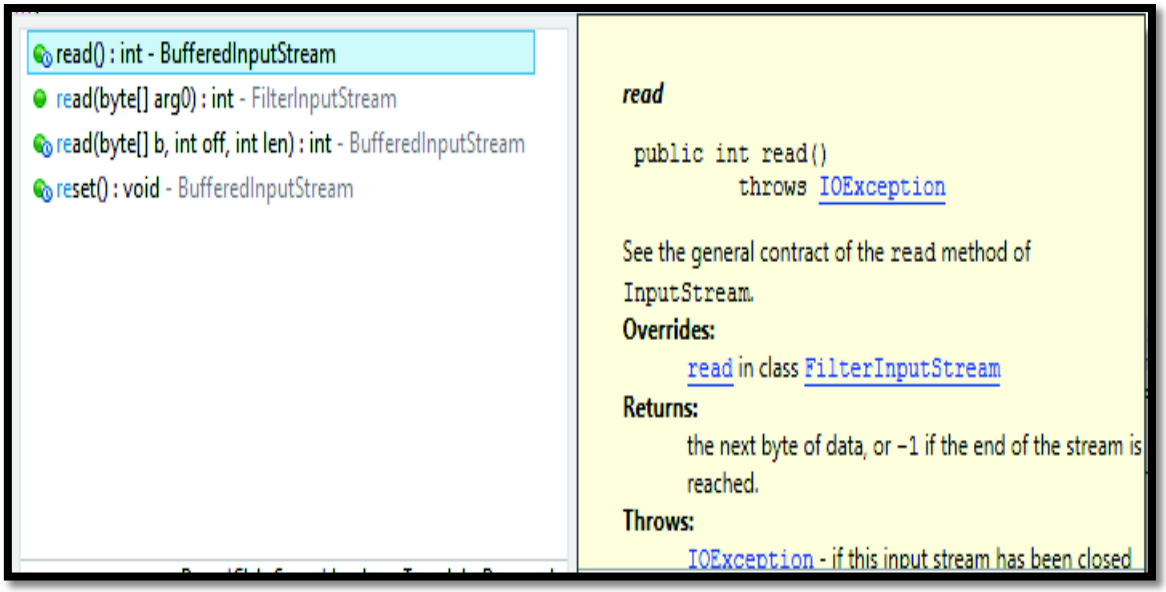


Figura 3.15. Propiedad *read(BufferedInputStream)*

Además se tiene una función de nombre "esperarXsegundos", esta función ayuda a generar un retardo para que este hilo "Conectividad_Esclavo1" se ejecute cada cierto tiempo, debido a que cuando se inicia un hilo (*start* ()) este se ejecuta constantemente.

En caso de que el valor leído en el flujo de entrada sea diferente de -1 será interpretado como que el esclavo sigue conectado y la variable booleana "estado1" seguirá con el valor true.

En la codificación de esta clase se puede observar que existe un contador el cual va registrando cuantas veces el valor leído en el flujo de entrada es -1. Esto se debe a que hay ocasiones en la cuales se tiene falsos positivos en cuanto a un proceso de desconexión de red, esto se puede deber a un retardo en la red u otros eventos que no necesariamente se deben interpretar como un evento de desconexión de un esclavo.

Si la lectura repetidas veces es igual a -1, esto será interpretado como una desconexión del esclavo, con esto la variable booleana "con11" también cambia su estado de true a false lo cual será interpretado y gestionado en la clase "Diagrama_Red".

Para esto se realiza el llamado al método resultado_conectividad() de la clase Establecer_Conexion, aquí se revisa si el estado de la conexión representado por el booleano "estado1" es true o false y de ser false muestra un mensaje de que se perdió la conexión, tal como se presenta en el Código 3.21.

```java
import java.io.BufferedInputStream;

public class Conectividad_Esclavo1 extends Thread {
    private boolean estado1=true;
    private int contador;
    // Constructor, getter & setter
    @Override
    public void run() {
        Establecer_Conexion con = new Establecer_Conexion();
        while (con.con11==true){
            this.esperarXsegundos(1);
            try {
                if (con.input_11.read()==-1 ){
                    if (contador>=10) {
                        estado1=false;
                    }
                    contador= contador+1;
                } else {
                    estado1=true;
                }
            } catch (IOException e) {
            }
            con.setCon11(estado1);
        con.resultado_conectividad(estado1);
        }
    }
    private void esperarXsegundos(int segundos) {
        try {
            Thread.sleep(segundos * 1000);
        } catch (InterruptedException ex) {
            Thread.currentThread().interrupt();
        }
    }
```

Código 3.21. Codificación Hilo Conectividad_Esclavo1.

c. Codificación Hilo Reset

El hilo “Reset1” al igual que “Conectividad_Esclavo1”, hay uno diferente para cada esclavo. La tarea de este hilo es resetear la conexión con los esclavos cada cierto tiempo, esto debido a que en la fase implementación se detectó que la conexión con los esclavos se perdía cada cierto tiempo, tanto con los esclavos virtuales como con el esclavo físico.

El hilo “Reset1” va a resetear la conexión cada determinado tiempo, es decir volverá a crear el mismo socket, flujo de entrada y flujo de salida. Con esto el tiempo de desconexión se reseteará a cero cada cierto tiempo.

Al igual que en el hilo “Conectividad_Esclavo1” se desarrolló una función que genere un retardo y así controlar cada que tiempo debe ser ejecutado este hilo.

La codificación de este hilo se muestra a continuación en el Código 3.22.

```
import java.io.BufferedInputStream;
import java.io.IOException;
import java.io.OutputStream;
import java.net.Socket;
import java.net.UnknownHostException;

public class Reset1 extends Thread {

	@Override
	public void run() {
		Establecer_Conexion con = new Establecer_Conexion();

		while (true){
			this.esperarXsegundos(480);
			Establecer_Conexion a = new Establecer_Conexion();
		try {
			a.socket 10    =new Socket (Establecer_Conexion.getDir ip(),502);
			a.output 10=a.socket 10.getOutputStream();
		 a.input 10= new BufferedInputStream(a.socket 10.getInputStream());

		 a.socket 11=a.socket 10;
		 a.output 11=a.output 10;
		 a.input 11=a.input 10;

		 a.socket 1=a.socket 10;
		 a.output 1=a.output 10;
		 a.input 1=a.input 10;

		 a.socket 111=a.socket 10;
		 a.output 111=a.output 10;
		 a.input 111=a.input 10;

		 System.out.println("ya paso transfiero el tiempo1");

		} catch (UnknownHostException e) {

		} catch (IOException e) {

		} } }

	private void esperarXsegundos(int segundos) {
		try {
			Thread.sleep(segundos * 1000);
		} catch (InterruptedException ex) {
			Thread.currentThread().interrupt();
		} } }
```

Código 3.22. Codificación Hilo Reset1

3.2.4 Codificación Módulo Lectura/Escritura

A continuación se muestra la codificación de las interfaces gráficas que conforman el módulo de Lectura/Escritura. Estos son Esclavo_UNO, Esclavo_DOS y Esclavo_TRES, además de las funciones principales las cuales son leer_coil, leer_inputstatus, leer_input, leer_holding, escribir_coil y escribir_holding.

Las tres interfaces principales que conforman este módulo son idénticas con la única diferencia que cada interfaz gráfica realiza las funcionalidades para un esclavo diferente.

Por esta razón a continuación solo se explicará la codificación de la interfaz Esclavo_UNO.

a. Codificación Interfaz gráfica Esclavo_UNO

Esclavo_UNO hereda de la clase *JFrame* ya que es una interfaz gráfica con la cual el usuario utiliza las funcionalidades de Lectura y Escritura de los registros que contiene el esclavo.

Los paquetes adicionales que se usa en esta clase se presentan en el Código 3.23.

```
import java.io.BufferedInputStream;
import java.io.OutputStream;
import java.net.Socket;
```

Código 3.23. Paquetes Adicionales Clase Esclavo_UNO

Los dos primeros son de la librería java.io, los cuales son necesarios para crear los flujos tanto de entrada *BufferedInputStream* y de salida *OutputStream*, también se tiene el paquete *java.net.Socket* el cual es necesario para crear el socket y en conjunto con los flujos para crear el canal de comunicación.

Anteriormente en la fase de diseño se definió la estructura de la interfaz gráfica la cual se presenta en la Figura 3.16.

Figura 3.16. Diseño Interfaz Gráfica Esclavo_UNO

Como se observa en la figura, existe un botón para cada funcionalidad, cuatro para la lectura de cada uno de los tipos de registros y dos para escribir en los dos tipos de registros que permiten su escritura. Además de cajas de texto para seleccionar la dirección y cantidad de registros.

A continuación se detalla la codificación de cada una de las funciones del ModBus/TCP utilizadas para leer y escribir cada uno de los registros.

b. Codificación Función Leer Coil y Leer Input Status

La función Leer Coil es utilizada para la lectura de registros del tipo “Coil”, es decir salidas binarias (1/0) sobre la cuales se puede realizar procesos de lectura y escritura. El número asignado para esta función es el 1.

En el Código 3.24 se muestra la parte de estructuración de la trama en la codificación de la función Leer Coil.

```
  public void leer_coil(Establecer_Conexion conexion1) {
    // Rutina para funcion leer de MODBUS. Codigo de funcion 03.

    int unidad=0;   // Identificador de unidad (#n esclavo - PC)
    int direccion = Integer.parseInt(txtdirreg1.getText())-1;
    // Direccion de memoria
        int cantidad = Integer.parseInt(txtcantreg1.getText());
    // Cantidad de registros a leer

int registros[]= new int [256];// Buffer para colocar valores leidos
int c, i,j;

try {
    //  Construir la trama Modbus/TCP  leer registros
    for ( i=0; i<5; i++ )
    buffer[i] = 0;
    buffer[6] = (byte) unidad;
    buffer[7] = 1; //3: leer; 16: escribir
    buffer[8] = (byte) (direccion >> 8);
    // MSB desplaza 8 bits a la    derecha
    buffer[9] = (byte) (direccion & 0xFF); // LSB resto
    buffer[10] = 0;
    buffer[11] = (byte) cantidad;
    buffer[5] = 6; // n de bytes que siguen
```

Código 3.24. Codificación Trama Leer *Coil*

Como se mostró anteriormente en la Figura 1.8, la cabecera MBAP está compuesta por 7 bytes que son los siguientes:

- buffer[0] buffer[1] : representan el valor alto (H) y bajo (L) del número de trama: 0
- buffer[2] buffer[3] : representan el valor alto (H) y bajo (L) del identificador de protocolo: 0
- buffer[4] buffer[5] : representan el valor alto (H) y bajo (L) de bytes que continúan: [0] [6]

- buffer[6] : representan el número de esclavo: [0]
- buffer[7] : representan el número de función de ModBus/TCP (Leer Coil): [1]
- buffer[8] buffer[9] : representan el valor alto y bajo de la dirección del registro: [dirección]
- buffer[10] buffer[11] : representan el valor alto y bajo de la cantidad de registros a leer: [cantidad]

En el siguiente proceso se prepara los flujos tanto de salida, para enviar la trama, como el flujo de entrada para leer la respuesta del esclavo. Esto se observa en el Código 3.25.

```
 //  Enviar la solicitud al servidor
output=conexion1.getOutput 1();
output.write(buffer, 0, buffer[5] + 6);
 //  Esperar y leer la  respuesta
  c= conexion1.getInput 1().read(buffer, 0, buffer.length);
```

Código 3.25. Generación Flujo Entrada y Salida

Una vez realizado el envío de solicitud al esclavo y recibido una respuesta, se procede a interpretarla. Se interpreta el valor que el esclavo está devolviendo.

Este proceso se puede observar en el Código 3.26, en este caso se solicita el estado de 16 salidas digitales las cuales esta numerada del 0 hasta el 15, por lo tanto el esclavo devolverá un entero de 2 bytes uno la parte alta y otro la parte baja. Tal como se indicó anteriormente la respuesta del esclavo en respuesta a la función 1 sería la presentada en la Figura 3.17.

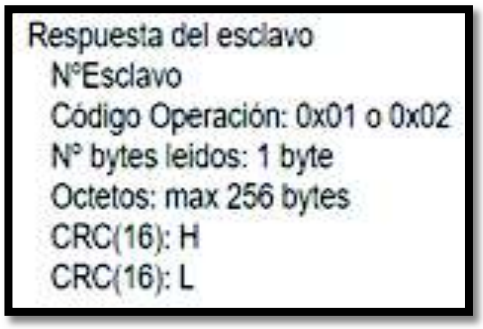


Figura 3.17. Formato Respuesta de Esclavo

Por esta razón en el *buffer* que devuelve el esclavo a partir del espacio del *buffer* [9] y *buffer* [10] se almacena el valor de los registros consultados. En este caso solo va a devolver un byte alto y un byte bajo es por eso que el lazo *for* correrá una sola vez.

Una vez obtenido el registro se almacena en la variable registros [i] y se tiene un número binario de 2 bytes listo para ser interpretado.

Antes de interpretar el resultado se tiene un condicional el cual va a ayudar a completar el número binario de dos bytes, esto debido a que el esclavo devuelve un número con lo bits más significativos.

Por ejemplo si las salidas binarias son las siguientes:

REGISTRO: salida1 salida2 salida3 salida4 salida5 salida6 salida7 salida8
ESTADO: 0 0 1 1 1 1 0 1
El esclavo debería devolver el siguiente número: 00111101
Sin embargo solo devuelve el número más significativo es decir: 111101

Por tal motivo dentro de este condicional se analiza si el valor devuelto contiene los 16 bits, en este caso 16 caracteres ya que se convirtió el entero a *String*, y de no tener los 16 caracteres estos serán rellenados con ceros. Luego de esto se separa el resultado en dos cadenas y se asigna un valor a su correspondiente *RadioButton* de la manera como se muestra en la Figura 3.18.

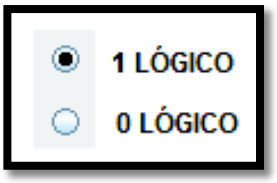


Figura 3.18. Estados Binarios de I/O Digitales

Si el valor de estado es 1, entonces el *RadioButton* se mostrará seleccionado, caso contrario al ser el estado cero el *RadiButton* se mostrará en estado normal (no seleccionado).

De esta manera se realiza el proceso de lectura de registros del tipo *Coil*. Para los otros dos esclavos se ocupa la misma función, y su codificación es exactamente igual, pero utiliza un flujo de comunicación (*socket, OutStream, BufferedInputStream*) distinto.

El tipo de registros Input Status es un tipo de variable binario igual al tipo de registro *Coil*, con la diferencia que son registros de entrada. Por este motivo sobre ellos solo se puede realizar la funcionalidad de lectura y no de escritura.

```
if ( buffer[7] == 1) {
    for (i=0;i<1;i++) {
         registros[i] = (((int) buffer[9+2*i]) << 8) & 0xFF00 |
                 ((int) buffer[10+2*i] & 0xFF);
 String binario = Integer.toBinaryString(registros[i]);

if (binario.length()<16){
  int res=16-binario.length();
  for(j=0;j<res;j++){
      binario="0"+binario;
  }
 }
 String binario1=binario.substring(0,8);
 System.out.println(binario1);
 String binario2=binario.substring(8,16);
 System.out.println(binario2);

if(binario1.substring(0,1).equals("1")){
          s8.setSelected(true);

       } else if(binario1.substring(0,1).equals("0")){

         s8.setSelected(false);
       }

if(binario1.substring(1,2).equals("1")){
  s7.setSelected(true);
```

Código 3.26. Codificación Función Leer *Coil*

La función Leer Input Status es idéntica a la función Leer Coil explicada anteriormente, con la única diferencia que cambia el valor de la función del ModBus/TCP, el valor de función para leer coil era 1 en cambio el valor de función para leer Input Status es 2. Tal como se muestra a continuación en el Código 3.27.

```
buffer[i] = 0;
buffer[6] = (byte) unidad;
buffer[7] = 2; //Función de ModBus Leer Input Status
buffer[8] = (byte) (direccion >> 8); // MSB desplaza 8 bits
//a la derecha
buffer[9] = (byte) (direccion & 0xFF); // LSB resto
buffer[10] = 0;
buffer[11] = (byte) cantidad;
buffer[5] = 6; // n de bytes que siguen
```

Código 3.27. Codificación Trama Leer *Input Status*

El proceso para recibir e interpretar el resultado de esta función es exactamente el mismo que la función anterior. A partir de un registro determinado se leen 16 entradas digitales y estas se reflejan en un *RadioButton*. Es por eso que para el proceso de lectura de ambas funciones se tiene dos filas idénticas de *RadioButtons* como se muestra a continuación en la Figura 3.19.

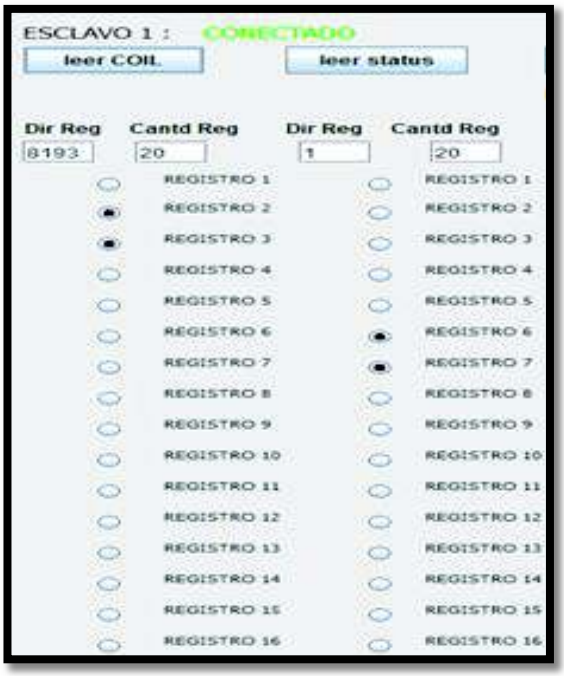


Figura 3.19. Resultado Funciones Leer *Coil* y Leer *Input Status*

c. Codificación Función Holding Registers e Input Registers

En el Código 3.28 se muestra la codificación del método Leer Holding, con esta función se lee un registro de salida del tipo palabra es decir un número de hasta 16 bits (2 bytes), el número a leer o escribir en esta función puede tomar un valor desde 0 hasta 65535. Su codificación es similar a la de las funciones explicadas anteriormente pero con algunas diferencias puntuales las cuales serán explicadas a continuación.

El tipo de registros que se gestiona con esta función son del tipo palabra por tal motivo se realiza una lista con los valores leídos, el número de función 3 del ModBus/TCP se utiliza para leer múltiples registros a la vez, por este motivo al principio de la función se elabora una lista y se añade un *JScrollPane* para almacenar y posteriormente mostrar los valores obtenidos al momento de realizar la solicitud al esclavo.

Luego se elabora la trama de la misma manera que en las funciones anteriores con la diferencia del número de función ModBus/TCP que en este caso es 3 (leer múltiples *holding registers*).

```
   public void leer_holding(Establecer_Conexion conexion4) {
         // leer
         modelo.removeAllElements();
         lista = new JList();
         lista.setBounds(524, 60, 160, 800);
         contentPane.add(lista);
         lista.setModel(modelo);
   JScrollPane scrollPaneusuario = new JScrollPane(lista);
   scrollPaneusuario.setBounds(524, 90, 200, 500);
   getContentPane().add(scrollPaneusuario);
   // Rutina para funcion leer de MODBUS. Codigo de funcion 03.
    int unidad=0; // Identificador de unidad (#n esclavo - PC)
    int direccion = Integer.parseInt(txtdir4.getText());
   // Direccion de memoria
         int cantidad = Integer.parseInt(txtcantidad4.getText());
  // Cantidad de registros a leer

int registros[] = new int [126]; // Buffer para colocar valores leidos
int c, i;
try {
    //  Construir la trama Modbus/TCP  leer registros
    for ( i=0; i<5; i++ )
    buffer[i] = 0;
    buffer[6] = (byte) unidad;
    buffer[7] = 3; //3: Función de ModBus Leer Holding Registers
    buffer[8] = (byte) (direccion >> 8);
    // MSB desplaza 8 bits a la derecha
    buffer[9] = (byte) (direccion & 0xFF); // LSB resto
    buffer[10] = 0;
    buffer[11] = (byte) cantidad;
    buffer[5] = 6; // n de bytes que siguen
```

Código 3.28. Codificación Trama Función *Leer Holding*

En la segunda parte como se muestra en el Código 3.29 se genera el flujo de salida sobre el cual se escribe la solicitud al esclavo y el flujo de entrada sobre el cual se recibe la respuesta desde el esclavo.

Otra diferencia entre esta función y las dos formas anteriores de es como recibir la respuesta del esclavo e interpretarla. Por lo general el esclavo devuelve un arreglo con los valores de los registros solicitados. Por esta razón el lazo que almacena los valores en la lista que se creó al principio de la función va a correr una cantidad de veces igual al número de registros de los cuales se quiere leer su valor.

Una vez terminado el lazo se tiene los valores de los registros solicitados como se muestra en la Figura 3.18.

```
   //  Enviar la solicitud al servidor
  output=conexion4.getOutput_1();
  output.write(buffer, 0, buffer[5] + 6);
   //  Esperar y leer la  respuesta
     c= conexion4.getInput_1().read(buffer, 0, buffer.length);
   //  Verificar la respuesta y extraer los valores leidos
   if (c == (9+2*cantidad) && buffer[7] == 3) {
         for (i=0;i<cantidad;i++) {
           registros[i] = (((int) buffer[9+2*i]) << 8) & 0xFF00 |
                          ((int) buffer[10+2*i] & 0xFF);
 modelo.addElement("registro # "+i+": valor: "+(registros[i]) );
         }
     }
     else {
          JOptionPane.showMessageDialog(null,"RANGO ERRONEO REGISTROS A
    LEER..!!!");
    registros[0] = 0; // Hacemos que no coincida el primer elemento
     }
 }
catch (Exception e1) {
     JOptionPane.showMessageDialog(null,"LECTURA INCORRECTA...!!");
     registros[0] = 0; // Hacemos que no coincida el primer elemento
 }

    }
```

Código 3.29. Codificación Respuesta *Leer Holding*

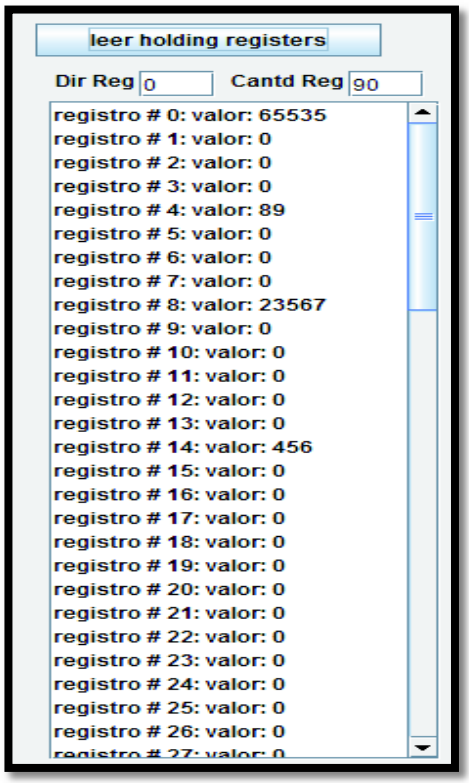


Figura 3.18. Resultado Función Leer *Holding Registers*

La función Leer *Input Registers* lee registros de entrada del tipo palabra, al ser del tipo input solo se puede realizar la funcionalidad de lectura y no de escritura. Su codificación es idéntica a la codificación de la función Leer *Holding Registers*, donde la única diferencia está en el valor de la función del ModBus/TCP que en este caso es 4. La estructura de la trama se presente a continuación en el Código 3.30.

```
buffer[i] = 0;
buffer[6] = (byte) unidad;
buffer[7] = 4; //Función de ModBus Leer Input Registers
buffer[8] = (byte) (direccion >> 8); // MSB desplaza 8 bits
//a la derecha
buffer[9] = (byte) (direccion & 0xFF); // LSB resto
buffer[10] = 0;
buffer[11] = (byte) cantidad;
buffer[5] = 6; // n de bytes que siguen
```

Código 3.30. Codificación Trama Función Leer *Input Registers*

d. Codificación Función Escribir Holding Registers y Escribir Coils

A continuación se muestran las dos funciones utilizadas en el prototipo para realizar escritura sobre los registros *Holding Registers* y *Coils,* únicamente sobre estos dos registros de los 4 anteriormente mencionados, ya que solo los registros de salida pueden ser de lectura y escritura mientras que los registros de entrada solo pueden ser de lectura.

La codificación de cada una de estas funciones se detalla a continuación. Ambas funciones tienen algunas diferencias.

La función leer holding debería tener el formato que se muestra en la Figura 3.20.

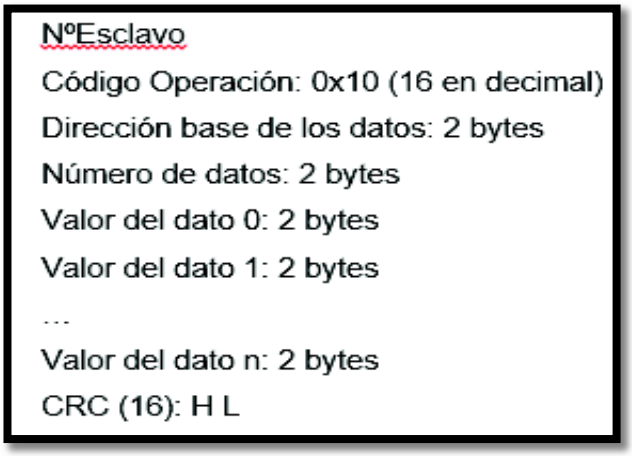


Figura 3.20. Formato Mensaje Función *Leer Holding*

Como se indicó anteriormente se debe primero crear la trama a enviar con su respectiva cabecera MBPA. Esto se muestra en el Código 3.31.

```
public void escribir_holding(Establecer_Conexion conexion6) {
// Rutina para funcion escribir de MODBUS. Codigo de funcin 16.
        int unidad=0;  // Identificador de unidad (n esclavo - PC)
        int direccion=Integer.parseInt(txt_inicio_direccion.getText());
// Direccion de memoria
        int cantidad=Integer.parseInt(txtcantidad.getText());
// Cantidad de registros a escribir
        int registros[] = new int [256]; // Buffer valores a escribir

    int c, i;
    boolean resultado=false;

    try {
        //  Construir la trama Modbus/TCP (leer registros)
        for ( i=0; i<5; i++ )
            buffer[i] = 0;
        buffer[6] = (byte) unidad;
        buffer[7] = 16; //3: leer; 16: escribir
        buffer[8] = (byte) (direccion >> 8);
   // desplaza 8 bits a la derecha
        buffer[9] = (byte) (direccion & 0xFF); // resto
        buffer[10] = 0;
        buffer[11] = (byte) cantidad;
        buffer[12] = (byte) (cantidad * 2);

        for ( i=0; i<cantidad; i++ ) {
            buffer[13 + 2*i] = (byte) (matriz[i][0] >> 8);
            buffer[13 + 2*i + 1] = (byte) (matriz[i][0] & 0xFF);
        }
        buffer[5] = (byte) (7 + cantidad*2); // n de bytes que siguen
```

Código 3.31. Codificación Función Escribir *Holding Registers*

Para escribir los arreglos de palabras (datos) se pasa como parámetro una matriz la cual se crea previamente al proceso de escritura. Para crear esta matriz primero se debe ingresar la cantidad de registros y la dirección inicial del registro a partir de la cual se va a empezar a escribir. Una vez ingresados estos dos valores se activará el botón para guardarlos dentro de la matriz y solo cuando termine de ingresar todos los valores se activará el botón escribir *holding registers* en el orden que se muestra en la Figura 3.21.

Figura 3.21. Creación Matriz para Función Escribir *Holding*

La matriz creada se pasa como parámetro a la función Escribir Holding para tomar cada uno de sus valores y escribirlos en los registros, de estos valores se guarda la parte alta (H) y baja (L) en cada dos bytes a partir del buffer [13]. El proceso para cargar los valores a escribir está dentro de un lazo el cual corre la cantidad de registros que se vaya a modificar su valor.

Luego se tiene la creación del flujo de salida, para enviar la trama, y el flujo de entrada para recibir la respuesta del esclavo. Por último se verifica la respuesta del esclavo y se muestra un mensaje sobre la escritura correcta o incorrecta de los datos. Esta codificación se la puede observar en el Código 3.32.

```
//  Enviar la solicitud al servidor
    output=conexion6.getOutput_1();
 output.write(buffer, 0, buffer[5] + 6);
//  Esperar y leer la  respuesta
//       Establecer_Conexion co = new Establecer_Conexion();

   c= conexion6.getInput_1().read(buffer, 0, buffer.length);
//    c = input.read(buffer, 0, buffer.length);
    if (c == 12 && buffer[7] == 16) {
    JOptionPane.showMessageDialog(null,"DATO ESCRITO CORRECTAMENTE");
        resultado=true;
    }
    else {
        resultado=false;
    }
}
catch (Exception e) {
    JOptionPane.showMessageDialog(null,"ESCRITURA INCORRECTA...!!");
    resultado=false;
}
  }
```

Código 3.32. Codificación Resultado Escribir *Holding Registers*

La codificación de la función Escribir *Coil* se detalla a continuación en el Código 3.31, aquí se observa que primero se debe determina el valor a escribir en el *coil* (registro binario de salida) este valor puede ser únicamente 1 o 0, por tal motivo se dispone en la interfaz gráfica con dos *RadioButton* para indicar cuál de los dos valores se va a escribir.

Luego se estructura la trama como en las funciones anteriores. Se crean los flujos de entrada y salida y se muestra un mensaje de si la escritura del *Coil* se realizó con éxito o no.

```
public void escribir_coil(Establecer_Conexion conexion5) {
          coil= Integer.parseInt(txtcoil.getText())-1;
          if(rbtn_on.isSelected()==true){
                valor_coil=0xFF;
          }else if (rbtn_off.isSelected()==true){
                valor_coil=0x00;
          }

    // Rutina para funcion escribir de MODBUS. Codigo de funcin 16.
      int unidad=0;      // Identificador de unidad (n esclavo - PC)
  int direccion=coil;  // Direccion de memoria
  int cantidad=1;
  int c, i;
  boolean resultado=false;
  try {
   //  Construir la trama Modbus/TCP (leer registros)
   for ( i=0; i<5; i++ )
       buffer[i] = 0;
   buffer[6] = (byte) unidad;
   buffer[7] = 5; //3: leer; 16: escribir
   buffer[8] = (byte) (direccion >> 8);
   // desplaza 8 bits a la   derecha
   buffer[9] = (byte) (direccion & 0xFF); // resto
   buffer[10] = (byte)valor_coil ;
   buffer[11] = (byte) 0x00;
   buffer[5] = (byte) (6); // n de bytes que siguen

   //    Establecer_Conexion col = new Establecer_Conexion();
   //  Enviar la solicitud al servidor
   output=conexion5.getOutput_1();
   output.write(buffer, 0, buffer[5] + 6);
   //  Esperar y leer la  respuesta
   //  Establecer_Conexion co = new Establecer_Conexion();

   c= conexion5.getInput_1().read(buffer, 0, buffer.length);
   //    c = input.read(buffer, 0, buffer.length);
   if (c == 12 && buffer[7] == 5) {
  JOptionPane.showMessageDialog(null,"DATO ESCRITO CORRECTAMENTE");
       resultado=true;
   }
   else {
       JOptionPane.showMessageDialog(null,"no escribio");
       resultado=false;
   }
}
catch (Exception e) {
    JOptionPane.showMessageDialog(null,"ESCRITURA INCORRECTA...!!");
    resultado=false;
}
  }
```

Código 3.33. Codificación Función Leer *Coil*

3.2.5 Codificación Módulo Generar Registros

A continuación se muestra la codificación de las clases que conforman el módulo Generar Registros: Generar_Registros, Registro (1,2 y 3), y Guardar_Historial.

a. Codificación Clase Generar_Registros

Esta clase hereda de *JFrame* ya que es la interfaz gráfica donde el usuario utiliza la funcionalidad de generar registros de determinados valores y guardarlos en la base de datos. Esta clase permite al usuario generar un histórico de tres variables específicas, para esto previo a iniciar la generación de los registros se debe proporcionar información adicional la cual será almacenada en la base.

Para obtener esta información se cuenta con cajas de texto para ingresar los parámetros como: nombre de las variables, nombres de historial, tiempo de muestreo, etc. Esto se puede observar en la Figura 3.22.

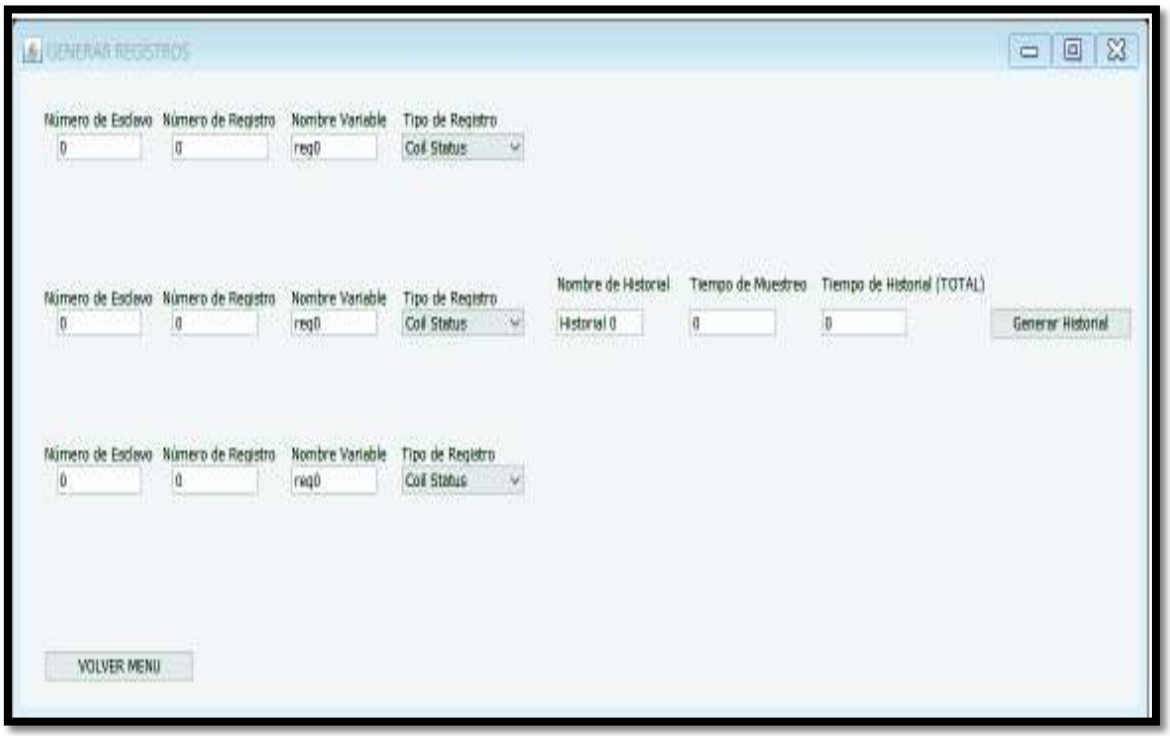


Figura 3.22. Interfaz Gráfica Generar Registros

Antes de guardar la información se debe leer los valores de los registros a guardar, elegir el esclavo, número y tipo de registro a leer. Para esto se tiene un *ComboBox* en los cual se elige cual es el tipo de registro a leer.

En la figura 3.23 se muestra un ejemplo de cómo seleccionar el primer registro del tipo holding *registers* que pertenezca al esclavo uno.

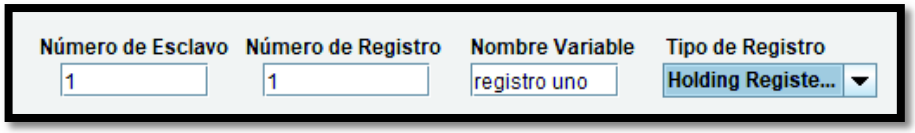


Figura 3.23. Ejemplo de Selección de Registro

Al seleccionar el tipo de registro, automáticamente se empieza a leer la variable para actualizar su valor y posteriormente guardarlo en la base de datos.

Todas estas funciones están codificadas dentro del *ComboBox*, como ejemplo en el Código 3.34, se muestra el proceso de elegir el esclavo del cual se va a leer el valor.8.

```
JComboBox cmbregistros = new JComboBox();
cmbregistros.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent arg0) {

            esclavo = txtnumeroesclavo.getText();
            registro = txtregistro.getText();

            if (esclavo.equals("1") ){
      Establecer_Conexion co1 = new Establecer_Conexion();
                  output=co1.getOutput_111();
                  input= co1.getInput_111();
                  output10=co1.getOutput_11();
                  input10= co1.getInput_11();

numesclavo =Integer.parseInt(txtnumeroesclavo.getText());
registrog= Integer.parseInt(txtregistro.getText());
nombreregistro= txtnombrevar.getText();

            }
            if (esclavo.equals("2")){
      Establecer_Conexion co2 = new Establecer_Conexion();
                  output=co2.getOutput_222();
                  input= co2.getInput_222();
numesclavo =Integer.parseInt(txtnumeroesclavo.getText());
registrog= Integer.parseInt(txtregistro.getText());
nombreregistro= txtnombrevar.getText();

            }
            if (esclavo.equals("3")){
      Establecer_Conexion co3 = new Establecer_Conexion();
                  output=co3.getOutput_333();
                  input= co3.getInput_333();
numesclavo =Integer.parseInt(txtnumeroesclavo.getText());
registrog= Integer.parseInt(txtregistro.getText());
nombreregistro= txtnombrevar.getText();
            }
```

Código 3.34. Codificación *ComboBox* Elegir Esclavo

Como se observa en el código anterior primero se debe tomar el valor ingresado en el cuadro de texto de número de esclavo para determinar de qué esclavo se trata, e instanciar al flujo correspondiente de lo que fueron creados al momento de establecer la conexión.

Una vez creados los flujos se identifica el tipo de registro que se va a leer y a qué número de registro pertenecen.

Con esto se llama a cada una de las funciones de lectura las cuales pueden ser leer_coil, leer_input_status, leer_input y leer_holding. También se inicia el hilo respectivo dependiendo de que esclavo se trate, en este caso como es el esclavo 1 se inicia el hilo Registro1. Esto se puede observar en el Código 3.35.

```
if(cmbregistros.getSelectedItem().toString()== "Coil Status"){
    leer_coil(output, input, registro);
    generar=1;
     registrol1 = new Registrol();
    registrol1.start();
}
else if(cmbregistros.getSelectedItem().toString()== "Input Status"){
    leer_input_status(output, input, registro);
    generar=2;
     registrol1 = new Registrol();
    registrol1.start();
}
else if(cmbregistros.getSelectedItem().toString()== "Input Registers"){
    leer_input(output, input, registro);
    generar=3;
     registrol1 = new Registrol();
    registrol1.start();
}
else if (cmbregistros.getSelectedItem().toString()== "Holding Registers"){
    leer_holding(output, input, registro);
    generar=4;
 registrol1 = new Registrol();
    registrol1.start();
}
```

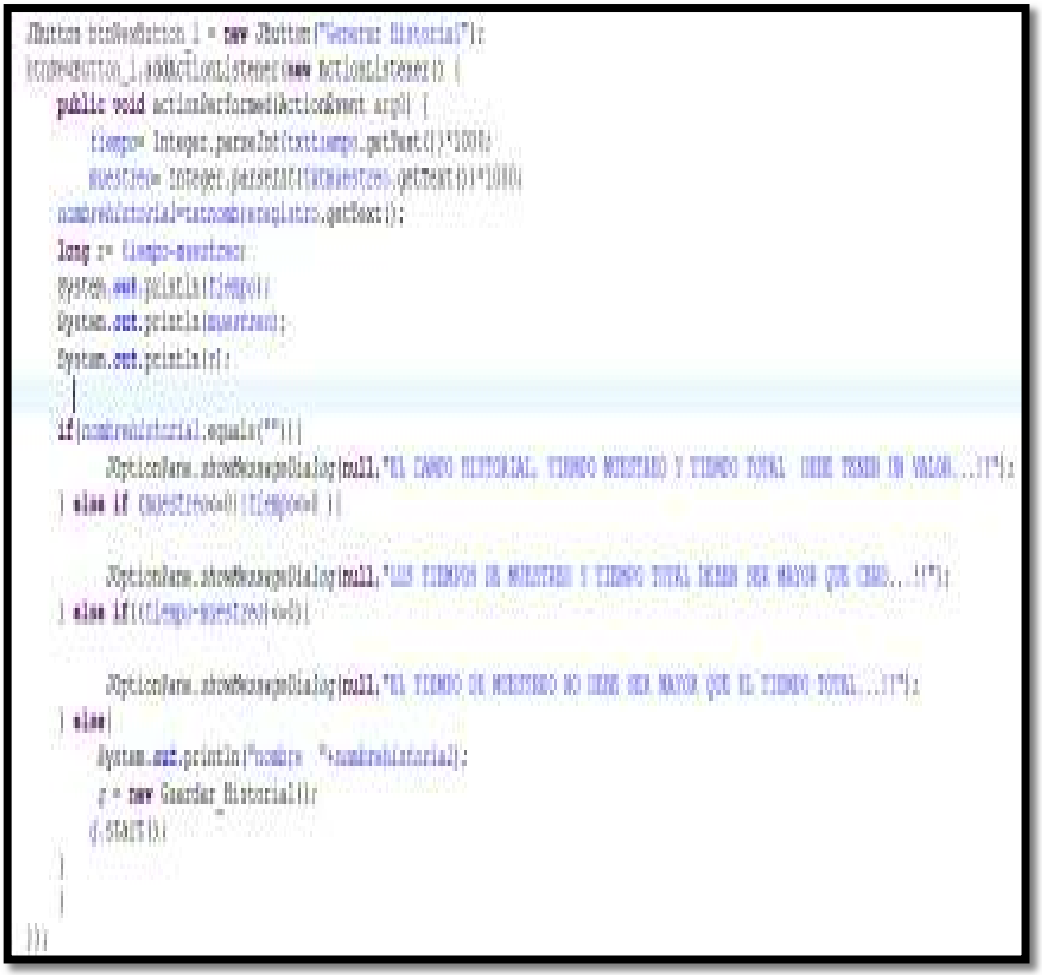
Código 3.35. Codificación *ComboBox* Parte 2

El hilo Registro1 se encarga de leer constantemente el registro seleccionado. La tarea principal de este hilo es actualizar constantemente el valor de lectura, debido a que cuando empiece a correr el hilo para guardar los registros en la base de datos se tomará el valor actualizado. Su codificación se muestra en el Código 3.36.

En la clase principal se detalla la codificación del botón Generar Historial que es el encargado de lanzar el hilo que estará constantemente guardando los valores en la base de datos. Adicional a esto se tomará los valores ingresados en las cajas de texto para ser transformados al tipo de datos *long* y crear el contador que definirá si se continua o no guardando los valores en la base de datos.

Código 3.36. Codificación Hilo Registro1

Para realizar esta función existen algunas condiciones, por ejemplo los valores de tiempos tanto de muestreo como total deben ser mayores que 0 y también que el tiempo de muestreo no puede ser mayor al tiempo total del historial. Todo esto se puede observar en el Código 3.37.



Código 3.37. Codificación Botón Generar Registros

3.2.6 Codificación Módulo HMI

A continuación se muestra la codificación de las clases que conforman el módulo de GUI. La clase que conforma este módulo es HMI. Esta clase hereda de *JFrame* ya que es la interfaz gráfica con la cual el usuario puede utilizar la funcionalidad de HMI específica. Aquí se visualiza constantemente el estado de una variable la cual está asociada al valor de temperatura, y otras dos variables las cuales están asociadas al nivel de un tanque de agua y el caudal de entrada.

Se está obteniendo valores desde registros específicos de entrada y en base a estos se realiza un tipo de control básico.

Las dos principales funciones que se desarrollan en esta interfaz son:

- Control de temperatura
- Control del nivel del tanque

a. Codificación Control de Temperatura

Como se muestra en la Figura 3.24, el control de temperatura se basa en estar permanentemente tomando valores de temperatura y si sobrepasa un umbral, el cual será configurado en la caja de texto, mostrar una alarma que indique que se produjo una alarma de sobre-temperatura.

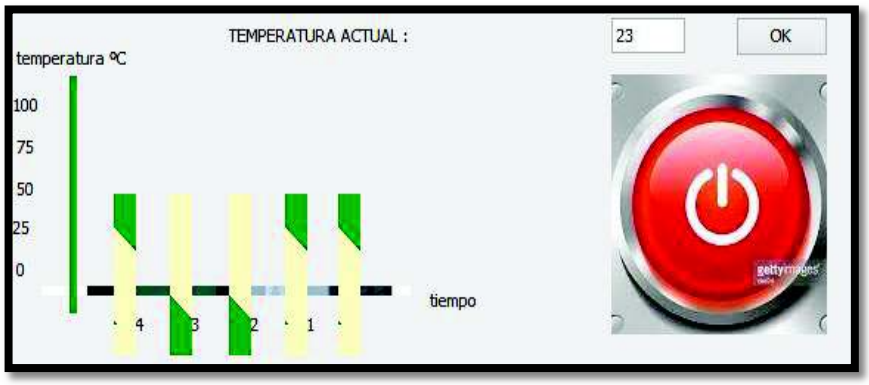


Figura 3.24. Interfaz HMI Parte Control de Temperatura

Para implementar estas funciones se utiliza *timers*, los *timers* son hilos predefinidos que se los pueden utilizar en cualquier otra clase.

En este caso el timer3 que es una instancia de la clase *Timer*, es el encargado de leer la variable actual de temperatura y representarla con el tamaño adecuado de la barra en el momento tn y mover hacia las barras de la izquierda los valores de tn-1, tn-2 y así sucesivamente. Esta codificación se muestra en el Código 3.38.

```
Timer timer3 = new Timer (5000, new ActionListener ()
{
    public void actionPerformed(ActionEvent e)
    {
        int a =Generar_Registros.getValorleido();
        String b= Integer.toString(a);
         label.setText(b+" °C");
         int res=100-a;
     if (contador==0){
         boton5.setBounds(230, 65+res, 15, 100-res);
         c1=65+res;
         c2=100-res;
         boton4.setBounds(194, c9, 15, c10);
         boton3.setBounds(156, c7, 15, c8);
         boton2.setBounds(116, c5, 15, c6);
         boton1.setBounds(78, c3, 15, c4);
         boton_alarma(a);
         } else if (contador==1){
             boton5.setBounds(230, 65+res, 15, 100-res);
             boton4.setBounds(194, c1, 15, c2);
             boton3.setBounds(156, c9, 15, c10);
             boton2.setBounds(116, c7, 15, c8);
             boton1.setBounds(78, c5, 15, c6);
             c3=65+res;
             c4=100-res;
             boton_alarma(a);
```

Código 3.38. Codificación Timer3 Control Temperatura

El Timer3 guarda el valor anterior de temperatura para en el siguiente lazo pasar el valor leído a la siguiente barra. Este *timer* se ejecutara cada 5 segundos.

En cada lazo se revisa si se ha sobrepasado el umbral para esto se llama a la función boton_alarma y se le pasa como parámetros el valor leído de temperatura.

La codificación de la función boton_alarma se presenta en el Código 3.39.

```
public void boton_alarma(int a1) {

     if (umbral<=a1){
         botonumbral.setVisible(true);
         java.util.Date d = new java.util.Date();
            java.sql.Date date2 = new java.sql.Date(d.getTime());
            Date d1=new Date();
            GregorianCalendar gc= new GregorianCalendar();
            gc.setTime(d1);
            SimpleDateFormat ff=new SimpleDateFormat("hh:mm:ss");
            String s=ff.format(d);
            lblalerta.setText("ALARMA DE TEMPERATURA: "+date2+" "+s);
        timer3.stop();
     }
         repaint();
}
```

Código 3.39. Codificación Boton Alarma

La tarea de esta función es determinar en cada lazo si la temperatura sobrepasó o no el umbral, de hacerlo entonces se detiene el hilo que lee la temperatura, activará el botón de alarma de sobre temperatura, además obtendrá la fecha y hora en que se dio ese evento.

b. Codificación de Control Nivel de Tanque

Como se observa en la Figura 3.25 la tarea de esta función es controlar el caudal de entrada para que el nivel de agua en el tanque no se desborde, para esto constantemente lee el valor del nivel del tanque y de acuerdo a esa lectura toma la decisión de cerrar o abrir tanto la válvula de llenado como la válvula de caudal de salida.

El valor para abrir la válvula del caudal de salida es cuando el nivel del tanque sea mayor a los 250 litros, y la válvula de caudal de entrada se cierra cuando el nivel del tanque se superior a los 800 litros.

La alarma de que el nivel del tanque esta rebosado se mostrará cuando el nivel del tanque haya sobre pasado su capacidad es decir los 1000 litros.

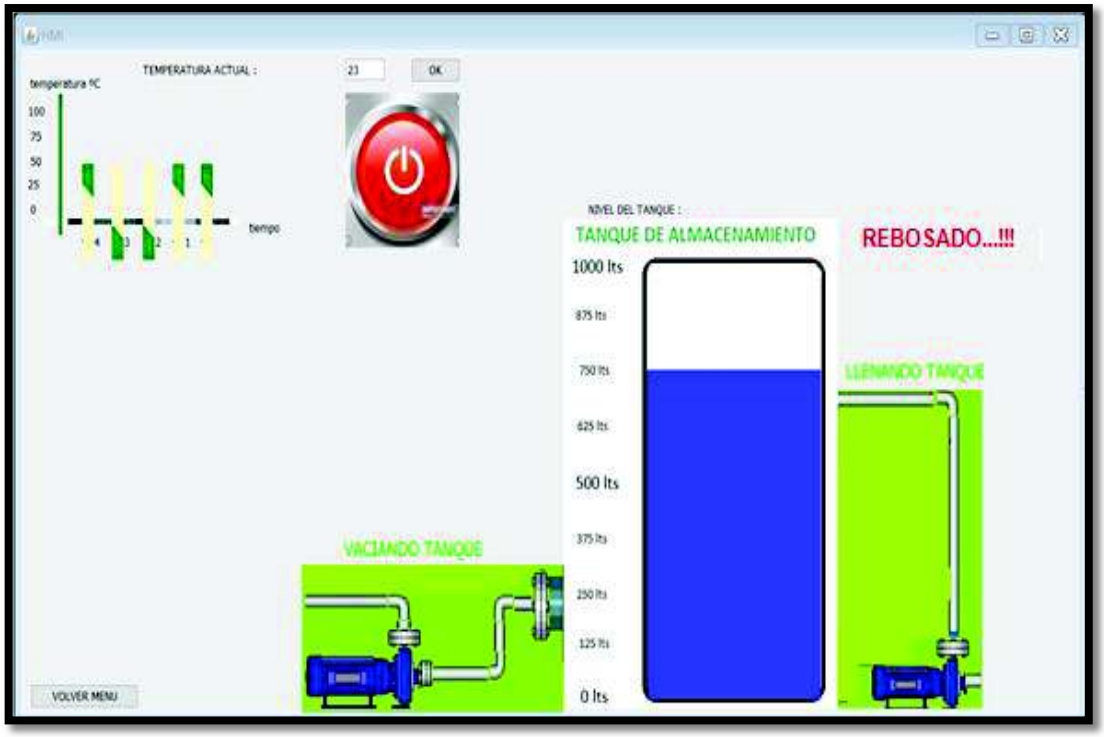


Figura 3.25. Interfaz Control de Nivel de Tanque

En el Código 3.40 se muestra el hilo timer2, encargado de redibujar el nivel del agua y determinar si el nivel esta sobre o bajo un determinado valor para de acuerdo a esto abrir o cerrar las válvulas de llenado y salida del agua.

```
Timer timer2 = new Timer (1000, new ActionListener ()
{
    public void actionPerformed(ActionEvent e)
    {
         a2 =Generar_Registros.getValorleido2();
        String b2= Integer.toString(a2);
        int resultado = (a2*452)/1000;
        int resultado2=452-resultado;
         lblnivel.setBounds(801, 220+resultado2, 222, 456-resultado2);
         lbllevel.setText(b2+" lts");
        repaint();
        if(a2>800){
            timer5.stop();
            contentPane.add(valvulaentrada);
            timer4.start();
            lblllenando.setVisible(false);
            lblvaciandotanque.setVisible(true);
            repaint();
        } else if(a2>250 && a2<=800 ){
            timer4.start();
            timer5.start();
            lblllenando.setVisible(true);
            lblvaciandotanque.setVisible(true);
            repaint();
        } else if (a2<=250){
            timer4.stop();
            contentPane.add(valvulasalida);
            timer5.start();
            lblllenando.setVisible(true);
            lblvaciandotanque.setVisible(false);
            repaint();
```

Código 3.40. Codificación Timer2 Control de Nivel

4 PRUEBAS Y AJUSTES AL PROTOTIPO DE SOFTWARE

En este capítulo se detallan los procesos de pruebas a los que fue sometido el prototipo de software para validar la funcionalidad de daca uno de los módulos y el funcionamiento en conjunto de todo el prototipo. En base a estas pruebas se desarrolló una etapa de ajustes tanto en la parte visual como de programación de las funcionalidades del prototipo.

Al concluir con estas dos etapas se escribirán las recomendaciones y conclusiones sobre los procesos realizados para la elaboración del prototipo de software.

4.1 Actualización del Tablero Kanban

Una vez terminada las fases de diseño e implementación, las tareas de codificación de cada uno de los módulos pasan al estado “Tareas en Fase de Pruebas” y las tareas de ajustes de cada uno de los módulos se mueven al estado de “Tareas en Proceso”. Esto se muestra en la Figura 4.1.

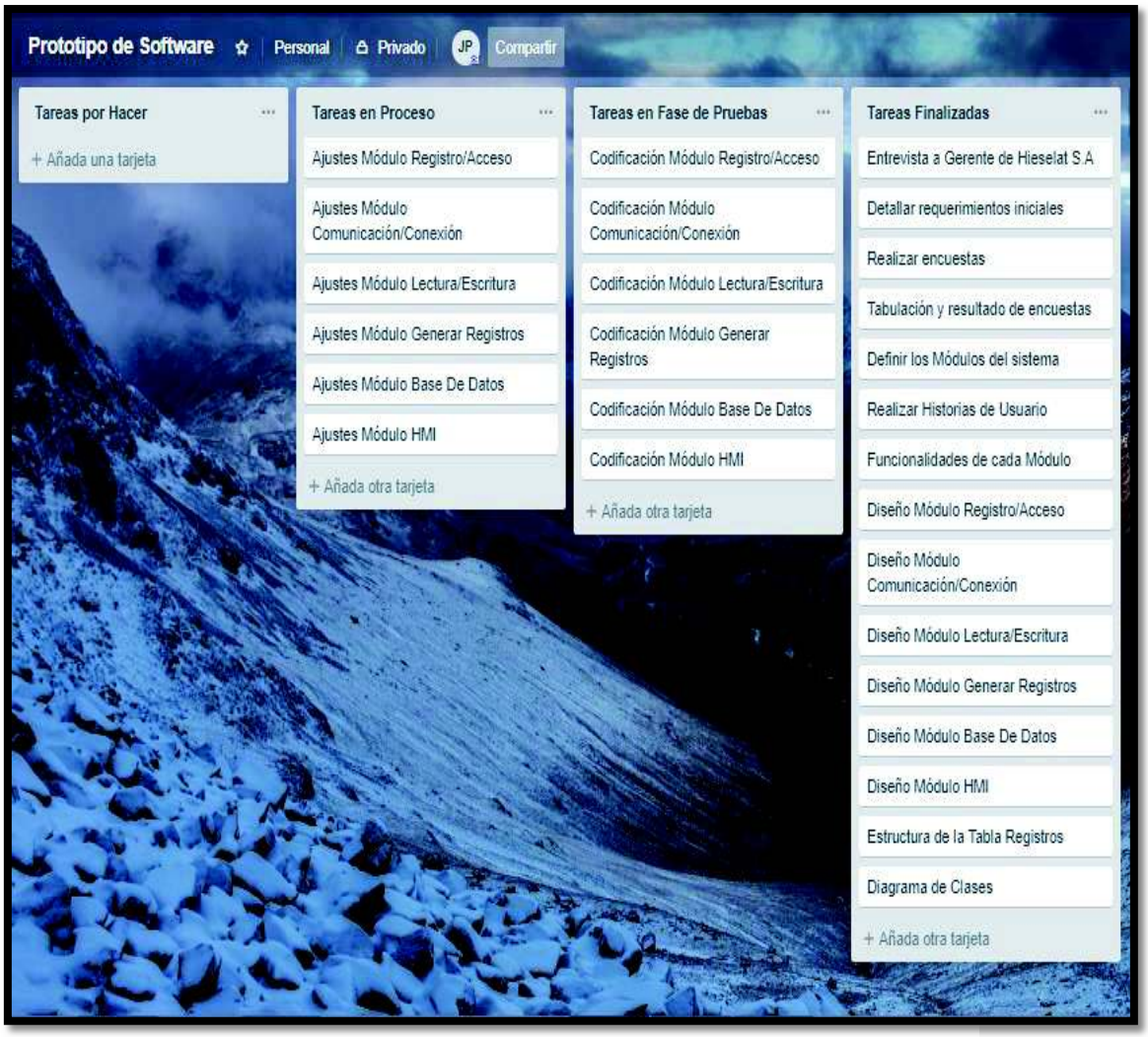


Figura 4.1. Actualización del Tablero Kanban

4.2 Pruebas

Para la etapa de pruebas se realizó lo especificado en el alcance del plan de titulación. Se realizaron pruebas de funcionalidad a cada uno de los módulos y de todo el prototipo para verificar su correcto funcionamiento, y la usabilidad del prototipo de software.

En la primera fase se realizó un formato de pruebas en Excel, para que los ingenieros Jhonny Curimilma y Daniel Rosales, Gerente General y Sub Gerente de la empresa Hieselat.SA respectivamente, evalúen la funcionalidad de cada uno de los módulos y de todo el prototipo. En la Tabla 4.1 se muestra el formato de las pruebas que se realizaron.

Tabla 4.1 Formato de Prueba

# PRUEBA:	1				
NOMBRE PRUEBA:	INGRESO CORRECTO DE CREDENCIALES				
ESCENARIO:	El operador ingresa a la interfaz gráfica Login e ingresa el nombre de usuario y password correctamente. Por lo tanto se habilita el Menú con las funcionalidades permitidas para el tipo de perfil.				
		ESTADO DE PRUEBA			OBSERVACIÓN
ITEMS:		**NO CUMPLE**	**EN PROCESO**	**CUMPLE**	
1	Se puede ingresar el nombre de usuario.				
2	Se puede ingresar el password.				
3	Existe un botón para verificar las credenciales.				
4	No se puede omitir el proceso de verificación de credenciales.				
5	Al ingresar las credenciales correctas brinda al acceso a las funcionalidades del menú de acuerdo al perfil.				
6	Se muestra el nombre de usuario y nivel de acceso del operador.				
7	Se puede cerrar la sesión de operador.				

Este formato tiene los siguientes campos:

- **#PRUEBA:** el número de prueba a realizar respecto a un módulo.
- **NOMBRE PRUEBA:** el identificativo o descripción de la prueba a realizar.
- **ESCENARIO:** este describe el escenario en el cual se va a realizar la prueba, aquí se define estado del maestro y esclavos. y también la interacción del usuario con el prototipo de software.

- **ÍTEMS:** se detalla cada una de los procesos en los cuales se validará su funcionalidad.

- **ESTADO DE PRUEBA:** para el estado de la prueba se establecen tres estado: NO CUMPLE, EN PROCESO Y CUMPLE. Con esto se determina si la funcionalidad del módulo evaluado cumple o no con el alcance definido anteriormente en el plan.

  En caso de que un ítem este en proceso se entenderá de dos formas diferentes.

  - Está en el alcance pero necesita realizar ajustes (NECESARIO IMPLEMENTAR CAMBIOS).

  - No está en el alcance pero sería recomendable implementarlo (OPCIONAL IMPLEMENTAR CAMBIOS, O DEJARLO COMO RECOMENDACIONES).

- **OBSERVACION:** aquí se detalla la manera de validación sobre algunos ítems, observaciones y recomendaciones sobre las funcionalidades de los módulos.

Con este formato se realizaron pruebas a cada uno de los módulos que conforman el prototipo de software. Todas las pruebas realizadas en cada módulo se encuentran adjuntas en el ANEXO D.

A continuación se presenta el resultado de las pruebas realizadas.

4.2.1 Pruebas Módulo Registro/Acceso Usuarios

Este módulo cumple con la funcionalidad de permitir al usuario ingresar al sistema a través de la verificación de credenciales y también acceso a las funcionalidades de acuerdo a un perfil de usuario. A continuación se presenta cada uno de los escenarios con sus respectivas pruebas (ÍTEMS).

ESCENARIO 1: El operador ingresa a la interfaz gráfica “LOGIN” e ingresa el nombre de usuario y password correctamente. Por lo tanto se habilita el Menú con las funcionalidades permitidas para el tipo de perfil.

ÍTEMS:

- Se puede ingresar el nombre de usuario.
- Se puede ingresar el *password*.
- Existe un botón para verificar las credenciales.
- No se puede omitir el proceso de verificación de credenciales.

- Al ingresar las credenciales correctas brinda al acceso a las funcionalidades del menú de acuerdo al perfil.
- Se muestra el nombre de usuario y nivel de acceso del operador.
- Se puede cerrar la sesión de operador.

PROCESO: Se ingresa las credenciales correctas con los dos tipos de perfiles, Administrador y Operador en el formulario de inicio "LOGIN". Automáticamente se muestra el formulario "MENÚ" con las funcionalidades habilitadas para cada perfil. A partir de la validación y acceso de credenciales, en cada uno de los formularios, se muestra en la parte inferior izquierda el nombre de usuario y nivel de acceso.

En las Figuras 4.2 y 4.3 se observa el ingreso del usuario "Administrador" y "Operador".

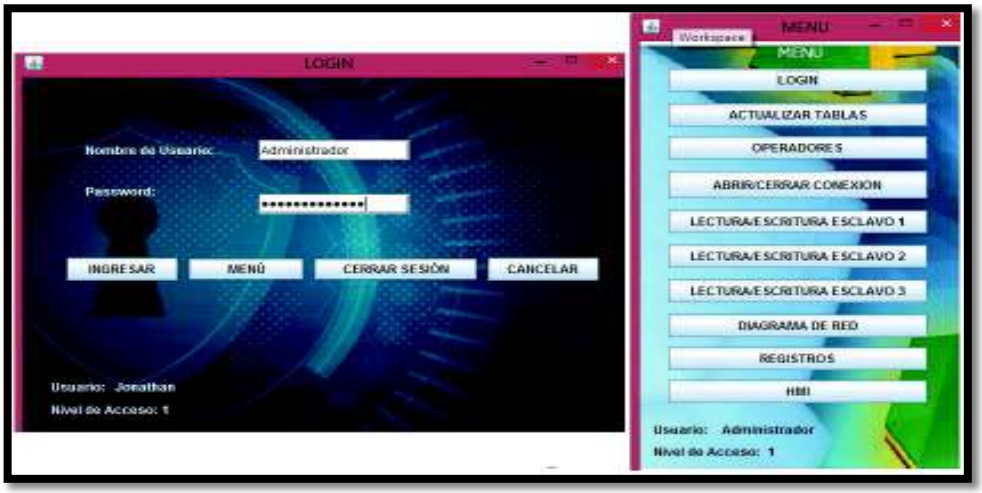


Figura 4.2. Ingreso del Usuario "Administrador"

Figura 4.3. Ingreso del Usuario "Operador"

ESTADO DE LA PRUEBA: esta prueba cumple con todos los ítem planteados.

ESCENARIO 2: En la interfaz gráfica “LOGIN” se ingresa el nombre de usuario o password incorrectos, por lo tanto tiene la opción de cancelar el proceso de login o recuperar las credenciales mediante el correo electrónico.

ÍTEMS:

- Se muestra un mensaje de credenciales incorrectas.
- Existe un botón para cancelar el proceso de validación de credenciales.
- Se habilita un botón para recuperar credenciales.
- Se muestra una interfaz gráfica para recuperar las credenciales mediante correo electrónico.
- Se envía el correo electrónico con las credenciales.
- Se indica que no se puede recuperar credenciales en caso de un correo no existente.

PROCESO: El usuario ingresa sus credenciales de forma incorrecta, por lo tanto el sistema muestra un mensaje de ERROR notificando que las credenciales son incorrectas. Al mismo tiempo se habilita un botón para recuperar las credenciales a través del correo electrónico registrado con ese usuario. Este proceso se observa en la Figura 4.4.

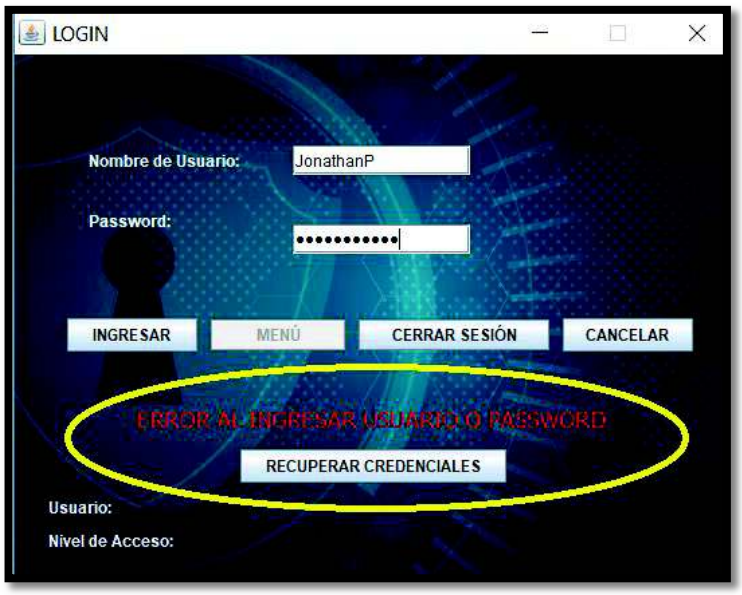


Figura 4.4. Ingreso de Credenciales Incorrectas

Al seleccionar la opción recuperar credenciales se muestra un nuevo formulario en el cual se ingresa la dirección de correo electrónico y las credenciales son enviadas como se muestra en la Figura 4.5.

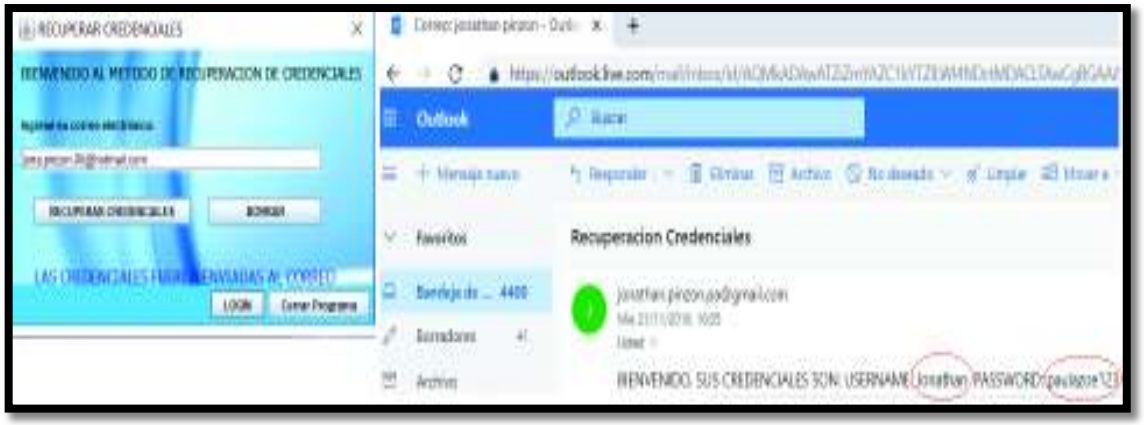


Figura 4.5. Envío de Credenciales al Correo Electrónico

Cuando el correo electrónico no esté registrado dentro de la base de datos o por algún motivo no pueda ser enviado por motivos de conectividad con el Internet o esté caído el servidor de correo y se muestra un mensaje de que existió un problema al recuperar las credenciales. En la Figura 4.6 se observan los dos casos que pueden ocasionar el NO envío de credenciales al correo.

Figura 4.6. Fallo en Envío de Credenciales

ESTADO DE LA PRUEBA: esta prueba cumple con todos los ítem planteados.

En el campo de observaciones existen dos.

- El título del botón “Cancelar” en el formulario “Login” debería cambiarse por el título “Limpiar Campos” para que sea más intuitiva la acción.
- El mensaje de NO envío de correo electrónico debería informarme si la acción no se realizó por motivo de correo inexistente o problemas con la conexión del servidor de correo o Internet.

ESCENARIO 3: El operador ingresa a la interfaz gráfica Operadores y allí crea, elimina y actualiza usuarios con todos sus atributos.

ÍTEMS:

- Se muestra la tabla donde están los usuarios creados.
- Al seleccionar un usuario en particular sus valores se muestran en las cajas de texto.
- Al crear un nuevo usuario se valida los campos que no pueden ser NULL y de igual manera los campos que deben ser UNIQUE (no se pueden repetir).
- Se verifica que posea un perfil existente.
- Se puede crear un nuevo operador en la tabla.
- Se puede actualizar un operador.
- Se puede eliminar un operador.

PROCESO: se crea un usuario ingresando todos los campos y esta acción se refleja tanto en la tabla mostrada en el prototipo como en la base de datos.

En la Figura 4.7 se muestra la creación de un nuevo usuario llamado “Administrador” con un perfil de administrador, su clave “admin123” y correo “admin.prototipo@hotmail.com”, la tabla usuarios se actualiza automáticamente.

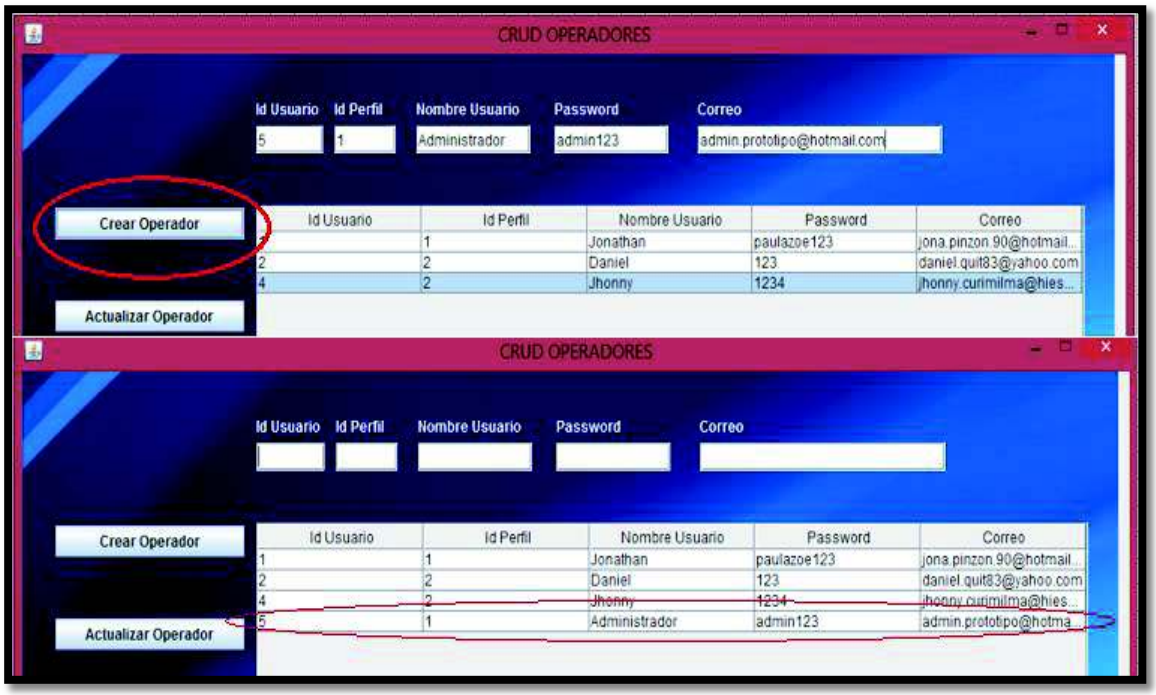


Figura 4.7. Crear Usuario “Administrador”

Se selecciona el usuario “Administrador” de la tabla para actualizar su atributo *password* de “admin123” a “admin”. Esto se muestra en la Figura 4.8.

Figura 4.8. Actualizar Password del Usuario “Administrador”

Para eliminar al usuario “Administrador” primero se selecciona de la tabla y luego se presiona el botón eliminar. La tabla al igual que en la funcionalidades anteriores se actualizará automáticamente. Esto se muestra en la Figura 4.9.

Figura 4.9. Eliminar Usuario “Administrador”

Existen varias validaciones que se realizan antes de crear, actualizar y eliminar usuarios, esto asegura que no se generen excepciones en el prototipo ni tampoco en la base de datos. Cada una de las validaciones muestra un mensaje de error para notificar al usuario que el o los parámetros de entrada están incorrectos. Estas validaciones son.

- No ingresar todos los campos para crear un usuario
- Ingresar un Id Usuario existente
- Ingresar un Perfil no existente
- Crear un usuario con un nombre de usuario repetido
- Actualizar un usuario sin seleccionar a ninguno
- Eliminar un usuario sin seleccionar a ninguno

Algunas de estas validaciones se muestran en la Figura 4.10.

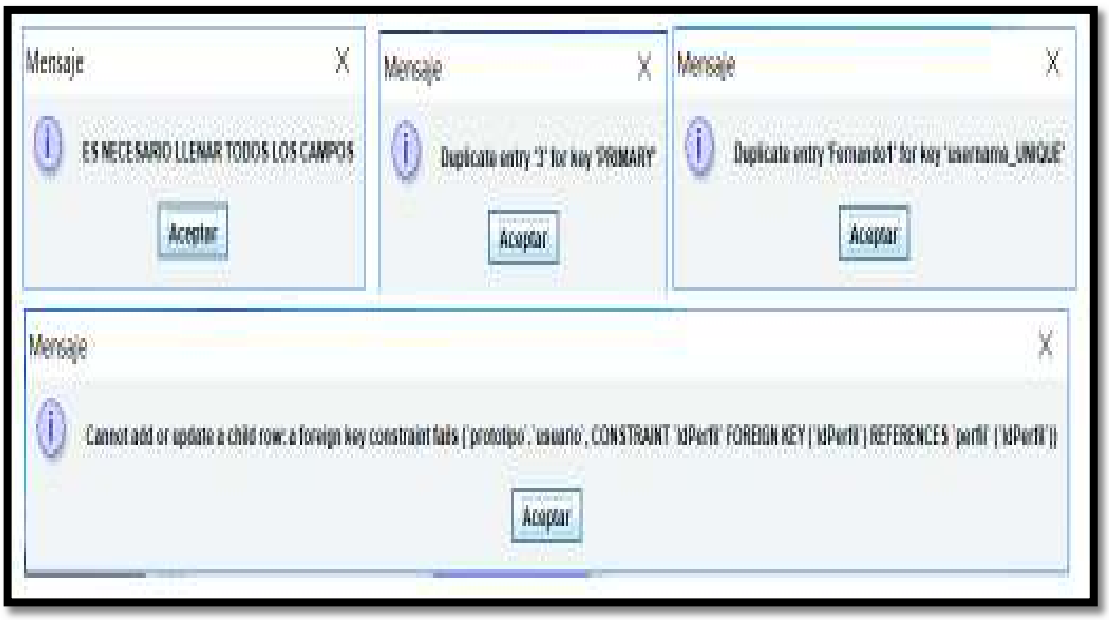


Figura 4.10. Mensajes de Error por Validación

ESTADO DE LA PRUEBA: esta prueba cumple con todos los ítem planteados.

Existen tres observaciones sobre esta prueba.

La primera es que debería existir un perfil más, uno en el que se tenga la opción de crear, modificar y eliminar usuario, mientras que el otro perfil de Administrador únicamente le permite visualizar la tabla usuarios con sus claves y correos. Mientras que el perfil Operador no tiene acceso a esta información.

Esta observación será implementada en la etapa de ajuste, por lo tanto la tabla perfiles estará conformada de la siguiente manera.

- SuperUsuario: 1
- Administrador: 2
- Operador: 3

La segunda observación es añadir un mensaje de confirmación cuando el usuario quiera crear, modificar y actualizar algún usuario en la tabla. Este cambio también será implementado en la etapa de ajustes.

La tercera observación se trata de implementar una funcionalidad adicional al prototipo de software de lo establecido en el alcance. Se sugiere un sistema de logs, en el cual se

registre todas las actividades que realizan cada uno de los usuarios que manipulan el prototipo de software.

Considerando esto como una funcionalidad importante pero al mismo tiempo que significa un cambio considerable dentro de la programación, quedará como recomendación para ser implementada en versiones futuras del prototipo de software.

4.2.2 Pruebas Módulo Comunicación/Conexión

Este módulo cumple con la funcionalidad de establecer la conexión y mantener una comunicación entre el prototipo de software y los esclavos. Para esto el usuario ingresa las direcciones IP de los esclavos y realiza el proceso de establecer la conexión a través del botón.

También se puede observar el estado de conectividad de los esclavos con el prototipo de software en la interfaz “Diagrama de Red”. A continuación se presenta cada uno de los escenarios con sus respectivas pruebas (ÍTEMS).

ESCENARIO 1: El operador entra a la interfaz gráfica "Establecer Conexión" e ingresa las direcciones IP de hasta tres esclavos que se encuentren en red. Se mostrará un mensaje y estado de la conexión.

ÍTEMS:

- Se puede ingresar las direcciones IP de los esclavos.
- Existe un botón para establecer la conexión.
- Se muestra un mensaje de establecimiento y estado de la conexión.
- Existe un botón para cerrar la conexión.

PROCESO: el usuario ingresa las direcciones de los esclavos y presiona el botón establecer conexión. Automáticamente el estado de conectividad del esclavo se actualizará. Este proceso se muestra en la Figura 4.11.

Al igual que el botón para establecer la conexión también existe uno para cerrar la conexión, con esto se cierra el flujo de comunicación y el socket respectivo.

ESTADO DE LA PRUEBA: esta prueba cumple con todos los ítem planteados.
No existen observaciones.

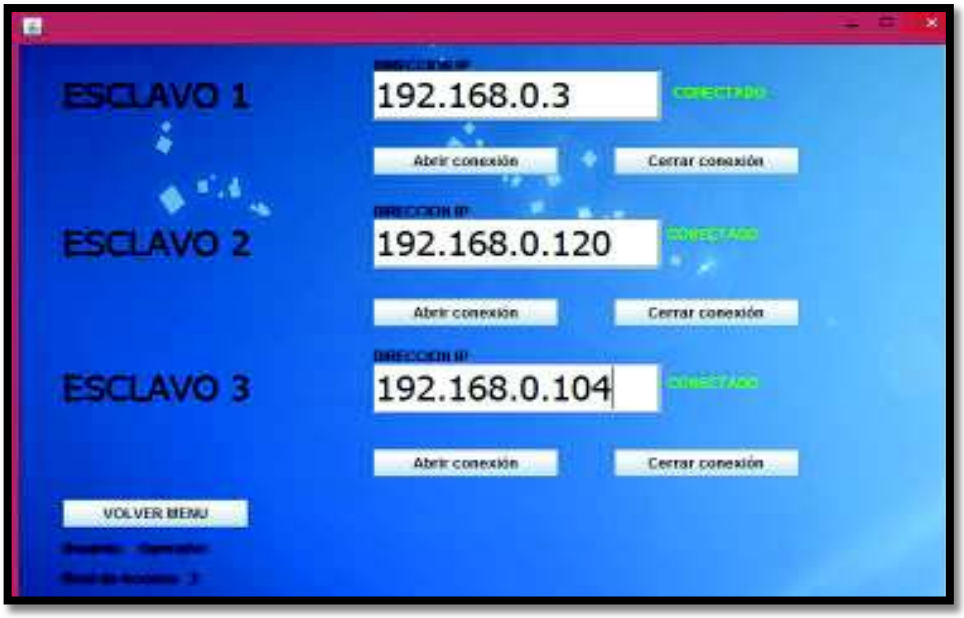


Figura 4.11. Establecimiento de Conexión con Tres Esclavos

ESCENARIO 2: El operador ingresa a la interfaz gráfica "Diagrama de Red" para verificar el estado de la conexión con cada uno de los tres esclavos. Aleatoriamente se desconectará a uno o varios esclavos, también se cerrará la conexión desde el prototipo de software para observar que el diagrama de conexión muestra la información correcta.

ÍTEMS:

- Se identifica a cada uno de los esclavos y el estado de la conexión con el maestro.
- Al desconectar o apagar a un esclavo, esto se refleja en el diagrama de red.
- Al cerrar la conexión con uno o varios esclavos, esto se refleja en el diagrama de red.
- Al reestablecer la conexión con los esclavos, esto se refleja en el diagrama de red.

PROCESO: el usuario ingresa a la interfaz gráfica “Diagrama de Red”, allí observa que los esclavos se encuentran conectados y tienen una comunicación establecida con el maestro mostrando una flecha de color verde. Este proceso se observa a continuación en la Figura 4.12.

Para el ítem número 2 se desconecta el cable de red del esclavo físico (PLC), en un lapso de tiempo de 20 segundos se muestra un mensaje de perdida de conexión con el esclavo, la flecha verde se cambia por una X de color rojo, para representar que se ha perdido la conexión. Esto se presenta en la Figura 4.13.

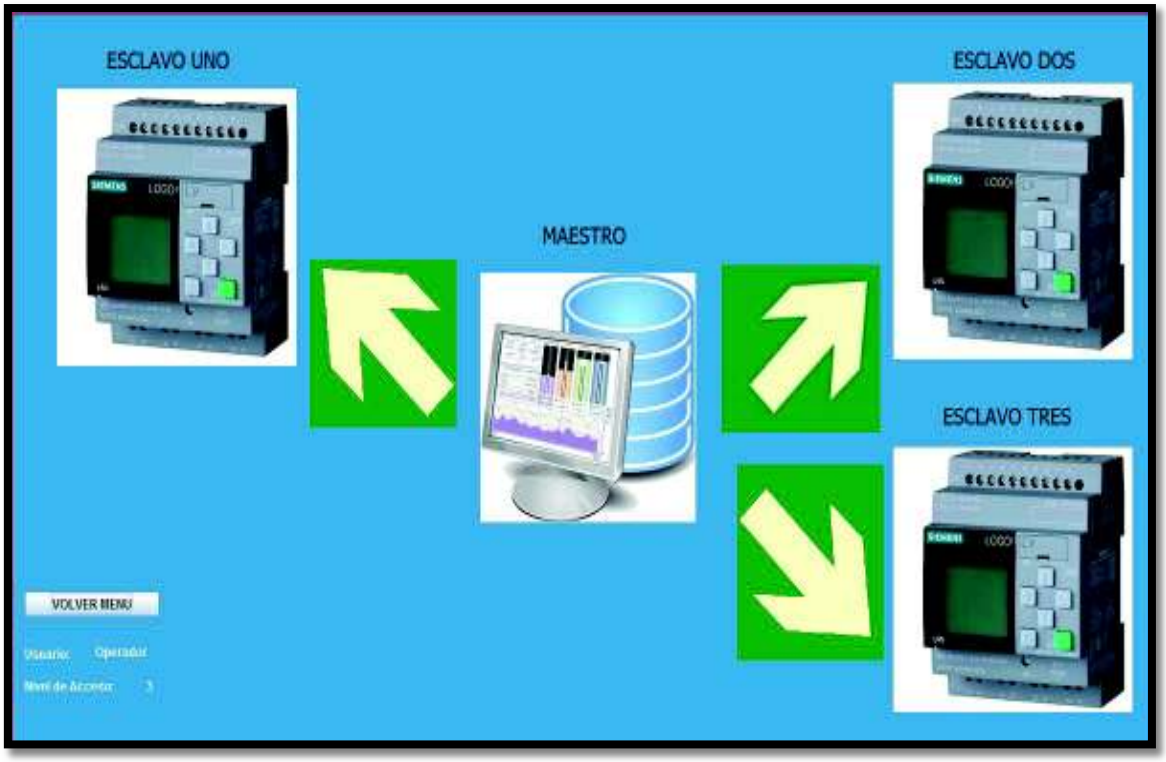


Figura 4.12. Conectividad Esclavos

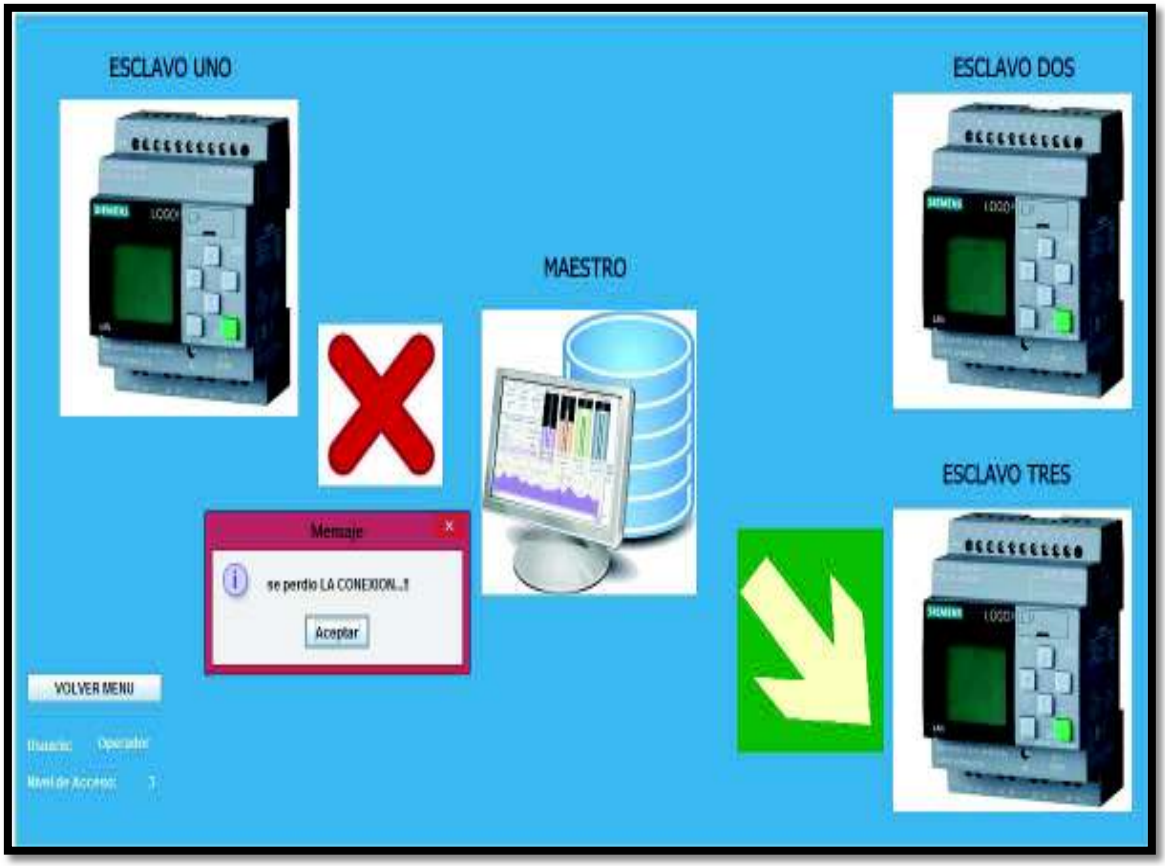


Figura 4.13. Desconexión del Esclavo 1 (PLC)

Se cierra la conexión del segundo esclavo y se visualiza el resultado en la interfaz “Diagrama de Red”, por último se desconecta el cable de red donde se encuentra corriendo el MAESTRO (prototipo de software). El resultado de estas pruebas se muestra en la Figura 4.14.

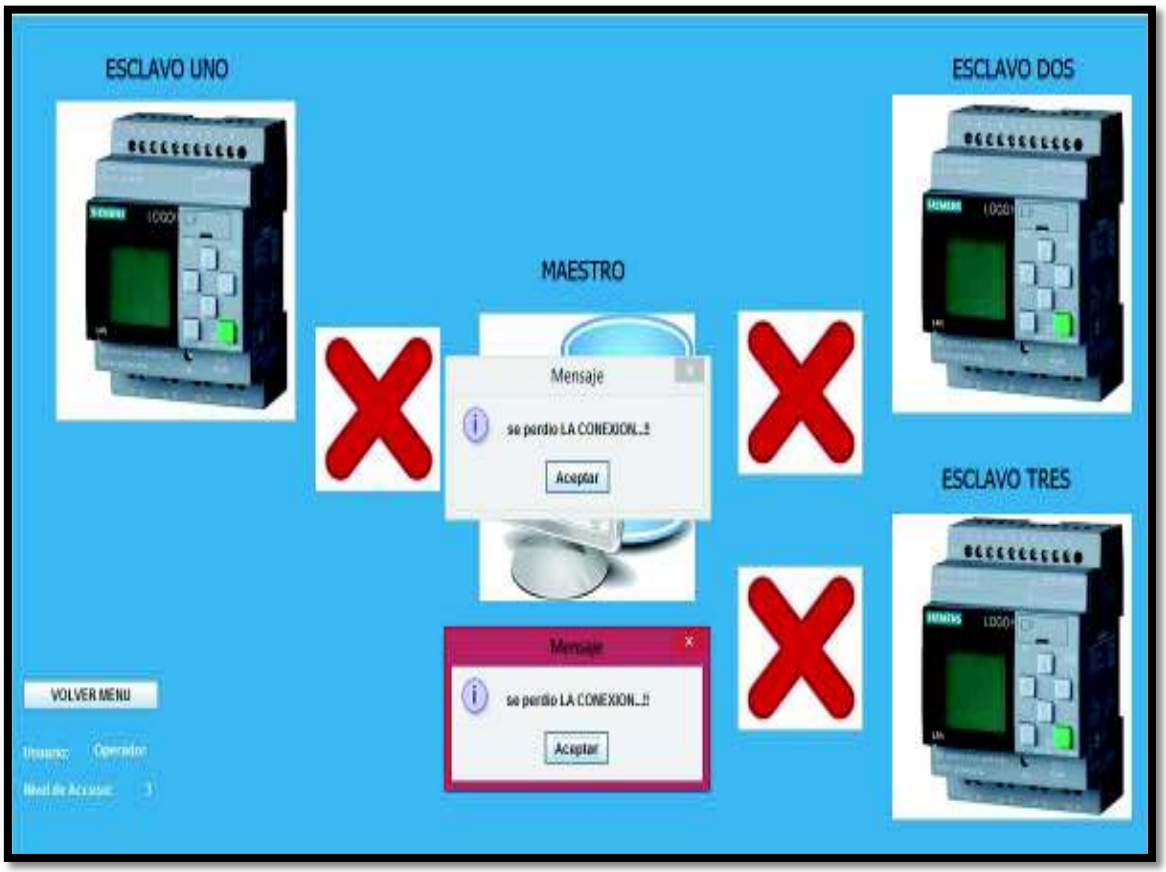


Figura 4.14. Desconexión Esclavos

ESTADO DE LA PRUEBA: esta prueba cumple con todos los ítem planteados.

Existen dos observaciones.

La primera observación plantea que los nombres con los que se identifica a cada esclavo: ESCLAVO 1, ESCLAVO 2 Y ESCLAVO 3 deberían ser editables, y que cada operador pueda identificar a los esclavos con el nombre que crea conveniente.

La segunda observación propone una funcionalidad adicional que puede representar una mejora en el módulo de establecer conexión.

La funcionalidad propone la existencia de un hilo independiente el cual se lance una vez establecida la conexión con un esclavo, este hilo estará censando la perdida de conexión y una vez ocurrida tratará de reestablecerla.

El adicionar esta funcionalidad implica la creación de nuevos hilos además de identificar el motivo de la pérdida de desconexión y en base a eso programar un método para intentar reestablecerla. Por tal motivo esta observación será tomada como una sugerencia de mejora para implementarlas en versiones futuras del prototipo.

4.2.3 Pruebas Módulo Lectura/Escritura

Este módulo cumple con la funcionalidad de leer y escribir en 4 de los tipos de registros que tienen los esclavos de una red industrial y que se comunican mediante el protocolo ModBus TCP. Los tipos de registros que se van a leer son: *Coil Status, Input Status, Input Registers y Holding Registers*, mientras que para el proceso de escritura solo se lo hará en los dos registros de salida los cuales son: *Coil Status y Holding Registers*.

A continuación se presenta cada uno de los escenarios con sus respectivas pruebas (ÍTEMS).

ESCENARIO 1: EL usuario lee los primeros 10 valores de los registros: *Coil, Input Status, Input Register y Holding Register*, en cada uno de los esclavos. Estos valores son comparados con los registros de los esclavos.

ÍTEMS:

- La lectura realizada refleja los mismos valores almacenados en los registros de los esclavos virtuales.
- La lectura realizada refleja los mismos valores almacenados en los registros del esclavo físico (PLC).
- Se visualiza el resultado al ejecutar la funcionalidad de lectura de registros.

PROCESO: en cada uno de los esclavos se realiza la lectura de los cuatro tipos de registros, para validar este proceso se compara los valores obtenidos en el prototipo con las capturas de pantalla en caso del esclavo virtual y con fotos de la pantalla en caso del PLC. Las Figuras de lectura de todos los esclavos se muestran en el ANEXO D.

En la Figura 4.15 se muestra los valores de los cuatro tipos de registros que tiene uno de los esclavos virtuales.

Los valores mostrados en esta figura se comparan con los que se muestran en la Figura 4.16 la cual representa la captura de pantalla del prototipo.

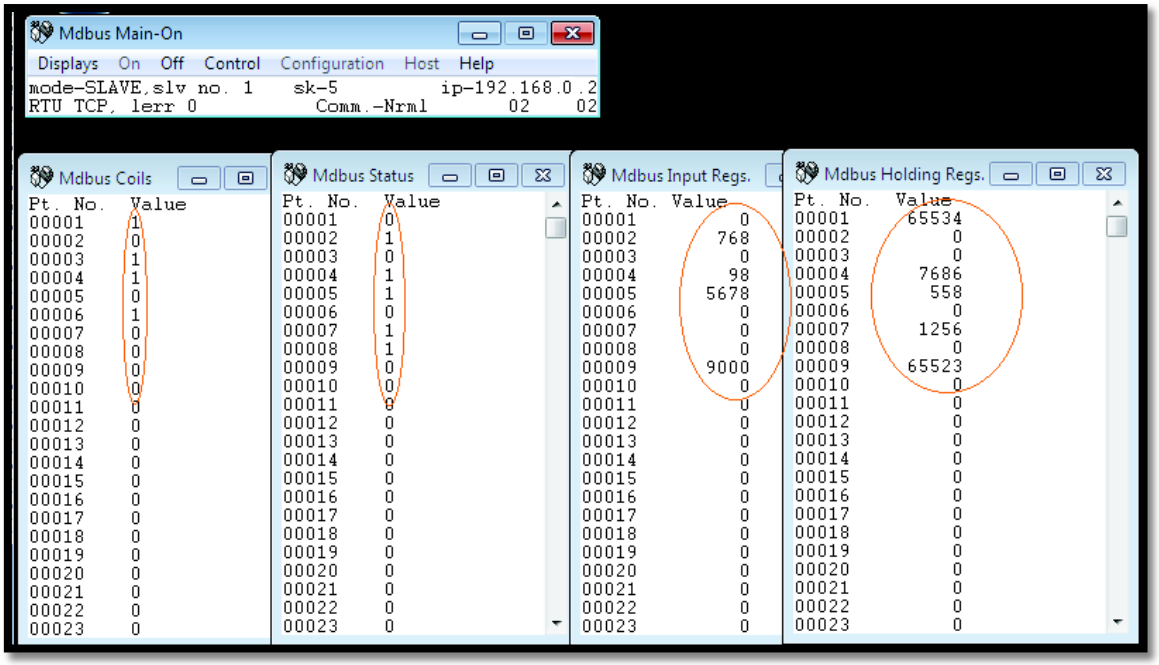


Figura 4.15. Lectura Registros Esclavo Virtual

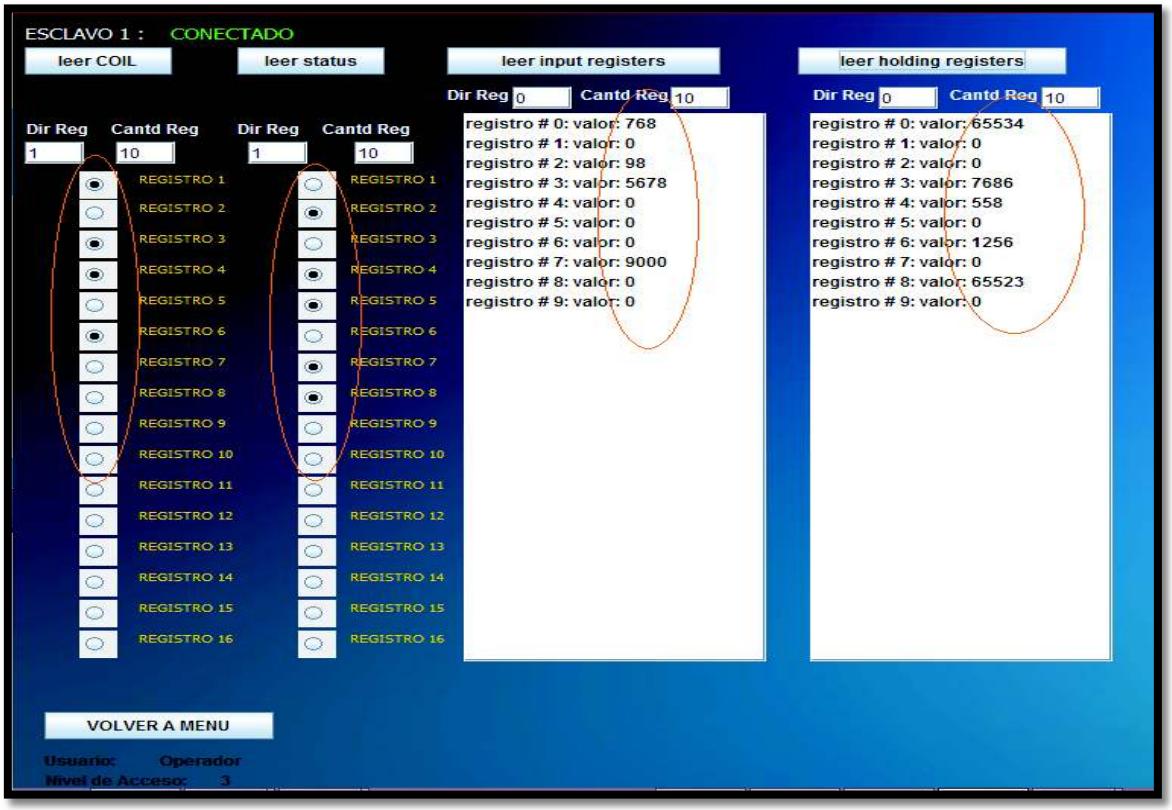


Figura 4.16. Registros Leídos desde el Prototipo de Software

ESTADO DE LA PRUEBA: esta prueba cumple con todos los ítem planteados.

No existen observaciones.

ESCENARIO 2: El operador escribe en el tipo de registro *Coil* la siguiente secuencia: 1010101100 en los 10 primeros *Coils*. Y en los registros del tipo *Holding registers* se escribe los siguientes valores: 65535, 0, 37890, 200 y 500, en las 5 primeras direcciones de cada esclavo.

ÍTEMS:

- Se escribe en los registros *Coil*.
- Se escribe los valores en los registros Holding *Registers*.
- La funcionalidad escritura de registros se ejecuta y se refleja este cambio en los registros del esclavo.

PROCESO: el usuario escribe en los registros los valores especificada en el escenario y se compara como en el escenario 1.

En la Figura 4.17 se observa un ejemplo del proceso de escritura sobre los registros *Coil Registers* y *Holding Registers* respectivamente.

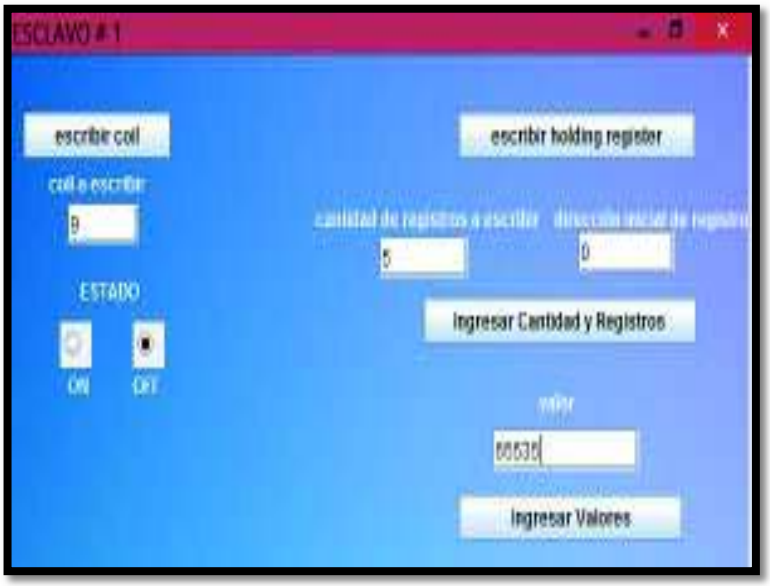


Figura 4.17. Escritura de Registros

Para comprobar el proceso se utiliza la funcionalidad escribir registros y de esta manera se observa la modificación de los registros. Esto se observa en la Figura 4.18.

Para comparar el proceso de escritura se verifica los valores del esclavo desde el simulador. Estos valores se muestran a continuación en la Figura 4.19.

ESTADO DE LA PRUEBA: esta prueba cumple con todos los ítem planteados.
No existen observaciones.

Figura 4.18. Verificación de Registros

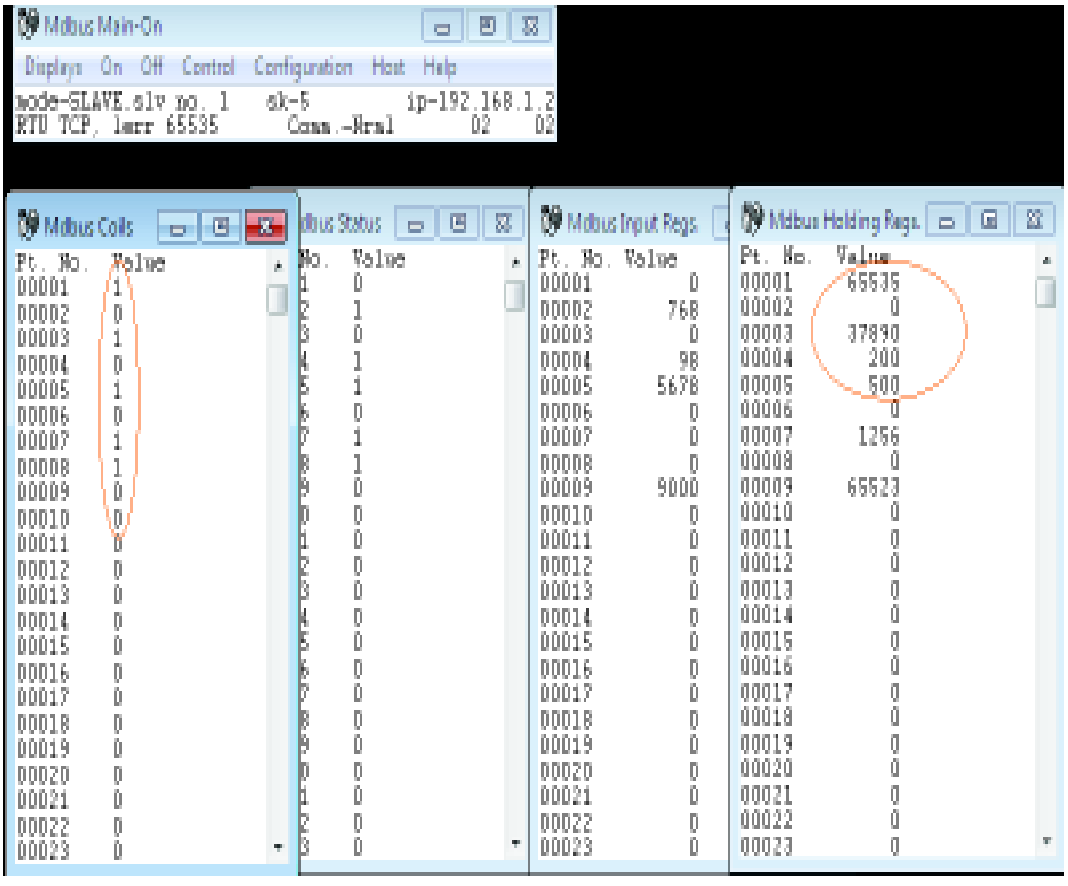


Figura 4.19. Comparación de Valor de Registros

ESCENARIO 3: El operador ingresa a la interfaz gráfica de lectura /escritura y encuentra un entorno donde puede diferenciar a cada una de las 4 funciones de lectura y las dos funciones de escritura.

ÍTEMS:

- Se diferencia cada una de las funciones de lectura y escritura.
- Se entiende y valida los parámetros que debe ingresar para el proceso de lectura y escritura (de acuerdo al mapa de direccionamiento).
- Se muestra el estado de la conexión con el esclavo.

PROCESO: el usuario utiliza las dos funcionalidades Lectura y Escritura mediante la interfaz gráfica diseñada para cada esclavo.

ESTADO DE LA PRUEBA: esta prueba tiene un ítem en proceso por lo tanto no cumple al 100% con su funcionalidad.

El ingreso de valores en la escritura de registros, debe ser cambiado para lograr un proceso más intuitivo con el usuario. Este cambio está dentro del alcance del módulo lectura/escritura por lo tanto será implementado en la fase de ajustes.

4.2.4 Pruebas Módulo Generar Registros

Este módulo cumple con la funcionalidad de generar un histórico con tres registros que el usuario elija, para esto se debe ingresar un tiempo de muestreo y un tiempo total durante el cual se va a estar guardando estos valores en la base de datos.

A continuación se presenta cada uno de los escenarios con sus respectivas pruebas (ÍTEMS).

ESCENARIO 1: El operador elije tres registros, sus valores corresponden a la temperatura, el nivel del tanque y la velocidad de llenado (caudal de entrada). Se genera un historial con estos tres valores, se agrega información como nombre del historial, nombre de la variable, operador que realizó el registro, tiempo de muestreo y tiempo total.

ÍTEMS:

- La se puede elegir a cada uno de los registros del tipo o esclavo al que pertenezca.
- Se detalla cada uno de los campos que se debe ingresar.
- Se valida los campos que se debe ingresar como tiempos de muestreo y tiempo total.

- Existe un botón para generar el histórico.
- Se muestra un mensaje cuando el proceso de crear el registro se ha terminado.

PROCESO: en la interfaz gráfica “Generar Registros” se elige tres registros de cualquier esclavo y estos representarán las variables antes mencionadas. Esto se observa en la Figura 4.20.

Figura 4.20. Generar Registros

ESTADO DE LA PRUEBA: esta prueba no cumple con la funcionalidad esperada, a pesar de que en el plan se especifica un alcance de generar un historial solo con tres variables, esto no se adapta a un escenario real ya que debería ser un número de registros aleatorio y muchos más que 3 registros a la vez.

Además de existir algunas observaciones de forma, por cual todas estas observaciones serán implementadas en la etapa de ajustes para cumplir en un 100% con la funcionalidad de este módulo.

ESCENARIO 2: El operador crea dos históricos con tiempos específicos: Histórico1 y Histórico2.

PROCESO: se crea los históricos con los tiempos especificados en la prueba:

Histórico1: los valores se tomarán cada 5 segundos por un periodo de 10 minutos.

Histórico2: los valores se tomarán cada 5 minutos por un periodo de dos horas.

ESTADO DE LA PRUEBA: esta prueba cumple con todos los ítem planteados.
No existen observaciones.

4.2.5 Pruebas Módulo Base de Datos

Este módulo cumple con la funcionalidad de permitir al usuario una conectividad hacia la base de datos en cada una de las funcionalidades. Además de visualizar las tablas de la base en cualquier momento con la información actualizada.

A continuación se presenta cada uno de los escenarios con sus respectivas pruebas (ÍTEMS).

ESCENARIO 1: EL usuario verificará que todas las tablas de la base de datos estén actualizadas conforme a las funcionalidades ejecutadas en el prototipo.

ÍTEMS:

- Actualización tabla Usuarios.
- Actualización tabla Registros.

PROCESO: desde el proceso de login hasta la funcionalidad de generar registros el usuario está realizando constantemente la conexión a la base de datos y realizando consultas a cada una de las tablas. En la opción del menú “Actualizar Tablas” se muestra cada una de las tablas y se compara con lo que se tiene en la base de datos.

Esta comparación se muestra en las Figuras 4.21, 4.22 y 4.23.

Figura 4.21 Prototipo de Software Actualizar Tablas

IdPerfil	nombreperfil
1	SuperUsuario
2	Administrador
3	Operador
NULL	NULL

IdUsuario	IdPerfil	username	passwordUser	correo
1	1	Jonathan	paulazoe123	jona.pinzon.90@hotmail.com
2	1	Daniel	123	daniel.quit83@yahoo.com
4	1	Jhonny	1234	jhonny.curimilma@hieselat.com.ec
6	3	Operador	operador	jhonny.curimilma@hieselat.com.ec
NULL	NULL	NULL	NULL	NULL

Figura 4.22 Tablas Perfiles y Usuarios de la Base de Datos

IdRegistro	operador	nombreRegistro	numeroEsclavo	numeroRegistro	nombreVariable	valorVariable	fechaMuestreo	horamuestra
1	Jonathan	Historial 1	2	1	temperatura	25	2018-11-16	22:00:00
2	Jonathan	Historial 0	1	1	reg0	0	2018-11-26	01:41:30
3	Jonathan	Historial 0	1	2	reg0	45	2018-11-26	01:41:30
4	Jonathan	Historial 0	1	3	reg0	56	2018-11-26	01:41:30
5	Jonathan	Historial 0	1	1	holding1	0	2018-11-26	01:41:30
6	Jonathan	Historial 0	1	1	holding2	45	2018-11-26	01:41:30
7	Jonathan	Historial 0	1	1	holding3	56	2018-11-26	01:41:30
8	Jonathan	Historial 0	1	1	holding4	23	2018-11-26	01:41:30
9	Jonathan	Historial 0	1	1	holding5	0	2018-11-26	01:41:30
10	Jonathan	Historial 0	1	1	holding6	0	2018-11-26	01:41:30
11	Jonathan	Historial 0	1	1	holding7	0	2018-11-26	01:41:30
12	Jonathan	Historial 0	1	1	holding8	0	2018-11-26	01:41:30
13	Jonathan	Historial 0	1	1	holding9	0	2018-11-26	01:41:30
14	Jonathan	Historial 0	1	1	holding10	111	2018-11-26	01:41:30
15	Jonathan	Historial 0	1	1	holding11	2222	2018-11-26	01:41:30
16	Jonathan	Historial 0	1	1	holding12	3333	2018-11-26	01:41:30
17	Jonathan	Historial 0	1	1	holding13	0	2018-11-26	01:41:30

Figura 4.23. Tabla Registros de la Base de Datos

ESTADO DE LA PRUEBA: esta prueba cumple con todos los ítem planteados.
No existen observaciones.

4.2.6 Pruebas Módulo GUI

Este módulo cumple con la funcionalidad de mostrar al usuario una interfaz gráfica en la cual pueda interactuar con los esclavos. Aquí se muestra la interfaz especifica HMI (Interfaz Maquina/Humano) en la cual se utiliza los registros de los esclavos para simular proceso de control industrial.

A continuación se presenta cada uno de los escenarios con sus respectivas pruebas (ÍTEMS).

ESCENARIO 1: Al abrir la interfaz HMI se visualiza el control de temperatura, el umbral de temperatura predeterminado es de 23°C (temperatura corporal). Se irá variando la temperatura para observar si las barras reflejan los valores leídos y aparece la alarma de sobre temperatura.

ÍTEMS:

- Las barras representan los valores leídos.
- Las barras van guardando mediciones previas de temperatura.
- Al sobrepasar el umbral de temperatura se detiene la medición y se identifica la hora y fecha del evento.
- El valor de umbral de temperatura es modificable.

PROCESO: el usuario ingresa a la interfaz gráfica "HMI" el primer control que se realiza es el de temperatura, se debe configurar el valor de umbral para determinar la condición de sobre temperatura. Cada valor leído de temperatura será representado con las barras tal como se muestra en la Figura 4.24.

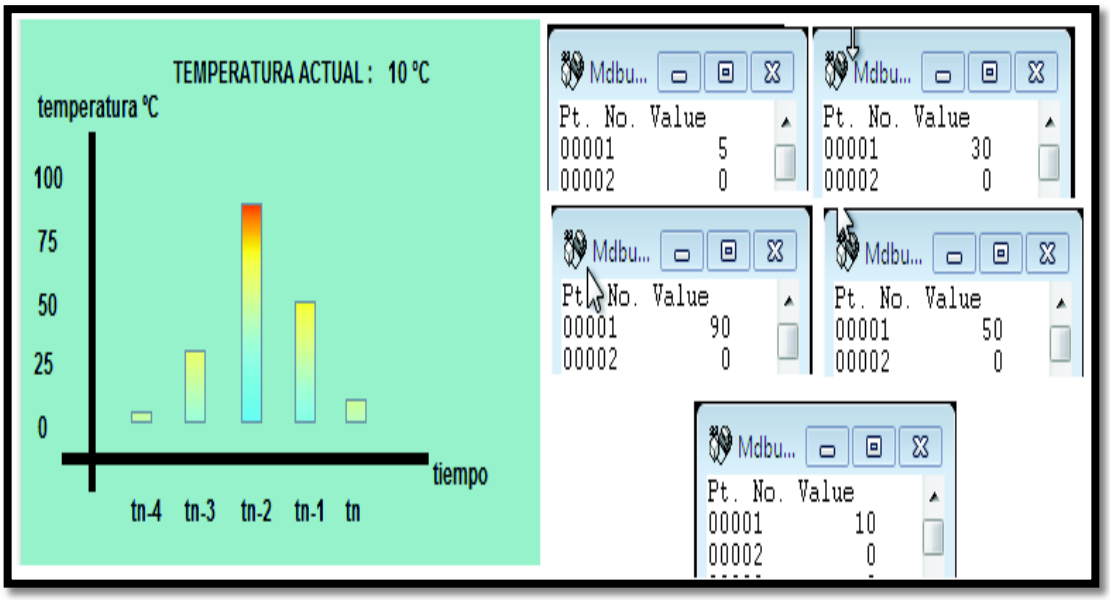
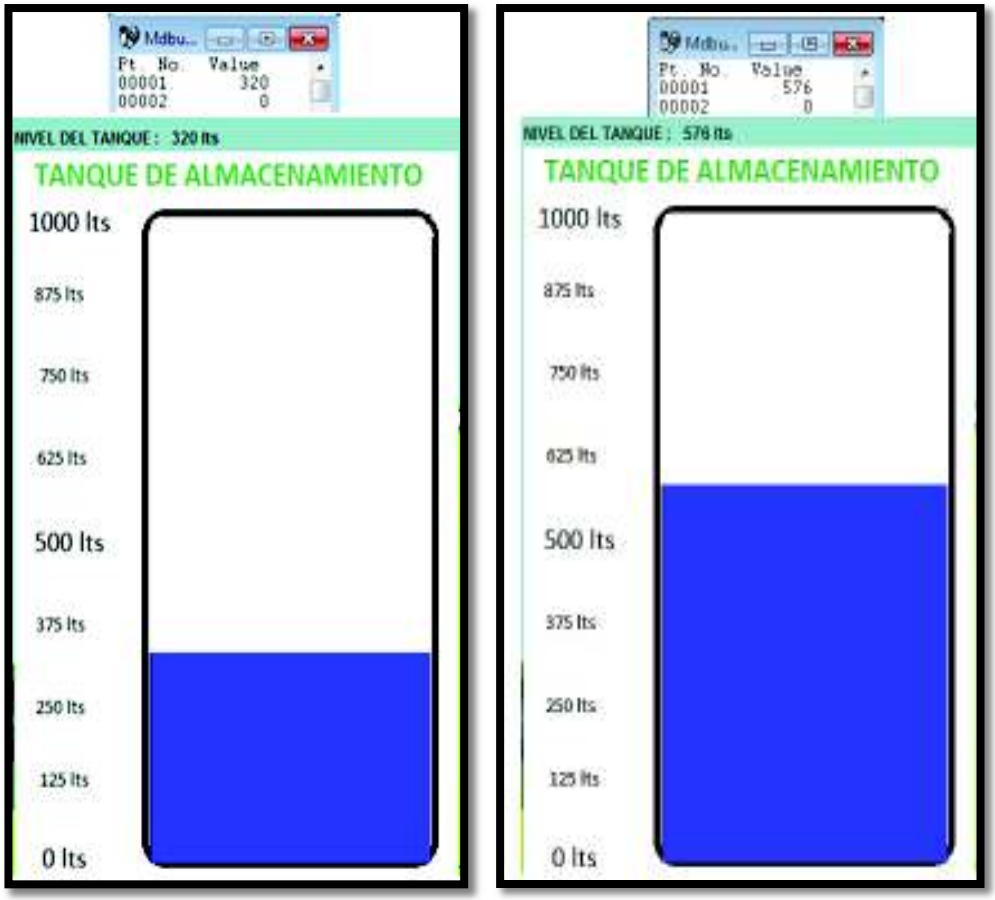


Figura 4.24. Lectura de Temperatura

El nivel del tanque irá variando de acuerdo al caudal de salida (velocidad de vaciado) el cual es un valor variable y el caudal de entrada (velocidad de llenado) es constante.

El caudal de llenado es tomado desde un registro de uno de los esclavos, el nivel del tanque irá variando en base a los dos valores descritos anteriormente y se sobrescribirá en un registro de salida del tipo *Holding Registers* como se muestra en la Figura 4.25.

ESTADO DE LA PRUEBA: esta prueba cumple con todos los ítem planteados.
No existen observaciones.



3

Figura 4.25. Lectura de Nivel del Tanque

4.3 Análisis de Trama ModBus TCP con *Wireshark*

Se realizaron dos pruebas adicionales, ya no con el cliente Hieselat, sino de una manera más personal para verificar la estructura de las tramas tanto de petición (*Query*) como de respuesta (Response).

Para esto se ha utilizado la herramienta *Wireshark*, un analizador de protocolos que viajan sobre la red, con esto se va a verificar que la estructura definida y codificada en cada una de las clases se esté enviando correctamente.

Para visualizar las tramas se va tomar dos ejemplos: Leer Coil, Escribir Holding Registers.

Se realizó la lectura y escritura tal como se muestra en la Figura 4.26.

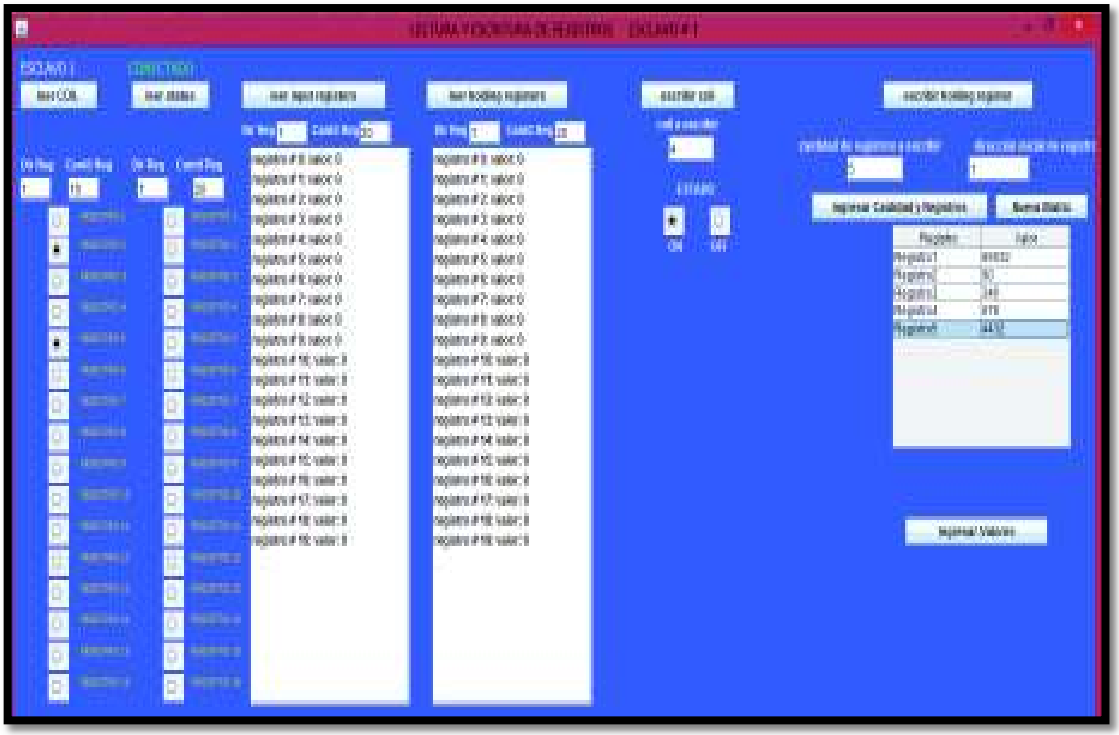


Figura 4.26. Escenario Para Análisis de Tramas

4.3.1 Leer Coil

En este caso el Maestro tiene la dirección IP: 192.168.1.3 y el esclavo la dirección IP: 192.168.1.4 al ejecutar la funcionalidad Leer Coil, el analizador captura tanto la petición como la respuesta desde el esclavo. El analizador Wireshark identifica el protocolo ModBus TCP y tambien el número de función por lo tanto la reconoce como Read Coil. Esto se visualiza en la Figura 4.27.

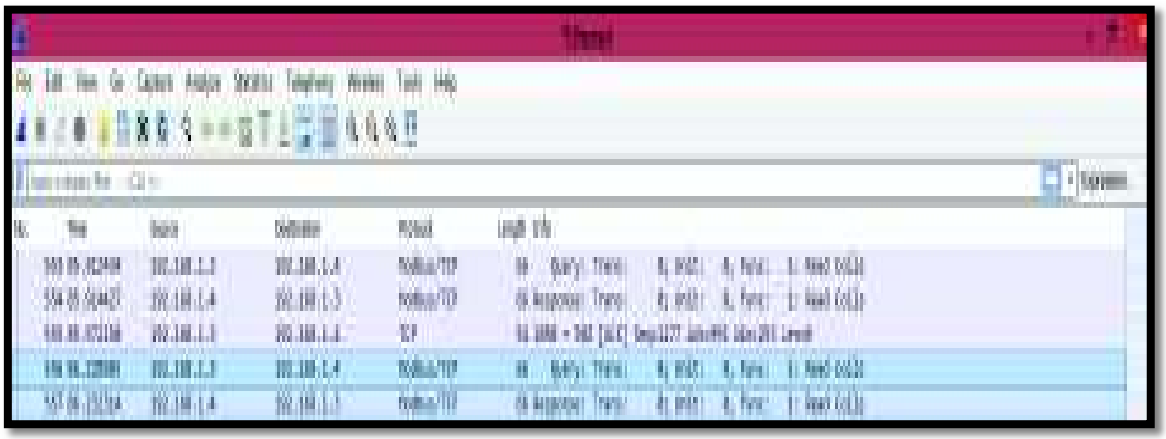


Figura 4.27. Trama Petición/Respuesta Trama Leer Coil

En la captura número 556 se tiene la petición hacia el esclavo, esta trama se encuentra dentro de TCP. La trama ModBus TCP, como se revisó anteriormente, empieza con una cabecera MBAP la cual está conformada por 7 bytes: 2 bytes que identifican la transacción,

2 bytes que identifican el protocolo, 2 bytes de la longitud que continúan después de la cabecera MBAP, 1 byte que identifica el número de la unidad. Como se muestra en la Figura 4.28.

También se puede visualizar la segunda parte de la trama para función Leer Coil (Data), aquí se tiene y byte para identificar la función, 2 bytes para identificar la dirección de registro de inicio y 2 bytes para especificar la cantidad de registros a leer, en este caso los 16 valores a partir del registro 1 (Registro 0 según el mapa de memorias del esclavo), esto se muestra en la Figura 4.29.

En la Figura 4.30 se tiene la trama de respuesta desde el esclavo, aquí se puede visualizar el estado de las 15 variables binarias de las cuales se solicitó el estado.

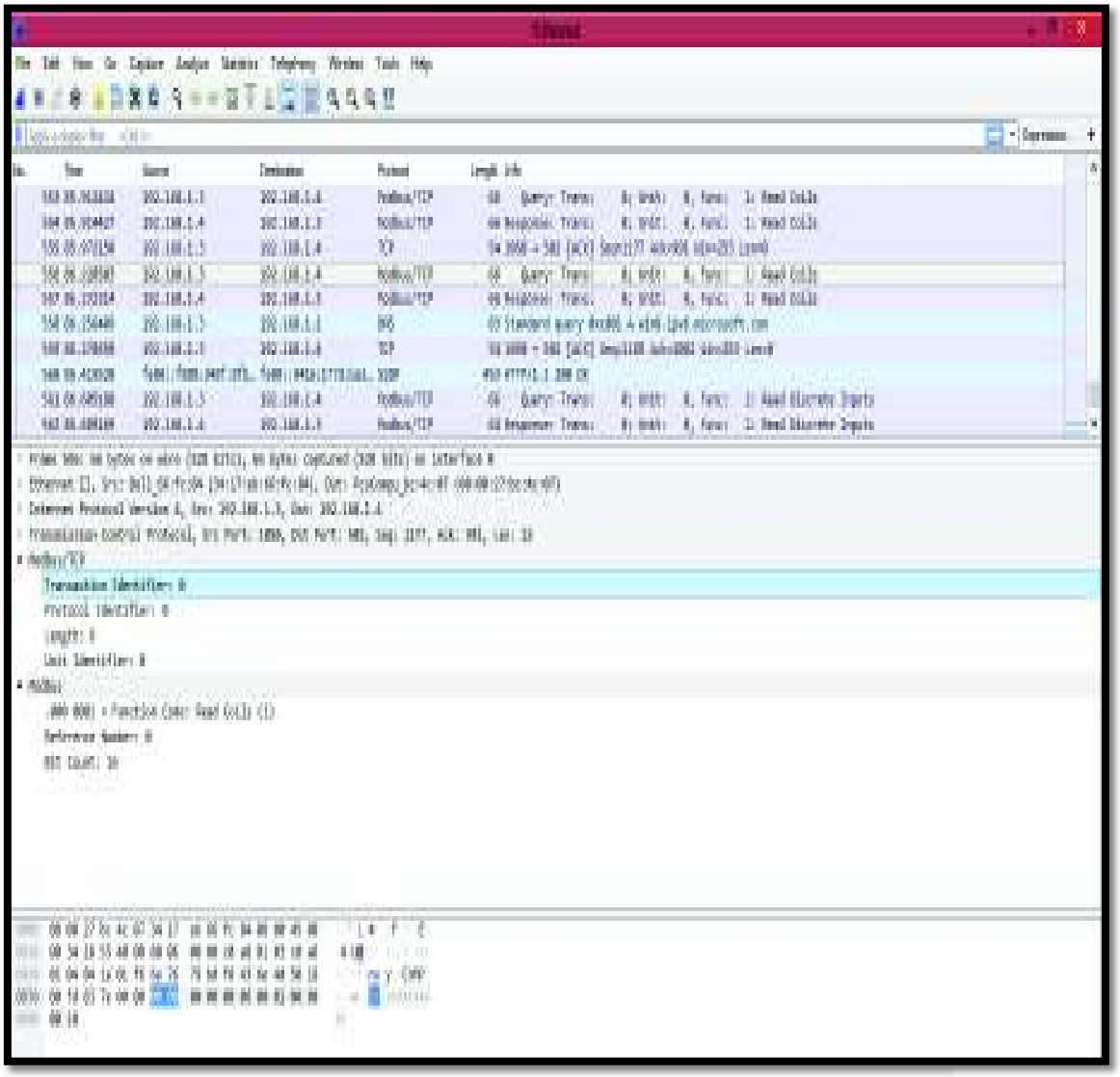


Figura 4.28. Trama Petición Leer Coil Cabecera (MBAP)

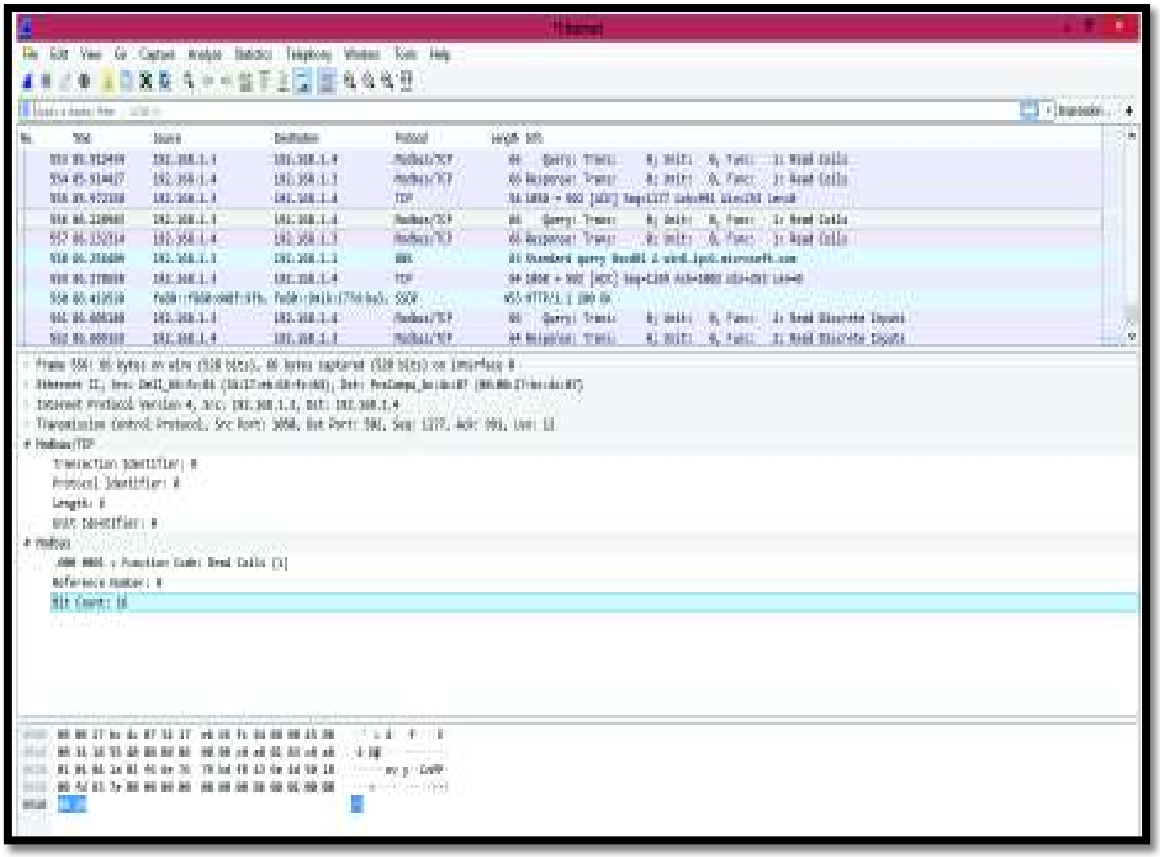


Figura 4.29. Trama Petición Leer Coil (Datos)

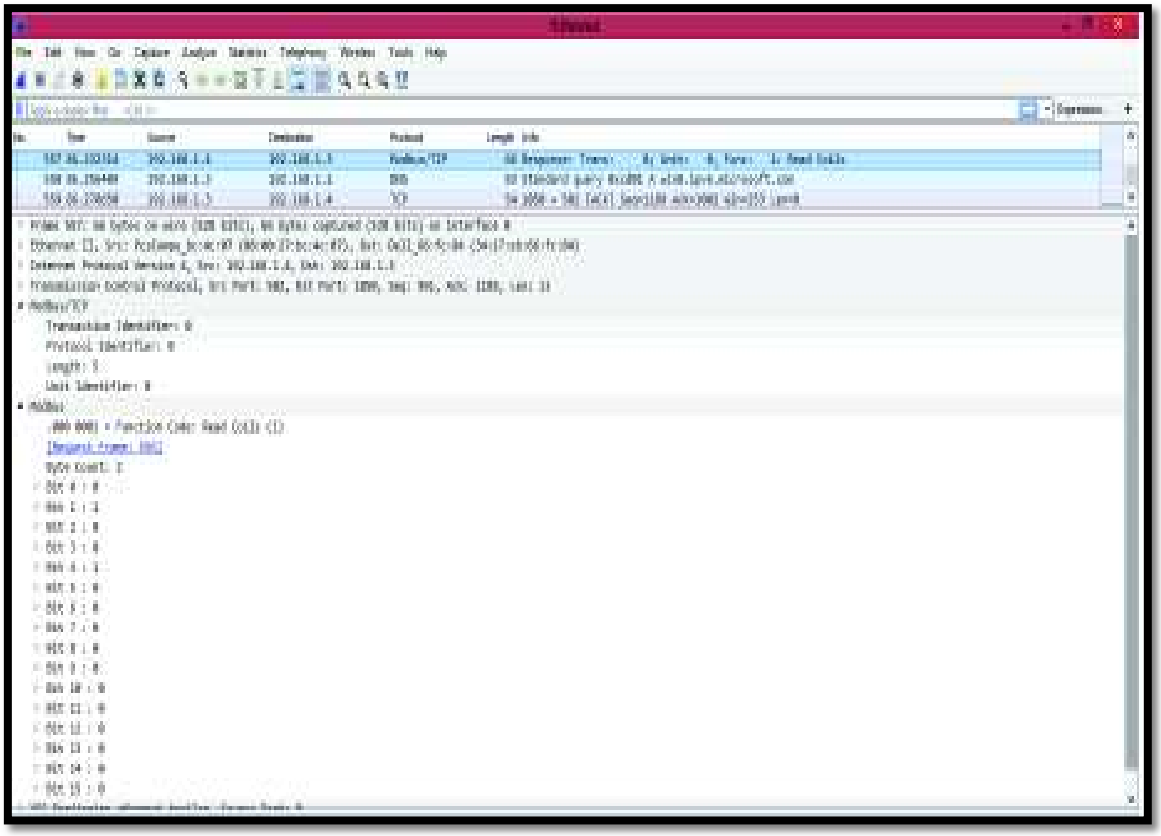


Figura 4.30. Trama Respuesta Leer Coil

4.3.2 Escribir Holding Registers

En la Figura 4.31 se observa la captura número 85 y 86 donde se tiene la trama petición y repuesta al escribir múltiples registros del tipo holding.

Al igual que en la trama anterior se tiene primero la cabecera MBAP de la trama seguido de: el número de función ModBus TCP, dirección del registro de inicio, cantidad de registros a escribir, cantidad de bytes enviados en los registros y el valor para cada uno de los registros a escribir.

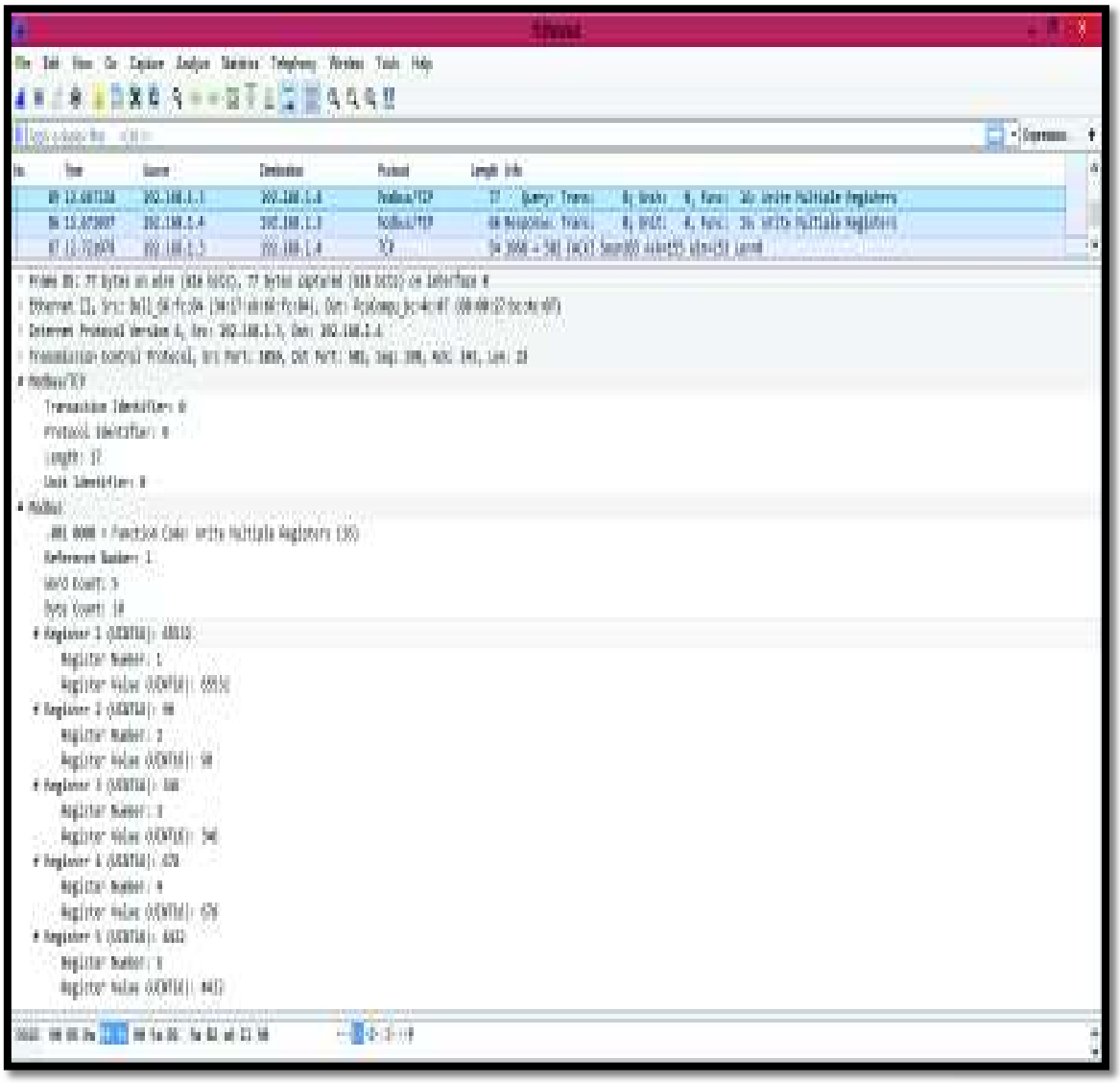


Figura 4.31. Trama Escribir *Holding Registers* Petición

En la respuesta del esclavo se tiene de igual forma la cabecera seguido de la información en la cual se encuentra el número de función, dirección inicial de registros y la cantidad de comandos, tal como se muestra en la Figura 4.32.

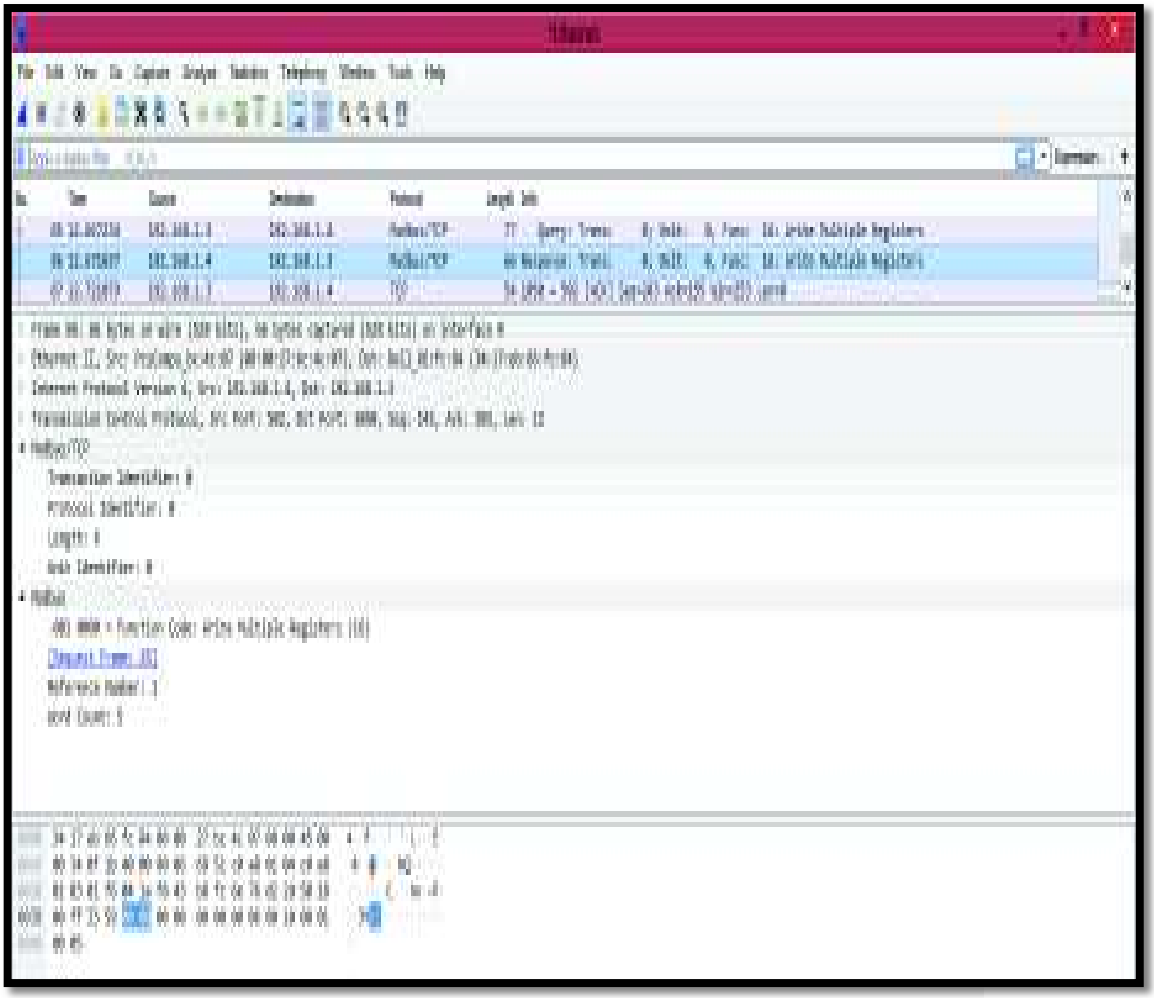


Figura 4.32. Trama Escribir *Holding Registers* Respuesta

4.4 Ajustes al Prototipo de Software

4.4.1 Ajustes Módulo Registro/Acceso Usuarios

En este módulo se tiene 4 ajustes a realizar, 3 de ellos son en relación al diseño de la interfaz gráfica y el cuarto tiene que ver con la creación de un perfil más de usuario.

A continuación se presenta cada uno de estos ajustes.

1. Cambio del texto para cancelar el proceso de *login*, de “CANCELAR” por “LIMPIAR CAMPOS” como se muestra en la Figura 4.33.

Figura 4.33. Cambio de Texto en Botón Cancelar Login

2. Aumentar información sobre él no envio del correo electrónico con las credenciales. Al momento de no poder enviar el correo electrónico, el usuario debe informarse de si la razón es por un correo inexistente o por un problema de conectividad con internet o con el servidor de correo. Los dos casos se muestran en la Figura 4.34.

Figura 4.34. Mensajes de no Envio de Correo Electrónico

3. Adicionar mensajes de confirmación al momento de realizar la funcionalidad de CRUD usuarios. Para cada una de las acciones crear, eliminar o actualizar un usuario se debe mostrar un mensaje de dialogo en el cual se confirme la acción que esta por realizar. Estos mensajes se muestran en la Figura 4.35.

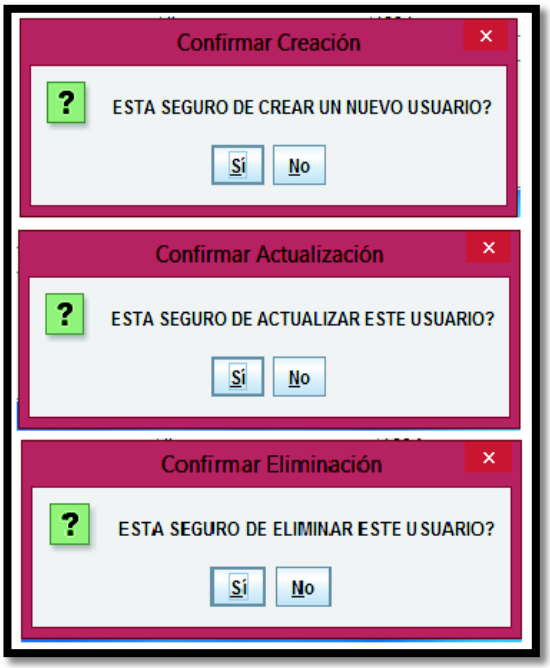


Figura 4.35. Mensajes de Confirmación CRUD Usuarios

4. Crear un perfil adicional del tipo Súper Usuario. Se debería tener un perfil donde sea una única persona que tenga los permisos necesarios para crear, eliminar o actualizar un usuario. El perfil Administrador tendrá acceso pero solo de lectura y revisar credenciales y atributos de los usuarios. Con esto los tres perfiles existentes con su respectivo nivel de acceso quedarían como se muestran en la Figura 4.36.

Id Perfil	Nombre Perfil
1	SuperUsuario
2	Administrador
3	Operador

Figura 4.36. Perfiles de Usuarios

4.4.2 Ajustes Módulo Comunicación/Conexión

En este módulo existe un solo ajuste, el de tener un nombre editable para cada esclavo y que este se replique en las otras interfaces gráficas donde sea necesario identificar a un esclavo como por ejemplo en la interfaz “Diagrama de Red”.

Para esto en la interfaz “Establecer Conexión” se coloca un *JTextField* para cada esclavo, el cual, cada que cambie su propiedad de texto, guardará un nombre nuevo para ese esclavo, y ese nombre se actualizará en las interfaces “Diagrama de Red” y “Esclavo_UNO” o según corresponda.

Por defecto los nombres de los esclavos serán “ESCLAVO 1”, “ESCLAVO 2” y “ESCLAVO 3” como se muestra en la Figura 4.37.

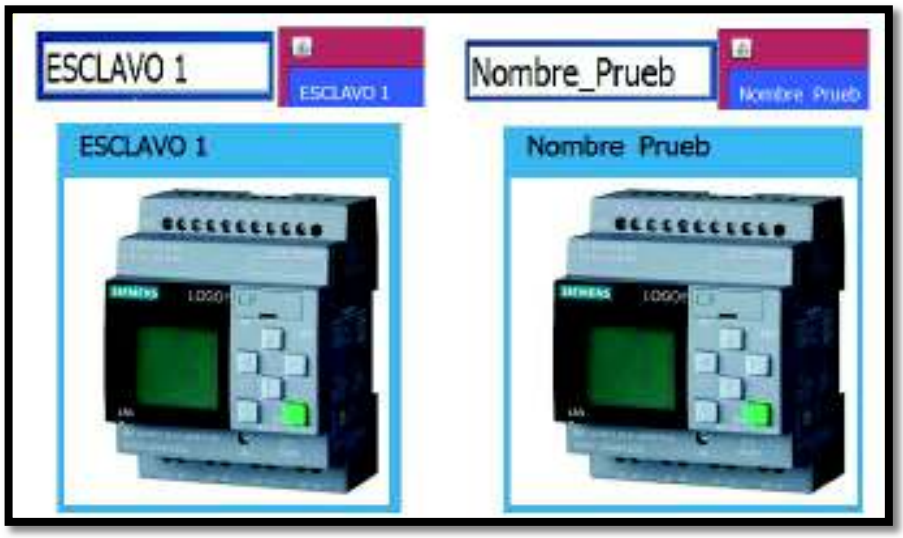


Figura 4.37. Nombre Editable de Esclavos

4.4.3 Ajustes Módulo Lectura/Escritura

En este módulo también existe un solo ajuste. Se trata de cambiar la manera de ingresar los valores para ejecutar la funcionalidad escribir múltiples *Holding Registers*, ya que la forma inicial no es tan intuitiva y además no da lugar a rectificar los valores a escribir en caso de que se ingrese un valor erróneo.

Se llegó a la conclusión de que sería bueno tener una matriz dinámica donde se puedan ingresar todos los valores antes de ejecutar la funcionalidad escribir *Holding Registers*. Con esto se puede chequear los valores antes de ejecutar la funcionalidad. Como parámetros de entrada se pide la cantidad de valores a escribir, esto para crear la matriz, y la dirección del registro de inicio, tal como se muestra en la Figura 4.38.

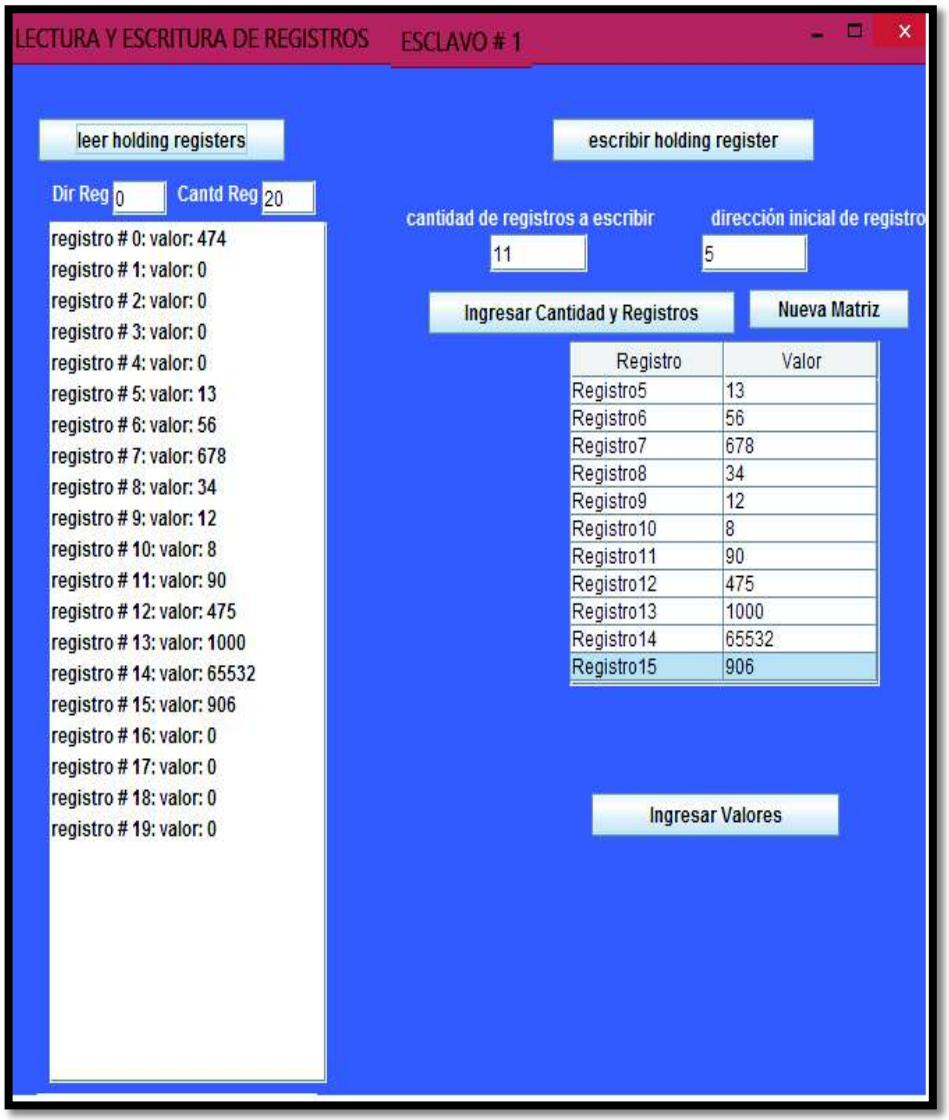


Figura 4.38. Método de Ingreso de Valores

4.4.4 Ajustes Módulo Generar Registros

En este módulo se tiene 3 ajustes, 2 de ellos pequeños detalles en cuanto a la interfaz gráfica y el tercero un cambio adicional para que el prototipo se adapte más a un escenario real. Donde se genere un Historial con más de tres esclavos a la vez.

1. Cambiar el texto del botón “Generar Historial” por el texto “Inicio de Monitoreo” para que el proceso sea más intuitivo. Este cambio se puede visualizar en la Figura 4.39.

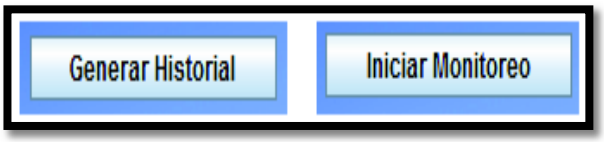


Figura 4.39. Botón Iniciar Monitoreo

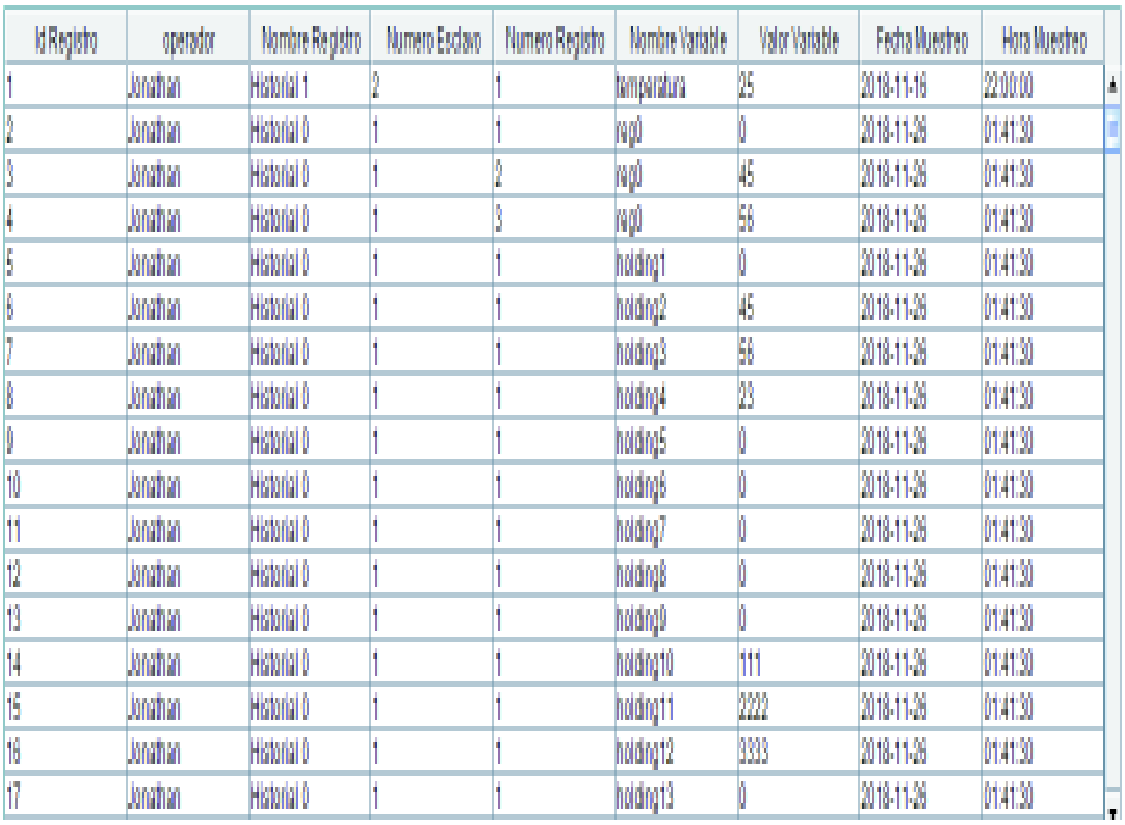
2. Anadir un texto que indique las unidades del tiempo que se ingresa para generar el registro. En este caso el tiempo que se ingresa tanto de muestreo como tiempo total se encuentra en segundos. Este cambio se lo puede observar en la Figura 4.40.

Figura 4.40. Tiempos de Muestreo

3. El tercer cambio implica añadir programación a la funcionalidad “Generar Registros”, esto debido a que hubo observaciones con respecto a guardar únicamente tres registros al crear el historial, ya que esto no se adapta a un escenario real en donde la cantidad de variables a guardar es mucho mayor.

   Por este motivo se agregó un tercer grupo de cajas de texto en las cuales el usuario podrá guardar cualquiera de los 4 tipos de registros, especificando la dirección de inicio y la cantidad de registros; esto se muestra en las Figuras 4.41 y 4.42.

Figura 4.41 Generación Historial Múltiples Registros



Id Registro	operador	Nombre Registro	Numero Esclavo	Numero Registro	Nombre Variable	Valor Variable	Fecha Muestreo	Hora Muestreo
1	Jonathan	Historial 1	2	1	temperatura	25	2018-11-16	22:00:00
2	Jonathan	Historial 0	1	1	reg0	0	2018-11-26	01:41:30
3	Jonathan	Historial 0	1	2	reg0	45	2018-11-26	01:41:30
4	Jonathan	Historial 0	1	3	reg0	56	2018-11-26	01:41:30
5	Jonathan	Historial 0	1	1	nodirg1	0	2018-11-26	01:41:30
6	Jonathan	Historial 0	1	1	nodirg2	45	2018-11-26	01:41:30
7	Jonathan	Historial 0	1	1	nodirg3	56	2018-11-26	01:41:30
8	Jonathan	Historial 0	1	1	nodirg4	23	2018-11-26	01:41:30
9	Jonathan	Historial 0	1	1	nodirg5	0	2018-11-26	01:41:30
10	Jonathan	Historial 0	1	1	nodirg6	0	2018-11-26	01:41:30
11	Jonathan	Historial 0	1	1	nodirg7	0	2018-11-26	01:41:30
12	Jonathan	Historial 0	1	1	nodirg8	0	2018-11-26	01:41:30
13	Jonathan	Historial 0	1	1	nodirg9	0	2018-11-26	01:41:30
14	Jonathan	Historial 0	1	1	nodirg10	111	2018-11-26	01:41:30
15	Jonathan	Historial 0	1	1	nodirg11	2222	2018-11-26	01:41:30
16	Jonathan	Historial 0	1	1	nodirg12	3333	2018-11-26	01:41:30
17	Jonathan	Historial 0	1	1	nodirg13	0	2018-11-26	01:41:30

Figura 4.42. Generación Historial Múltiples Registros

4.4.5 Ajustes Módulo Base de Datos

En este módulo se cumple con todas las funcionalidades, pero existe una observación sobre la implementación de un tipo de filtro sobre la tabla registros ya que es la más extensa.

Para esto se coloca cajas de texto en las cuales se debe ingresar el nombre o una parte del nombre del registro que se desea encontrar. Esto proporciona la opción de filtrar la tabla por:

- Operador
- Nombre Registro
- Número Esclavo
- Nombre Variable
- Fecha Muestreo

Lo anteriormente descrito se muestra en la Figura 4.43.

TABLA DE REGISTROS

Jonathan | Historial | 1 | hol | 2018-11-26

Id Registro	operador	Nombre Registro	Numero Esclavo	Numero Registro	Nombre Variable	Valor Variable	Fecha Muestreo	Hora Muestreo
5	Jonathan	Historial 0	1	1	holding1	0	2018-11-26	01:41:30
6	Jonathan	Historial 0	1	1	holding2	45	2018-11-26	01:41:30
7	Jonathan	Historial 0	1	1	holding3	56	2018-11-26	01:41:30
8	Jonathan	Historial 0	1	1	holding4	23	2018-11-26	01:41:30
9	Jonathan	Historial 0	1	1	holding5	0	2018-11-26	01:41:30
10	Jonathan	Historial 0	1	1	holding6	0	2018-11-26	01:41:30
11	Jonathan	Historial 0	1	1	holding7	0	2018-11-26	01:41:30
12	Jonathan	Historial 0	1	1	holding8	0	2018-11-26	01:41:30
13	Jonathan	Historial 0	1	1	holding9	0	2018-11-26	01:41:30
14	Jonathan	Historial 0	1	1	holding10	111	2018-11-26	01:41:30
15	Jonathan	Historial 0	1	1	holding11	2222	2018-11-26	01:41:30
16	Jonathan	Historial 0	1	1	holding12	3333	2018-11-26	01:41:30
17	Jonathan	Historial 0	1	1	holding13	0	2018-11-26	01:41:30
18	Jonathan	Historial 0	1	1	holding14	0	2018-11-26	01:41:30
19	Jonathan	Historial 0	1	1	holding15	0	2018-11-26	01:41:30
20	Jonathan	Historial 0	1	1	holding16	0	2018-11-26	01:41:30
21	Jonathan	Historial 0	1	1	holding17	0	2018-11-26	01:41:30

Figura 4.43. Filtrado Tabla Registros

4.5 Actualización Final del Tablero Kanban

Una vez terminadas todas las fases del diseño e implementación del Prototipo de Software cada una de las tareas se encuentran en el estado “Tarea Finalizada”, esto quiere decir que todas las otras columnas se encuentran vacías ya que no existe ninguna tarea por desarrollar o en proceso. Así es como Trello, que utiliza la metodología Kanban, ayuda a llevar un proceso de las tareas a realizar como se puede observar en la Figura 4.44.

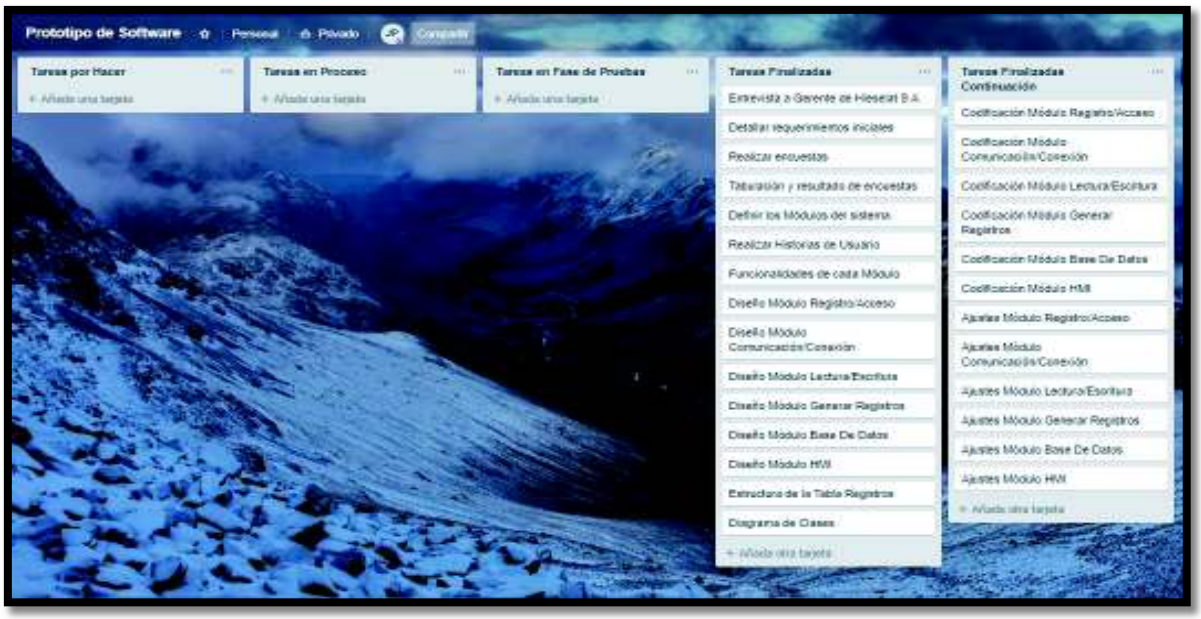


Figura 4.44. Actualización Final del Tablero Kanban

5 CONCLUSIONES Y RECOMENDACIONES

5.1 Conclusiones

- En base al proceso de análisis de requerimientos, diseño e implementación se desarrolló seis módulos: Registro/Acceso Usuarios, Comunicación/Conexión, Lectura/Escritura, Generar Registros, Base de Datos y GUI. Cada uno de estos con sus funcionalidades específicas y con esto se cumplió con el objetivo general que es: Desarrollar un prototipo de software para poder adquirir y gestionar las variables de dispositivos de una red industrial.

- Al analizar los conceptos del protocolo ModBus/TCP se pudo concluir que se puede establecer más de una conexión entre el Maestro y el Esclavo simultáneamente. Esto fue de gran ayuda en el desarrollo del Prototipo de Software, ya que el diseño de algunas funcionalidades implicó realizar más de una petición (*Query*) simultáneamente a los esclavos. Sin embargo debido a la sobrecarga que genera TCP, el monitoreo continuo de datos de un esclavo "*streaming data*" no es muy eficiente e incrementa el tráfico en la red. Esta limitante no se ve reflejada en la implementación de este prototipo ya que el monitoreo continuo solo se dará cuando el usuario quiera generar un registro de las variables y será por un intervalo definido de tiempo.

- Los Esclavos están constantemente a la espera de una petición del Maestro y se limitan a responder a dicha solicitud e inmediatamente se prepara para seguir escuchando por la siguiente petición. En este trabajo de titulación se implementa la arquitectura Maestro/Esclavo, utilizando al prototipo de software como maestro, el cual también puede ser visto como un HMI, desde el cual se envían tramas únicamente del tipo petición (*Query*) y se interpretan las tramas del tipo respuesta (*Response*) que envía el esclavo.

- En el análisis del protocolo se revisó que ModBus tiene definido un número para cada una de las funciones especificadas que utiliza (códigos de operación), este número puede tomar cualquier valor comprendido entre el 0 y el 127. Las funciones utilizadas para la implementación del prototipo de software son 6. Las cuatro primeras son las funciones: *Read Coil Satis* (1), *Read Input Status* (2), *Read Holding Registers* (3), *Read Input Registers* (4). Estas son utilizadas para implementar la funcionalidad de leer los registros de los esclavos. Las dos funciones restantes:

Force Multiple Coils (15) y *Preset Multiple Registers* (16) son empleadas para implementar la funcionalidad de escritura de registros.

- Para la comunicación entre el Maestro y los Esclavos se utiliza el sistema distribuido sockets. Para establecer la conexión y mantener la comunicación se crea un socket con la dirección IP del esclavo y el número de puerto 502 (puerto que utiliza ModBus TCP para la comunicación). También, es necesario crear un *OutputStream*[14] y *BufferedInputStream*[15] asociados al *socket*, ya que sobre estos dos flujos se intercambia bytes de información con el esclavo. Al momento de establecer la conexión con un esclavo se generan tres *sockets* idénticos, esto debido a que es necesario ocupar un canal de comunicación independiente para las funcionalidades: Lectura/Escritura, Diagrama de Red y Generación de Registros porque puede darse un escenario donde tengan que ejecutarse las tres funcionalidades al mismo tiempo y no exista un conflicto al utilizar el mismo canal de comunicación.

- Para la parte del modelado del prototipo de software se desarrollaron tres tipos de diagramas: Diagramas de casos de uso, Diagramas de Actividades y Diagramas de Clase. Para esto se utilizó la herramienta "StartUML", esto fue de gran ayuda al momento de la implementación de los diagramas, ya que se trata de un software dedicado para desarrollo de diagramas y ya tiene bloques predefinidos donde únicamente es necesario arrastrarlos y colocar su relación con otros bloques.

- En cuanto a la codificación de la interfaz gráfica HMI se puede concluir que la parte gráfica (barras de temperatura, y nivel del tanque) juegan un papel importante en el tema de la visualización de datos. Por ejemplo, el mostrar el valor medido de temperatura con la barra ayudó a visualizar si el valor fue creciendo gradualmente o se generó un pico. Ya que guarda estados anteriores para presentarlos en forma de historiales. El mostrar el nivel del tanque de acuerdo al valor resultante por el caudal de entrada, ayuda a tomar la decisión de aumentar o disminuir el caudal de salida.

- Se realizó una fase de pruebas y posterior a eso una fase de ajustes, esto ayudó a evaluar a cada uno de los módulos y todo el prototipo de software en conjunto. Para realizar estas etapas fue necesario desarrollar una matriz de pruebas en base a las

[14] *OutputStream*, Esta clase abstracta es la superclase de todas las clases que representan un flujo de salida de bytes[29].
[15] *BufferedInputStream,* agrega funcionalidad a otro flujo de entrada, es decir, la capacidad de almacenar en búfer la entrada y admitir los métodos de mark y reset. Cuando se crea el BufferedInputStream se crea una matriz de búfer interna[30].

funcionalidades planteadas en la etapa de diseño. Esta matriz debe ser implementada siempre, ya que de lo contrario las pruebas pueden tornarse subjetivas y será difícil establecer parámetros medibles y objetivos en relación a si el prototipo de software cumple o no con el alcance.

- La información obtenida en la tabla registros de las base a de datos, tiene la propiedad de ser exportada a Excel y allí se pueden analizar los datos obtenidos, visualizando de manera gráfica los resultados y ayudando a identificar el comportamiento de los mismos. Es decir, visualizar valores máximos, mínimos y valores promedios, y con ello implementar cambios correctivos o preventivos en relación a un proceso. En el Anexo V se muestra un ejemplo de análisis de datos de un historial generado en el intervalo de dos horas donde se lee el valor de temperatura, nivel del tanque y caudal de entrada.

5.2 Recomendaciones

- Se recomienda que en las fases del plan de trabajo se añadan siempre un periodo de estabilización del sistema, sea este un mínimo de dos semanas o en el mejor de los casos de un mes. Esto debido a que la fase de pruebas, si bien ayuda evaluar las funcionalidades puntuales del prototipo, siempre va a existir escenarios que no fueron tomados en cuenta o que el programador desconoce. Pero al existir este periodo de estabilización del sistema se pueden replantear en forma de funcionalidades adicionales e implementarlas en el prototipo.

- Una recomendación que surgió al momento de realizar la fase de pruebas es el implementar un sistema de logs en el cual el prototipo registre cada uno de las acciones que realiza un usuario desde el momento en que inicia sesión, y que esto se almacene en la base de datos. Con esto se podrá generar un mejor control en cuanto a las acciones que realiza cada usuario.

- El trabajo de titulación realizado es un PROTOTIPO, lo cual implica que se debería desarrollar nuevas versiones hasta llegar a tener una primera versión final del software. En las cuales se utilicen las funcionalidades ya implementadas como: Establecimiento de la Conexión, Lectura/Escritura de Registros y Creación de Históricos; para desarrollar nuevas funcionalidades de mayor complejidad como por ejemplo:

  - Una base de datos con los mapas de memorias de algunos modelos de PLC

- Sistemas de control más avanzados por ejemplo: un Controlador PID (Controlador Proporcional Integrativo y Derivativo).
- Desarrollo de una HMI menos estructurada, donde los objetos gráficos se encuentren ya predefinidos y solos se carguen a un Jframe donde se asocie sus parámetros con los registros de los esclavos, es decir contar con una biblioteca de objetos gráficos.

- Las versiones futuras del prototipo de software deberían ser desarrolladas por un estudiante de Electrónica y Redes de Información y otro de Electrónica y Control ya que al trabajar juntos se tendrían criterios más específicos tanto en la fase de diseño como en la fase de implementación.

6 REFERENCIAS BIBLIOGRÁFICAS

[1] National Instruments Corporation, «LabVIEW - National Instruments». [En línea]. Disponible en: http://www.ni.com/es-cr/shop/select/labview?edition=professional. [Accedido: 01-dic-2018].

[2] Modbus Organization, «Modbus FAQ», *Modbus*, 2008-2005. [En línea]. Disponible en: http://www.modbus.org/faq.php. [Accedido: 07-feb-2018].

[3] ANDRES FELIPE RUIZ OLAYA, «IMPLEMENTACION DE UNA RED MODBUS/TCP». 2002.

[4] A. F. Ruiz Olaya, A. Barandica, y F. Guerrero, *Implementación de una red MODBUS/TCP*. Universidad del Valle (Colombia), 2011.

[5] «Presentacion_modbus.pdf».

[6] I. S. M. G. ING. JOHN ALEXANDER PEÑALOZA CALDERÓN, «RED DE PLC'S Y VARIADORES DE VELOCIDAD CON PROTOCOLOS ETHERNET Y MODBUS». 04-may-2009.

[7] J. C. Q. Q. E. Flores Garcia, «MODBUS RTU PROTOCOL IMPLEMENTATION in the COMMUNICATION BETWEEN a PAC AND A VARIABLE SPEED DRIVE for the AUTOMATIC CONTROL of ELECTRIC MOTORS». [En línea]. Disponible en: https://www.uaeh.edu.mx/scige/boletin/tizayuca/n5/p7.html. [Accedido: 28-ene-2019].

[8] NATIONAL INSTRUMENTS, «Información Detallada sobre el Protocolo Modbus - National Instruments», *NATIONAL INSTRUMENTS*. [En línea]. Disponible en: http://www.ni.com/white-paper/52134/es/. [Accedido: 07-feb-2018].

[9] ANDRÉS FELIPE HURTADO BANGUERO, «Protocolo Modbus», *EEYMUC*, 02-feb-2018. .

[10] OMROM, «MODBUS TCP CS1W-ETN21 CJ1W-ETN21». 28-feb-2018.

[11] MODBUS ORG, «MODBUS APPLICATION PROTOCOL SPECIFICATION V1.1b3». 12-abr-2012.

[12] «¿Qué es SCADA? - WonderWare», *Wonderware Software - Powering the Industrial World*. .

[13] J. Bartolomé, «Protocolo MODBUS», ene-2011. [En línea]. Disponible en: http://www.tolaemon.com/docs/modbus.htm. [Accedido: 08-feb-2018].

[14] Nereida Deioc, «Sockets», *Herramientas y Lenguajes de Programación*, may-2006. [En línea]. Disponible en: http://nereida.deioc.ull.es/~cleon/doctorado/doc06/doc06/html/node9.html. [Accedido: 13-mar-2018].

[15] «java.net (Java Platform SE 8)». [En línea]. Disponible en: https://docs.oracle.com/javase/8/docs/api/index.html?java/net/package-summary.html. [Accedido: 30-ene-2019].

[16] Wonderware, «Wonderware InTouch HMI Software - Características - WonderWare», *Wonderware Software - Powering the Industrial World*, 2018. [En línea]. Disponible en: http://www.wonderware.es/hmi-scada/intouch/caracteristicas/. [Accedido: 01-abr-2018].

[17] «Desarrollo-Ágil-con-Kanban.pdf».

[18] H. Kniberg y M. Skarin, *Kanban and Scrum: making the most of both*. s. l.: C4Media, 2010.

[19] Daniel Matesa, «▷ Trello. Qué es, Para Qué sirve y Cómo Funciona», *ENO Expertos Negocios Online*, 13-oct-2017. [En línea]. Disponible en: https://www.expertosnegociosonline.com/que-es-trello-para-que-sirve/. [Accedido: 28-ene-2019].

[20] D. R. Cardozzo, *Desarrollo de Software: Requisitos, Estimaciones y Análisis. 2 Edición*. IT Campus Academy, 2016.

[21] «WikiUML - Diagrama de Actividad». [En línea]. Disponible en: https://wikiuml.wikispaces.com/Diagrama+de+Actividad. [Accedido: 30-may-2018].

[22] V. F. Alarcón, *Desarrollo de sistemas de información: una metodología basada en el modelado*. Univ. Politèc. de Catalunya, 2010.

[23] «StarUML - Open Source UML Tool». [En línea]. Disponible en: http://www.methodsandtools.com/tools/staruml.php. [Accedido: 28-ene-2019].

[24] «Introduction - StarUML documentation». [En línea]. Disponible en: https://docs.staruml.io/. [Accedido: 28-ene-2019].

[25] J. Wren, «WindowBuilder | The Eclipse Foundation», *The Eclipse Foundation*. [En línea]. Disponible en: https://www.eclipse.org/windowbuilder/. [Accedido: 28-ene-2019].

[26] «Eclipse WindowBuilder - Creating user interfaces - Tutorial». [En línea]. Disponible en: http://www.vogella.com/tutorials/EclipseWindowBuilder/article.html. [Accedido: 28-ene-2019].

[27] ORACLE, «MySQL :: MySQL Workbench», 2019. [En línea]. Disponible en: https://www.mysql.com/products/workbench/. [Accedido: 28-ene-2019].

[28] ORACLE, «MySQL :: MySQL Connector/J 5.1 Developer Guide :: 6.1 Connecting to MySQL Using the JDBC DriverManager Interface». [En línea]. Disponible en: https://dev.mysql.com/doc/connector-j/5.1/en/connector-j-usagenotes-connect-drivermanager.html. [Accedido: 28-ene-2019].

[29] ORACLE, «OutputStream (Java Platform SE 7)». [En línea]. Disponible en: https://docs.oracle.com/javase/7/docs/api/java/io/OutputStream.html. [Accedido: 28-ene-2019].

[30] ORACLE, «BufferedInputStream (Java Platform SE 7)». [En línea]. Disponible en: https://docs.oracle.com/javase/7/docs/api/java/io/BufferedInputStream.html. [Accedido: 28-ene-2019].

7 ANEXOS

ANEXO A. Encuestas

Las encuestas realizadas se encuentran en el CD adjunto a este documento.

ANEXO B. Conjunto de Historias de Usuario

Las encuestas realizadas se encuentran en el CD adjunto a este documento.

ANEXO C. Código Fuente

El código fuente generado se encuentra en el CD adjunto a este documento.

ANEXO D. Pruebas

Las pruebas realizadas al Prototipo de Software se encuentran en el CD adjunto a este documento.

ANEXO E. Gráficos

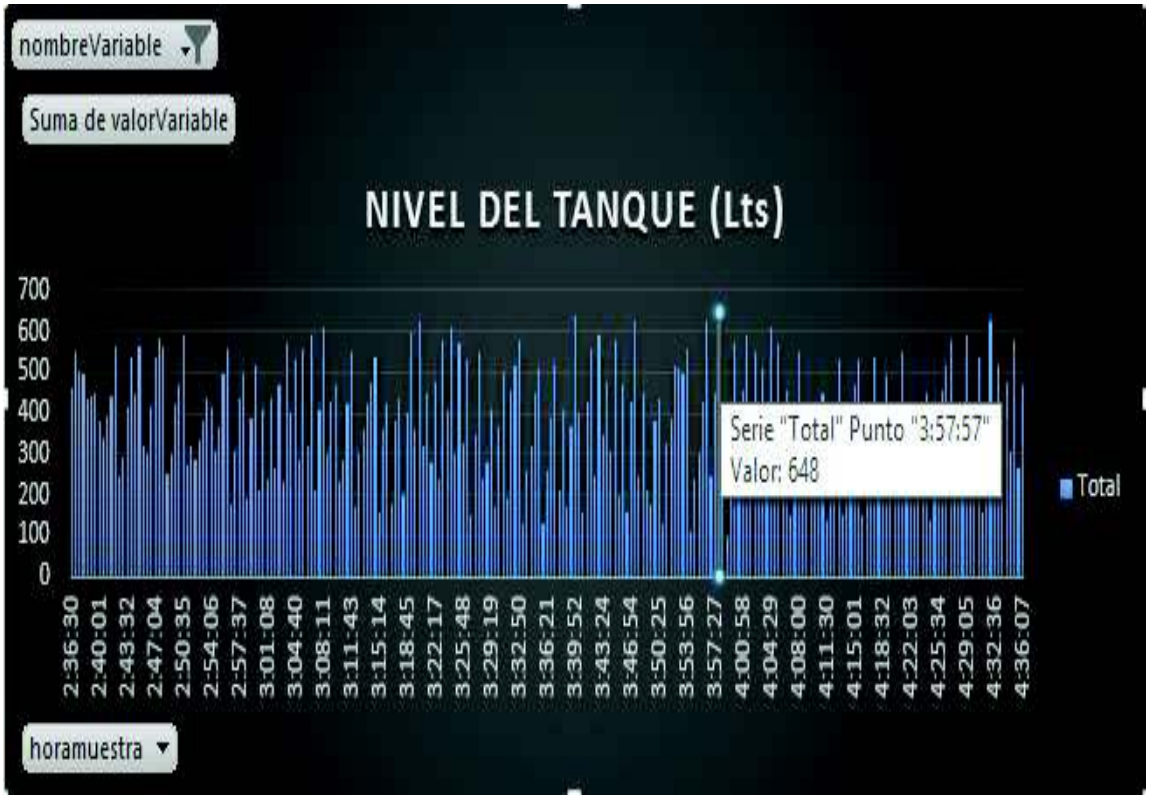


Figura E.1. Análisis Valores Nivel del Tanque

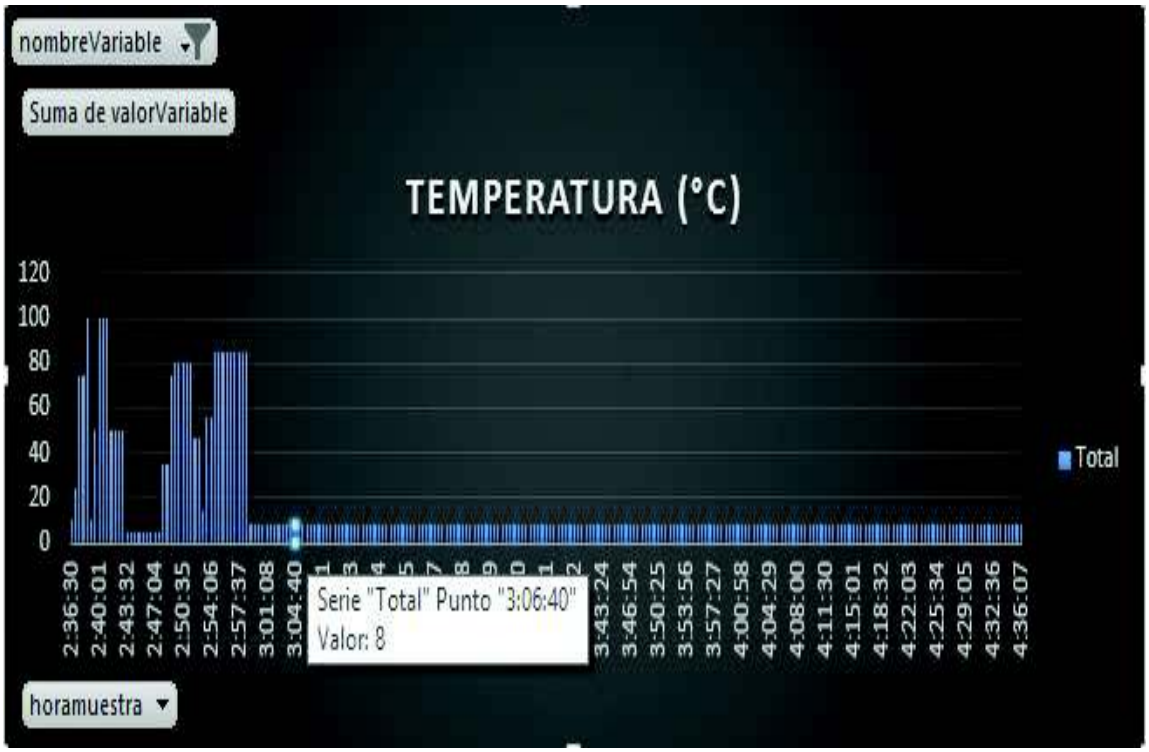


Figura E.2. Análisis Valores Temperatura

Figura E.3. Análisis Valores Caudal de Entrada

ANEXO F. Certificado de Conformidad

HIESELAT S.A.
HITO EN SOLUCIONES ELÉCTRICAS Y DE AUTOMATIZACIÓN

PARA: **ESCUELA POLITÉCNICA NACIONAL**

FECHA: **6 DE DICIEMBRE DEL 2018**

ASUNTO: **CERTIFICADO DE CONFORMIDAD DEL PROYECTO DE TITULACIÓN DEL SR. JONATHAN PINZÓN**

Yo, Daniel Vicente Rosales Alvarado como Presidente de la empresa HIESELAT S.A, entidad auspiciante del trabajo de titulación: **"DESARROLLO DE UN PROTOTIPO DE SOFTWARE PARA ADQUISICIÓN Y GESTIÓN DE DATOS DE EQUIPOS EN REDES INDUSTRIALES, PARA LA EMPRESA HIESELAT S.A"**

CERTIFICO:

Que Jonathan Eduardo Pinzón Rueda estudiante de la facultad de Ingeniería Eléctrica Y Electrónica, desarrolló el trabajo de titulación referido. Cumpliendo con el alcance definido en el plan de titulación, y realizando cada una de las etapas para el Prototipo de Software:

- Diseño
- Implementación
- Pruebas
- Ajustes

Atentamente,

Ing. Daniel Rosales A.

HIESELAT S.A.

ANEXO G. Manuales

Los manuales de Uusraio y de Instalación se encuentran en el CD adjunto a este documento.

ORDEN DE EMPASTADO