*r*

***Web Documentation***

*fdx-hs-newGen*

Logo

Description automatically generated

Contenido

[Introducción 2](#_Toc146198977)

[Propósito 2](#_Toc146198978)

[Glosario 2](#_Toc146198979)

[Alcance del proyecto 3](#_Toc146198980)

[Descripción General 4](#_Toc146198981)

[Perspectiva del producto 4](#_Toc146198982)

[Requerimientos Funcionales 5](#_Toc146198983)

[Clases de usuarios y características 8](#_Toc146198984)

[Requerimientos futuros 8](#_Toc146198985)

[Requerimientos no funcionales 8](#_Toc146198986)

[Diseño de la UI 8](#_Toc146198987)

[Tecnologías 13](#_Toc146198988)

[Dependencias y Suposiciones 14](#_Toc146198989)

[Seguridad 15](#_Toc146198990)

[Flujo de datos 16](#_Toc146198991)

[Comandos 17](#_Toc146198992)

[Estructura de datos de los comandos 19](#_Toc146198993)

[Sensor\_config: 19](#_Toc146198994)

[Alarm\_config: 20](#_Toc146198995)

[Labels: 20](#_Toc146198996)

[Modbus\_config: 21](#_Toc146198997)

[Modbus\_table: 21](#_Toc146198998)

# Introducción

## Propósito

El objetivo es diseñar una página interfaz web embebida para un sistema de monitoreo de temperatura en tiempo real, debe tener la capacidad de configurar las etiquetas para identificar la ubicación física de los sensores, configuración de alarmas según el umbral establecido, configuración de campos para identificar técnicamente el equipo, dicho sistema de monitoreo está conformado por tres (3) dispositivos principalmente:

* Sensores de temperatura RFID, los cuales son posicionados en los puntos de interés que el cliente desea hacerle seguimiento a la temperatura.
* Antenas, las cuales se encargan de monitorear y alimentar a los sensores RFID.
* **Registrador de temperatura (FDX-HS)**, es el encargado de recibir e interpretar los parámetros medidos por los sensores de temperatura y es el dispositivo en **donde se debe rediseñar e implementar la página web embebida debido a un cambio de tecnología,** como información adicional este dispositivo cuenta con la capacidad de comunicación por modbus, slots para instalar dos antenas y administrar 50 sensores de temperatura.

## Glosario

FDX: Función de X Ingeniería en desarrollo

RFID: “Radio Frecuency Identification”, es un sistema de identificación entre objetos que utiliza radiofrecuencia. Esta tecnología permite que los datos que se encuentran codificados en etiquetas RFID inteligentes sean capturados por un lector RFID mediante ondas de radio.

Web embebida: se refiere a la capacidad de mostrar contenido web dentro de una aplicación o software que se ejecuta en un dispositivo específico, lo que permite a los usuarios acceder a información de la web sin tener que abrir un navegador web por separado.

FDX-HS: Es un equipo que forma parte del sistema de monitoreo comercializado por FDX encargado de leer e interpretar mediante RFID sensores de temperaturas.

## Alcance del proyecto

La empresa FDX va a cambiar de tecnología en los equipos FDX-HS por lo que se debe realizar una actualización a la web embebida de dicho equipo, la cual debe contar con funcionalidades similares a la versión anterior, excluyendo algunas que ya no van a ser requeridas y agregando nuevas, siempre buscando brindarle a los usuarios una mejor experiencia y usabilidad, la nueva web debe permitir configurar el equipo de una manera fácil e intuitiva y que el usuario pueda realizar un monitoreo una vez finalizada la instalación del sistema.

## Elaborado por:

|  |  |  |  |
| --- | --- | --- | --- |
| Descripción | Elaborado por | Aprobado por | Fecha |
| Web documentation versión 2.0 | Luis Flores |  | 25/09/23 |

# Descripción General

## Perspectiva del producto

Mediante el módulo wifi que cuenta el equipo FDX-HS debe permitir al usuario acceder a una interfaz web, la cual será usada con los siguientes propósitos:

* Configuración técnica del dispositivo, para poder establecer los campos tales como: serial del equipo, dirección modbus, baudrate, definir el bit de paridad, señalización de las alarmas, definir las etiquetas “labels” que van a representar de manera lógica al equipo que se quiere monitorear y de esta forma saber dónde están instalados los sensores (ubicación, equipo, fase). Nota: Para la configuración del campo serial del equipo debe ser protegida con una contraseña diferente que solo persona de FDX tengan acceso.
* Configuración de sensores, al instalar el equipo se deben definir que sensores van a ser monitoreados por el registrador de temperatura, asignado el sensor una ubicación según las etiquetas de las posiciones técnicas cargadas en la configuración del equipo.
* Configuración de las alarmas, el cliente debe tener la posibilidad de definir una serie de alarmas indicando a que posición técnica representa, estas pueden ser de forma lógica que no activen ningún relé o que, si lo hagan, están definidos 3 algoritmos de activación de la alarma:
  + Valor absoluto: La alarma se activa cuando la temperatura de alguno o de los sensores supera el valor del set point. Se mantiene activa hasta que todos los sensores tengan una temperatura inferior al set point.
  + Desbalance: La alarma se activa cuando la diferencia entre la temperatura de algún sensor y el promedio de todos los sensores supera al valor del set point. Se mantiene activa hasta que la diferencia entre la temperatura de cada uno de los sensores y el promedio de todos los sensores sea inferior al set point.
  + Dispersión: La alarma se activa cuando la diferencia entre la temperatura del sensor con mayor temperatura y la temperatura del sensor con menor temperatura, supera al valor del set point. Se mantiene activa hasta que la diferencia entre la temperatura del sensor con mayor temperatura y la temperatura del sensor con menor temperatura, es inferior al valor del set point.
* Monitoreo de los sensores, el cliente debe poder de ver en tiempo real la temperatura registrada por cada sensor configurado, reflejando si dicho sensor tiene buena calidad de señal, tiempo transcurrido desde el último dato recibo, entre otros.
* Monitoreo de las alarmas, el usuario debe tener la capacidad de poder verificar el estado de las alarmas pudiendo identificar rápidamente cuales están activas y cuáles no.
* Obtención de la tabla modbus, debido a que el dispositivo cuenta con la capacidad de comunicación mediante el protocolo modbus el cliente debe tener la capacidad de acceder a la tabla de direcciones (auto generada) para poder configurar y operar el dispositivo mediante modbus.

## Clases de usuarios y características

|  |  |
| --- | --- |
| Usuario | características |
| Visualizador | Es aquel usuario que tiene acceso físico al equipo puede encender el wifi del mismo y verificar el correcto funcionamiento, accediendo a la vista de monitoreo de sensores y monitoreo de alarmas configuradas. |
| Administrador | Es aquel usuario que tiene acceso al equipo y tiene los permisos para configurarlo, permitiendo establecer las alarmas, agregar o eliminar sensores, entre otros, tiene acceso a la vista de configuración del sistema. |
| Super administrador | Es el usuario definido por el personal FDX que le permite acceso completo a las vistas configuradas en el equipo, así como también establecer valores de configuración de fábrica. |

## Requerimientos Funcionales

1. Barra de navegación: que contenga un menú conformado por los enlaces para redirigir al usuario a las diferentes vistas.
   1. Detalles
      1. Debe contener el enlace para cada vista definida en el sistema.
   2. Validaciones
      1. Se debe validar que este disponible en todas las vistas que conforman la web.
2. Vista monitoreo de sensores (página de inicio).
   1. Detalles:
      1. Función de listar sensores configurados: una tabla que contenga los siguientes campos de la estructura sensor\_config id, equipment, location, position; de la estructura new\_sensor\_data temp, std\_dev, quality, time\_stamp y de la estructura alarm\_states alarm.
      2. Función de resaltar fila del sensor: debe permitir al usuario identificar de manera fácil y rápida si un sensor tiene problemas de señal (“Out of service”) o si está presente alguna alarma (“1”),
      3. Leyenda de colores: debe contar con una leyenda que describa brevemente al usuario que significa cada color.
   2. Validaciones
      1. Se debe validar que esta es la vista que se muestre por defecto al conectarse al equipo.
      2. Se debe validar que al entrar a esta vista se envíe el comando para iniciar modo normal de operación.
3. Vista configuración del sistema
   1. Vista de configuración del equipo
      1. Detalles:
         1. Debe poder modificar los campos de la estructura hs\_confing serial(por definir si va a ser un campo editable o solo de visualización), dir\_modbus (definido por el usuario), baudrate (lista desplegable), bit\_paridad (debe contar con un tooltip con la información de cada modo, lista desplegable par, impar y sin paridad).
         2. Botón de guardado: debe contar con un botón para que el usuario pueda guardar los datos configurados.
         3. Debe enviar un mensaje que la configuración fue guardada correctamente.
      2. Validaciones
         1. Validar que los datos se encuentran almacenados correctamente en la base de datos.
         2. Validar que se envía el comando detener la consulta de los sensores al presionar el botón guardar.
   2. Vista de configuración de sensores (modo Discovery).
      1. Detalles:
         1. Debe contar con una tabla de los sensores instalados en el equipo que permita visualizar los siguientes campos de la estructura new\_sensor\_data id, temp, rssi, calidad; permita configurar los siguientes campos de la estructura sensor\_config equipment, location, position.
         2. Debe contar con una tabla de los sensores no instalados y que el equipo es capaz de consultar que permita visualizar los siguientes campos de la estructura new\_sensor\_data id, temp, rssi, calidad; permita configurar los siguientes campos de la estructura sensor\_config equipment, location, position.
         3. Debe contar con un botón de iniciar: para enviar el comando de comenzar la consulta de los sensores al alcance de las antenas instaladas en el equipo.
         4. Debe contar con un botón de guardado: para enviar el comando de guardar que sensores va a monitorear el equipo.
         5. Debe contar con un botón detener: para enviar el comando que el equipo deje de consultar a los sensores.
      2. Validaciones:
         1. Si no cuenta con sensores instalados debe mostrar la tabla pero con campos vacíos.
         2. Se debe validar que los sensores aparezcan en una sola de las tablas.
         3. Se debe validar que la cantidad máxima de sensores a instalar en un equipo sea 50.
   3. Vista de configuración de etiquetas.
      1. Detalles
         1. Función de agregar etiquetas: debe permitir al usuario incluir nombres a los campos de la estructura labels location, equipment y position.
         2. Función editar etiquetas: debe permitir al usuario editar nombres de los campos de la estructura labels location, equipment y position.
         3. Función eliminar etiquetas: debe permitir al usuario eliminar nombres a los campos de la estructura labels location, equipment y position.
         4. Botón de guardar: debe permitir al usuario guardar los cambios.
         5. Debe enviar un mensaje que los datos fueron guardados correctamente.
      2. Validaciones
         1. Se debe validar que los datos ingresados por el cliente fueron guardados correctamente en la base de datos.
         2. Se debe validar que las etiquetas por defecto no pueden ser editadas.
   4. Vista de configuración de alarmas
      1. Detalles:
         1. Debe contener una tabla de las alarmas permitiendo visualizar los siguientes campos de la estructura alarm\_config: name, alarm\_type, set\_point, relay\_flag, fields.
         2. Botón editar, cada fila del listado debe contener un botón editar que va a redirigir al usuario a la vista de detalle de la alarma.
      2. Validaciones
         1. Se debe validar que solo son 20 alarmas configurables.
   5. Vista de detalle de alarma: ventana emergente que le va a permitir al usuario definir los campos editables de una alarma.
      1. Detalle
         1. Debe tener un formulario con las siguientes entradas de datos de la estructura alarm\_config name, set\_point, alarm\_type, relay\_flag, fields.
         2. Debe contener un botón aceptar, envía el comando para guardar los valores de la alarma.
         3. Debe contener un botón cancelar.
4. Vista de monitoreo de alarmas.
   1. Debe mostrar una tabla con las alarmas configuradas en el equipo con los siguientes datos de la estructura alarm\_config name, set\_point, alarm\_type, relay\_flag, fields.
   2. Debe contener un botón de exportar.
5. Vista obtención del mapa modbuss
   1. Detalles
      1. Función de generar tabla modbus: debe permitir al usuario acceder a una tabla con los registros y descripción de acuerde a los parámetros configurados.
      2. Botón imprimir: debe permitir la tabla para entregar al responsable del sistema SCADA o sistema de monitoreo.

## Requerimientos futuros

1. Botón para exportar los datos en la pantalla de inicio: donde el usuario pueda bajar un reporte de la tabla.

## Requerimientos no funcionales

1. Se necesita que la página cuente con una única uri para comunicarse por el websocket, de manera que las diferentes pantallas van a ser vistas.

# Diseño de la UI

Las siguientes imágenes son a manera de ejemplo de la versión anterior

* Pantalla de configuración del sistema:

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

* Pantalla de configuración de sensores de temperatura

Interfaz de usuario gráfica, Tabla

Descripción generada automáticamente

**Nota:** en la nueva versión se necesita que esta pantalla este conformada por dos tablas para poder diferenciar entre sensores ya configurados y nuevos sensores detectados, para los casos en que clientes con sistemas ya instalados quieran agregar más sensores.

* Pantalla de configuración de alarmas.

Tabla

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

* Pantalla de monitoreo de sensores de temperatura

Tabla

Descripción generada automáticamente con confianza media

* Pantalla monitoreo de alarmas

Tabla

Descripción generada automáticamente

* Obtención de la tabla modbus

Tabla

Descripción generada automáticamente

* Pantalla de información para configuración de protocolo modbus.

Tabla

Descripción generada automáticamente

Tabla

Descripción generada automáticamente

# Tecnologías

* El frontend debe ser implementado en el framework VUEJS en la versión 3
* El backend es un sistema embebido con capacidades limitadas y la interacción con el frontend es mediante WebSocket y la estructura de datos es en formato JSON específico, que representan “comandos” que son enviados desde la web (frontend) y ejecutados por el sistema, así como otros que son “respuestas” o datos enviados desde el Sistema (backend) hacia la Web.

# Dependencias y Suposiciones

1. Ya se cuenta con un diseño de base de datos para soportar los comandos establecidos en el presente documento.

# Seguridad

* No se requiere implementar protocolos de seguridad, la seguridad se va a manejar desde el backend.

# Flujo de datos

Las “acciones” así como cualquier comunicación con la web embebida se realiza a través de un WebSocket, y mediante strings estructurados como comandos JSON

Ejemplo de flujo de datos para “Labels”:

Diagrama

Descripción generada automáticamente

# Comandos

Lista de comandos implementados en el backend:

|  |  |  |  |
| --- | --- | --- | --- |
| **cmd** | **arg** | **data** | **Descripción** |
| sensor\_config | get | (\*) | * Este comando actúa como una señal, indica al sistema que responda con la configuración almacenada del sensor. * La respuesta tiene la misma estructura, pero con una matriz de JSON en el campo "data".   Para más detalle consulte la estructura de datos de sensor\_config |
| sensor\_config | set | (\*\*) | * Hace que el sistema guarde la configuración que se está enviando en "data", para lo cual el formato debe ser correcto.   Para más detalle consulte la estructura de datos de sensor\_config |
| alarm\_config | get | (\*) | * Este comando actúa como una señal, indicando al sistema que responda con la configuración de alarmas almacenada. * La respuesta tiene la misma estructura, pero con una matriz de JSON en el campo "data".   Para más detalle consulte la estructura de datos de alarm\_config |
| alarm\_config | set | (\*\*) | * Hace que el sistema guarde la configuración que se está enviando en “data”, para ello el formato debe ser correcto. |
| labels | get | (\*) | * Este comando actúa como una señal, indicándole al sistema que responda con las etiquetas almacenadas en el sistema. * La respuesta tiene la misma estructura, pero con una matriz de JSON en el campo "data".   Para más detalle consulte la estructura de datos de labels. |
| labels | set | (\*\*) | * Hace que el sistema guarde la configuración que se está enviando en “data”, para ello el formato debe ser correcto. |
| discovery | start | (\*) | * Hace que el sistema inicie el modo Descubrimiento. Esto implica reconfigurar la velocidad de datos a la que la web recibe nuevos datos y que el lector recopile lecturas tanto de calibración como de temperatura. |
| discovery | stop | (\*) | * Hace que el lector deje de leer por completo. No se recibirán datos después de la ejecución de este comando |
| normal\_mode | start | (\*) | * Hace que el sistema inicie el modo Normal. Esto implica reconfigurar la velocidad de datos a la que la Web recibe nuevos datos |
| normal\_mode | stop | (\*) | * Hace que el lector deje de leer por completo. No se recibirán datos después de la ejecución de este comando |
| modbus\_config | get | (\*) | * Este comando actúa como una señal y le indica al sistema que responda con la configuración Modbus del sistema. * La respuesta tiene la misma estructura, pero con una matriz de JSON en el campo "data". |
| modbus\_config | set | (\*\*) | * Hace que el sistema guarde la configuración que se está enviando en “data”, para ello el formato debe ser correcto. |
| modbus\_table | get | (\*) | * Este comando actúa como una señal, le indica al sistema que responda con la tabla Modbus. * La respuesta tiene la misma estructura, pero con una matriz de JSON en el campo "data". |
| new\_sensor\_data | Get | (\*\*) | * Este comando reporta la nueva información disponible de sensores, es unidireccional. |
| alarm\_states | Get | (\*\*) | * Este comando reporta la nueva información disponible de alarmas, es unidireccional. |
| hs\_config | Set | (\*\*) | * Este comando hace que el sistema guarde la configuración que se esta enviando en “data”, para ello el formado debe ser correcto. |
| hs\_config | Get | (\*) | * Este comando reporta la configuración actual del equipo FDX-HS. |

(\*) Campos vacíos.

(\*\*) Campos con una estructura especifica.

Example:

User opens “Sensors” tab->

Web sends commands to load registered sensors: {“cmd”:”sensor\_config”,”arg”:”get”,”data”:””}

The system responds::

{“cmd”:”sensor\_config”,”arg”:”get”,”data”:”[array\_sensor\_config]”}

Web classifies registered and non-registered sensors and visualizes the data within their respective tables.

User presses “Play” button , Web sends command

{“cmd”:”normal\_mode”,”arg”:”start”,”data”:””}

From now on the Web will periodically receive new data, which will be parsed and visualized on the tables.

## Estructura de datos de los comandos

Los comandos son strings, con una estructura tipo JSON de la siguiente forma:

{ “cmd” : “campo\_1” , “arg” : “campo\_2“ , “data” : “campo\_3“ }

### Sensor\_config:

**“data” : [ sensor\_cfg\_1 , sensor\_cfg\_2, … , sensor\_cfg\_N ]**

Where each JSON which holds sensor configuration has the following structure:

**{**

**“id” : int\_num ,**

**“config” : [“location”: num\_l , “equipment”: num\_e , “position” : num\_p]**

**}**

**id:** Integer number (It can be a very long one -> long long int)

**config:** JSON holding each label.

**location:**  An integer number. It can range from - to - .

**equipment:** An integer number. It can range from - to - .

**position:**  An integer number. It can range from - to - .

### Alarm\_config:

**“data” : [ alarm\_cfg\_1 , alarm\_cfg\_2, … , alarm\_cfg\_N ]**

Where each alarm config JSON has the following structure:

**{**

**“name” : “” ,**

**“set\_point” : int\_1 ,**

**“alarm\_type”: int\_2 (0:non set,1: absolute, 2: unbalance, 3:dispersión)**

**“relay\_flag” : int\_3 (0: non set, 1: rele 1, 2: rele 2)**

**“fields”: [ pair\_1 , pair\_2 , … , pair\_N ]**

**}**

And each array pair JSON has the following structure:

**{ “location” : int\_4 , “equipment” : int\_5 }**

**name:** String with the user-assigned name for the alarm.

**set\_point:** An integer number. It can range from - to - .

**alarm\_type:** An integer number. It can range from - to - .

**relay\_flag:** An integer number. It can range from - to - .

**fields:** JSON array. It holds fields configured by the user.

### Labels:

**“data” :**

**{**

**“location”: loc\_array,**

**“equipment”: eq\_array,**

**“position”: pos\_array**

**}**

**loc\_array:** Array of string labels. It can hold up to -.

**eq\_array:**  Array of string labels. It can hold up to -.

**pos\_array:** Array of string labels. It can hold up to -.

### new\_sensor\_data

**“data” : [ sensor\_data\_1 , sensor\_data\_2, … , sensor\_data\_N ]**

Where each sendor\_data JSON has the following structure:

**{**

**“id”: long int,**

**“avg\_temp”: float,**

**“temp”: float**

**“std\_dev”: float,**

**“n\_readings”: unsigned int,**

**“quality”: String (‘Out of service’, ‘Bad’, ‘Regular’, ‘Good’, ‘Excellent’)**

**“rssi”: float (Always negative, it represents signal strength)**

**“time\_stamp”: unsigned int (In seconds)**

**}**

### alarm\_states

**“data” : [ alarm\_data\_1 , alarm\_data\_2, … , alarm\_data\_N ]**

Where each alarm\_data JSON has the following structure:

**{**

**“id”: long int,**

**“alarm”: boolean (0:no alarmado, 1: alarmado)**

**}**

### hs\_config

**“data” :**

**{**

**“serial”: long int,**

**“dir\_modbus”: long int,**

**“baudrate”: long int,**

**“bit\_paridad: string (“sin paridad”, “par”, “impar”)**

**}**

### Modbus\_config:

\*\* TO BE DEFINED \*\*

**“data” : { }**

### Modbus\_table:

\*\* TO BE DEFINED \*\*

**“data” : { }**