Tampere University of Applied Sciences

Historian signal diagnostics

Jussi Salokanta

BACHELOR'S THESIS
December 2023

Degree Programme in Software Engineering

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree programme in Software Engineering

JUSSI SALOKANTA
Historian signal diagnostics

Bachelor's thesis 24 pages, appendices 0 pages
December 2023

In industrial environments a huge amount of history is collected often through thousands of different signals, but how can it be made sure that everything is being collected successfully? Data is often stored in historian databases in industrial environments and those can become large quickly. Modern distributed control systems can have thousands of signals delivering continuous stream of data.

Even if one has access to a database, which not many should have, it requires skills to find and analyze any useful data. This problem can be solved by having a tool that does that for the user. Web-based applications that run on a browser are on the rise and this signal diagnostic tool is made to be used on browser and is part of Valmet automation’s historian configurator.

This thesis is about the process of developing a new tool for Valmet. From the requirements to design and from design to implementation. In the end its suitability is evaluated, and how it meets the requirements.

Signal diagnostics tool is strictly for looking at a single signal data and does not edit it in any way. Data is fetched from a database and then displayed on the screen for the user in chart and in table format. This removes the problem that not all has access to databases and skills to use them.

Key words: signal diagnostics, distributed control system, historian database

CONTENTS

LIST OF ABBREVIATIONS

API	Application Programming Interface
DCS	Distributed Control Systems
JSON	JavaScript Object Notation
UI	User Interface
UX	User Experience

1 INTRODUCTION

Control systems consists of mechanical or electronic devices that adjust other devices or systems. Usually, these systems are computerized, highly automated, and they are a central part of production. Control systems provide better production as they enhance efficiency and safety (Collimator website 2023). Modern control systems can range from small to huge, distributed control systems (DCS) consisting of many thousands of field connections. Collected signals values history are often stored in historian databases. Historian services are a major part of the Industrial systems, where many devices generate continuously data. These programs do collect, save, and retrieve data at high speeds, which enables monitoring, and analysis often in real time. (Influxdata website n.d.)
The amount of data that is collected through thousands of signals is so huge that it needs to be compressed using different algorithms.

Diagnostics tools are used to verify that all the important signals are collected properly, and they contain the right collection parameters. This enables good data analysis and visualization.

In chapter two the first part is software automation, second part is importance of it to Valmet and third part is requirements of the tool. Chapter three is about the design of the tool in two parts: User interface / user experience and software design following with technologies used in the implementation process. Chapter four and five are about how the work came out regarding the planning and implementation and what could be added to it in future.

2 AUTOMATION SOFTWARE

A distributed control system is a system of different sensors and controllers that are distributed around a plant. Control systems are highly automated, and they can operate independently, for example increase, or decrease production depending on the demand. A good example is that in powerplant it can increase the output of electricity in high demand situations such as cold days in the winter.

In the processes there are thousands of different kinds of measurements instruments like temperature sensors, flow sensors, and actuators in pumps and motors. Controlling these processes are often handled automatically by the controls systems based on the configuration set by the operators. For problem solving and optimization the history of all the signals is typically required.

Even with data compression the historian databases fill up eventually and that reduces the performance and requires human supervision. Automatic data deletion functionality is set to delete the old data. Historian configurator can be used to set the limits to how long the data must be stored and when it is allowed to be deleted. These limits must be agreed with customers. To prevent the space running out alarms are set to warn the users.

2.1 Automation in Valmet

Valmet automation provides future proof automation solutions to its products and services. Let it be Valmet's own products or existing facilities, Valmet offers a wide variety of automation solutions.

Valmet DNA Distributed Control System is a highly cybersecure control system with web-based UI and configuration tools for controlling any industrial processes.

The Valmet DNA Engineering environment contains tools to engineer and maintain plant automation. It is a scalable multiuser environment for concurrent engineering. This tool allows life cycle management for all control applications. (Valmet website n.d.)

Having process variables and/or signal values stored in historian database is necessary for data-analysis, and system troubleshooting. Valmet has its own tool called Historian configurator (figure 9) which is used to define which signals are being used to collect data and their configurations.

2.2 Requirements specification

2.2.1 Purpose

The main purpose of signal diagnostics tool is to identify problems in data collection. With signal diagnostics tool the user can quickly see that the data collection is operational (figure 5). Without the tool users would need to have some other means to search the database to identify any missing data.

2.2.2 Functional requirements

Use case1: Verify that data collection works in general.

Scenario: User opens historian configurator and from the configurator the user opens signal diagnostics tool. By selecting a signal, the tool should automatically fetch the data from the historian database and the user should see the signals status.

Use case 2: Verify that selected data collection options are reasonable.

Scenario: User can quickly verify, for example, that data compression is configured correctly by looking at the data table and comparing it to the specification.

Use case 3: Troubleshooting why certain process components like a motor do not work as expected. In normal cases the motor component in the operator UI receives data from multiple signals which leads to that the user cannot easily identify which of these signals are causing the problem

Scenario: Signal diagnostics tool can display single signals raw data to the user, and this helps the user to locate what is causing the problem. User uses filter functionality to show only the motors signals in historian configurator. Then the user checks what kind of data these signals produce.

Case 4: In case 3, the user is narrowing down the possible problematic signals. After narrowing them down the user is sure that the problem is in one of the few signals which seem to give some values that is out of bounds.

Scenario: Selecting all three signals simultaneously the signal diagnostics tool should give the user a link that transfers the selected signals to an analyze tool which can show more than one signal at the same time.

Use case 5: The diagnostic tool should always be easily available.

Scenario: No extra installation for operator UI is needed for analyzing the data as the signal diagnostic tool is part of historian configurator and can be opened by clicking the open button.

Use case 6: In some scenarios process applications do not work as expected. This typically causes signal having non good quality. It is important to identify if collected signal don’t contain a good quality.

Scenario: The signal status bar in the diagnostics tool should clearly show the status of the selected signal.

Use case 7: The user wants to update the view without closing and opening the tool.

Scenario: Users click the refresh button to fetch the latest data without the need to close and open the tool and see the latest new values on the screen.

The tool should be made modular so that different parts of the tool could be used elsewhere as a standalone feature with little to no modifications. Modularity helps with the testing and this tool should be able to be tested automatically.

2.2.3 Goals of implementation

Table 1. Goals of implementation.

Goal	How	Why
Divide the work in smaller tasks	Evaluating what the final product is and thinking which parts could be made its own component	Doing smaller parts and combining them one by one to the whole product is an agile way and it can produce modular components that can work even as standalone parts.
Testability	Tests can include manual testing like using the component and see if it works correctly or script-based unit tests.	Testing smaller components instead of the whole tool can be easier as the tester can focus on defined area of the tool. Implementation must be possible to be tested by test automation like Robot Framework
UI / UX design	A team of specialists oversees the design aspect. Working tool should look and work like it was designed and fit in with the other tools.	To have the best possible user experience and that the design is made to look like the other tools.
Finalization	Before release, the tests that covers the whole tool is to be carried out and fix any major findings.	This tool is supposed to be a time saving and helping tool that gives the users a quick overview of the signal and if the collector is operating and how the process is performing.

3 SIGNAL DIAGNOSTICS TOOL

Historian configurator is used to define which signals are added to the historian database for collection and configure the collection parameters.
The diagnostics tool opens on the right side of the screen next to the table of collected signals (figure 9). It is a quick way to check if the selected signal is still collecting data or has it stopped.

When the data that is fetched from the historian database it is often just plain values and strings. By creating a chart of the data and a table with values and corresponding names the raw data becomes easy to understand (figure 12). That is the main functionality of this tool. The signal diagnostics tool fetches selected signals key values from historian database and draws a chart of it and predefined values are presented in a table format with name/value pairs (figure 8).

3.1 UI Design

The user interface follows the same specification as other tools so users who operate in different tools don’t have to learn many new things.
The tool is designed to be easy to use. It has minimal buttons and requires only to be opened and a signal to be selected.
Different information is shown throughout the tool and from top to bottom it becomes from easy to consume visual to more technical table form at the bottom (figure 12).

Several meetings with stakeholders from different teams were held and everyone was able to contribute to the UI/UX side as they were able to tell what data they would like to see in the tool to help in their work.

3.2 Software design

A variant of Scrum was used during the development. Designing the software started from looking at the big picture, the milestones, and the wanted outcome from UI/UX specifications. Splitting the work to small but meaningful parts which

could be implemented during the sprints. Building the software in smaller parts is part of agile working style. It reduces the amount of planning when focusing on small part at the time. Creating a backlog of the tasks to Jira which is an issue / project tracking software.

3.3 Technology stack

Visual Studio Code
is a code editor made by Microsoft which is a lighter version of the Visual studio. Fresh installation of VS code has support for most common programming languages like C, C++, Java, and JavaScript. It has extensions which can be downloaded directly in the VS Code's extensions tab and installed to support other languages. (Visual studio code website n.d.)

TypeScript
In short it is a typed version of JavaScript, a superset. It is a high-level language and allows you to type your code and possibly catch errors earlier which brings safety to programs. (Typescript website n.d.)

Redux
It is a centralized state management framework used in front-end web applications. It allows using the state of different components in a centralized store without passing them as properties repeatedly. Redux updates the UI based on what is in the state and the state is updated by actions dispatched to it. (Redux website n.d.)

SCSS
Sassy CSS is a scripting language that is used to style web applications. It is like regular Cascading Style sheets (CSS) but it can improve maintainability and readability. In the end SCSS is compiled to regular CSS. (SASS website n.d.)

3.4 Implementation

In software development keeping the source code in well-structured and maintained folders is essential for the life cycle of the product. It enables finding the code and different components easier for future work.

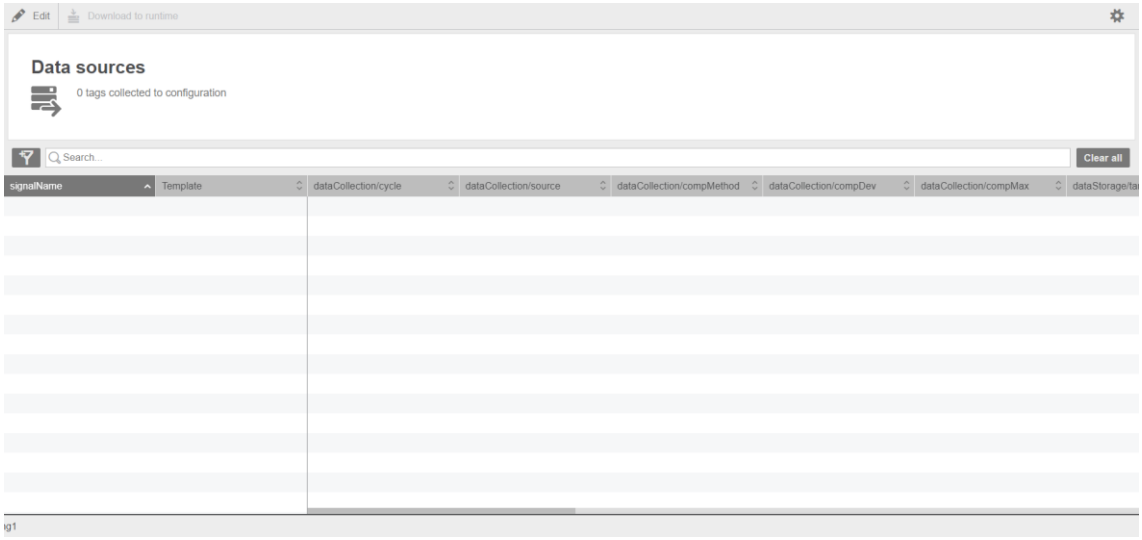


Figure 1. Historian configurator without the signal diagnostics tool.

The signal diagnose tool is located on the right side of the screen inside a tool called historian configurator. In figure 1 is the historian configurator before the implementation of signal diagnostics tool. The Signal diagnostics tool has an open and close functionality to leave more visibility in the historian configurator. Creating an opening empty sidebar and having it visible on the screen marked the start of the project. The base was then divided into three bigger components. header, chart, and table.

Header consists of the open and close button and icon with message displaying the status of the signal and a refresh button (figure 2). Placing an icons and text helped with the styling of the component.

Figure 2. Early-stage header component.

The second component, chart, was created and using border colors the placement of it was easier to see (figure 3). At the bottom of the chart component a button was added which would then dispatch the information to other tool for deeper analyzing.

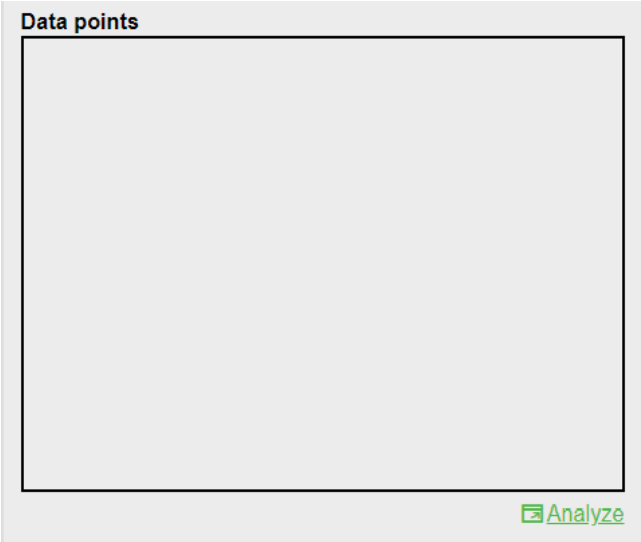


Figure 3. Early-stage chart component.

Third component, the data table was then added. The visible empty table was the first part. The table is vertically divided into key value pairs. The left side is for keys (names) and right side is for values. In Figure 4 below is shown a very early stage of the data table with one category and one key/value row to test it.

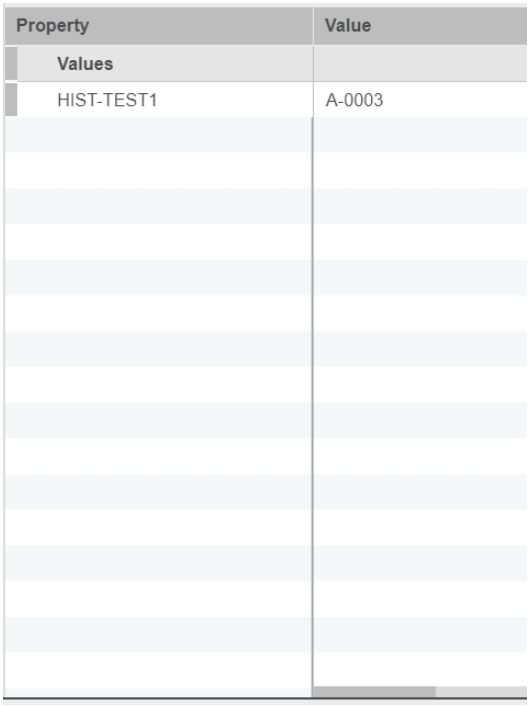


Figure 4. Early-stage data table component.

These three main components were then placed in a collecting component which oversees rendering. With this collector component the possibility to group and hide separate components became easier.

Implementation of functionality was after visualization. Using fetch API, the data is fetched from server and content of the tool is updated.

The tool displays various messages to the user, one being that if no signal is selected the components are hidden and the message tells the user to select a signal. When selecting just one of signals the components which show the values are shown and then when user selects more than one the tool displays a message and a link to analyze them in other tool designed to handle multiple signals at once. If the fetch fails or the signal is not present in the database a warning message is displayed to the user.

The header component, which was the simplest of these components, was made first (figure 5). Using fetch to get the data and then creating functions to change it into the wanted form and then replacing the placeholders with the real data. Depending on the values the header can dynamically change icons and messages.

Figure 5. Header component.

Chart component required values in a certain time range to be fetched and passed to the chart plotter (figure 6). It uses browsers own canvas API to render the charts.
X-axis has values from timestamps and Y-axis has the measured values from sensors.

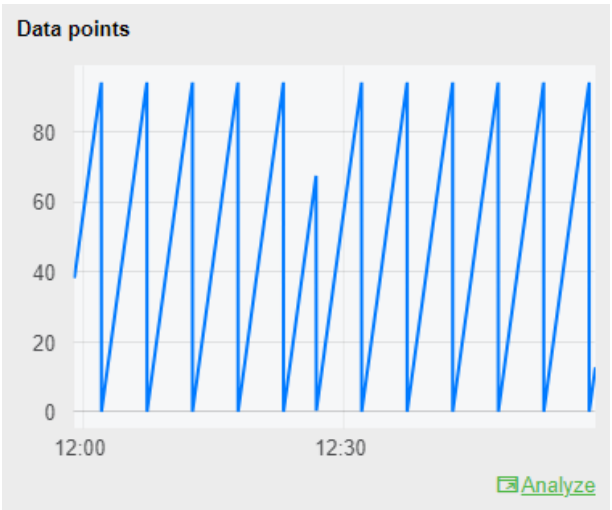


Figure 6. Working chart component.

When chart component was first implemented it used only a line chart and that did work for most charts, but when the collection was set to binary meaning the values measured were only 0 and 1, the outcome in line chart was not clear. Stepped mode for the chart was then implemented and used when the collection parameter had method as stepped (figure 7).

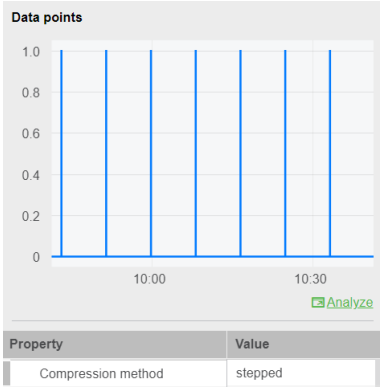


Figure 7. Stepped mode in chart component.

Table component required more than one fetch from the database. Setting up the fetches and then data handling functions, the values were shown in the table (figure 8).

Fetched data was in JSON format and as such was not usable. The responses were saved in a React state but needed to be handled with functions that leveled nesting and changed names to more descriptive display names. This way it became ready to be displayed in the UI.

Property	Value
Values	
Latest value	-40.35356604036664
Latest value timestamp	2023-11-01 10:32:59.539
Latest value status	0x0
Latest history value	-40.35356604036664
Latest history timestamp	2023-11-01 10:32:59.539
Latest history status	0x0
Oldest history value	9.320390859672262
Oldest history timestamp	2023-07-03 11:03:34.422
Oldest history status	0x0
Unit	%
Collection	
Collection cycle	500ms
Retention policy	mediumtermrp
Compression	
Compression method	swingingDoor
Compression deviation	0.1
Compression max time	1m
Types	
Type	/double_q
Signal type	/double
Services	
Collection service	history

Figure 8. Data table.

During the implementation of functionality of the refresh button brought along a major overhaul of the tool. Using separate states in different components was a problem. React states could be passed to other components and functions with props but that can lead to a messy code. Solution was to rework states and data handling. Fetch functions were moved to redux. Changing the way of how the fetches returned the data directly to the type of what was needed in the UI made separate data handling functions obsolete. Data that was now in redux centralized store could be accessed from anywhere within the tool's components. This rework enabled the refresh button to use a dispatch to the redux store to launch the API calls again and refresh the different components.

Even when the diagnose tool is open it fits well in the historian configurator tool without disrupting the usage of the configurator as seen in figure 9 below.

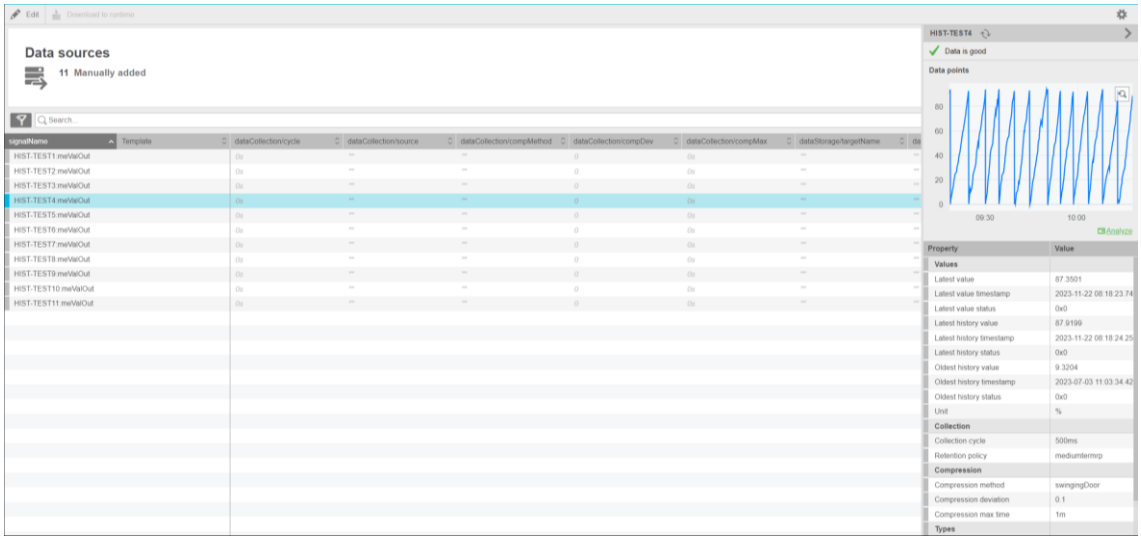


Figure 9. Finished signal diagnostics tool inside historian configurator.

3.5 Workflow

Basic workflow of the tool is shown in Figure 10. Conditional rendering will determine what the user can see. It is designed to work with one signal at a time.

The user can either open the tool first and receive a message that a signal must be selected or select a signal and then open the tool. If the user selects more than one signal at a time, the tool shows only a link to analyze the selected signals with other tool that is designed to work with multiple signals. If only one signal is selected, and the tool is open, the fetches are dispatched to get the data. Then depending on the response one of the following happens: 1. If the response is not ok an error is shown to the user. 2. If the response is ok, the data is received and rendered to the screen as seen in figure 12.

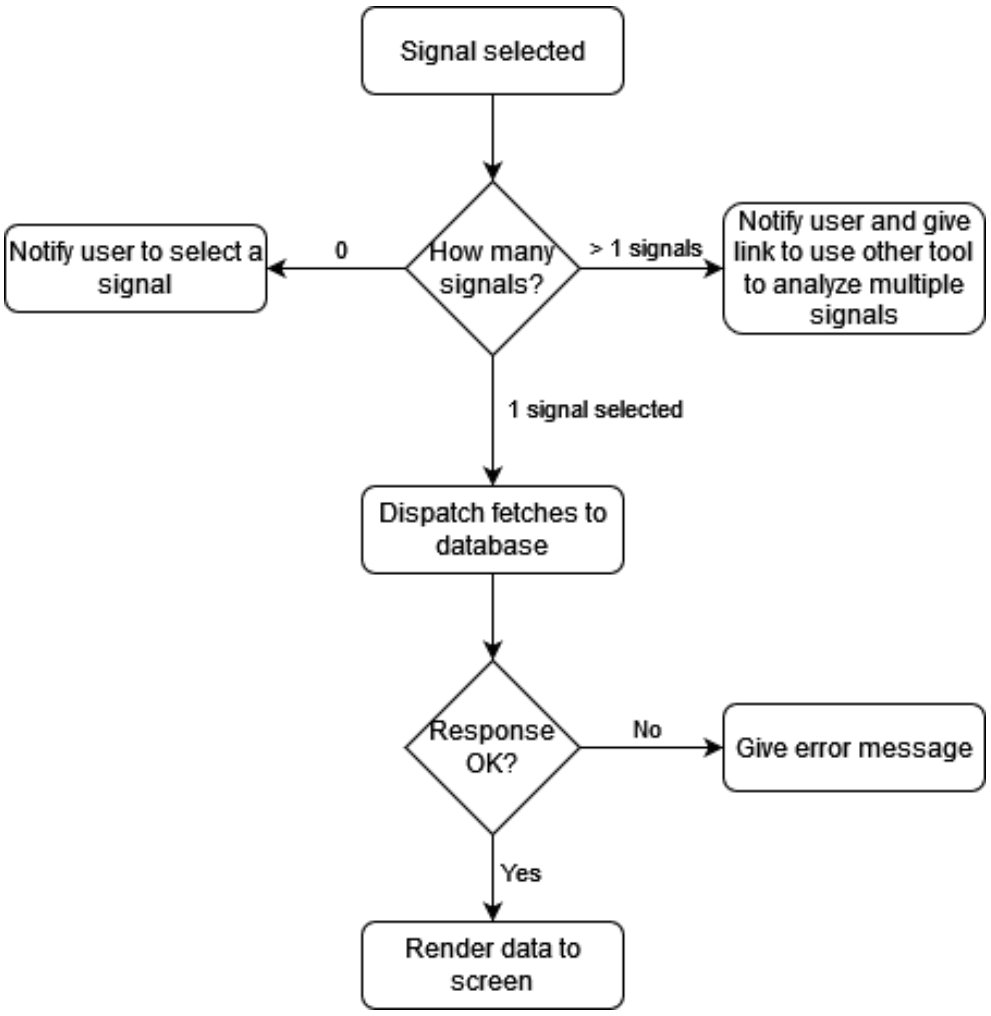


Figure 10. Basic workflow of signal diagnostics tool.

The refresh button is in the header component, and it refreshes only the tools components (figure 5). Refreshing only the tools components is made possible by using redux. Each component is connected to the store and when clicking the refresh button, a dispatch with an action and current state is then sent to the store. Redux store then uses reducer that takes current state and compares it to the arriving state and action. The state is then updated. A notification is sent to UI side components which then checks if any of the states they use has changed. If the current state is different than what the components find from the redux store it forces them to re-render the UI with the updated state as seen in Figure 11. Below.

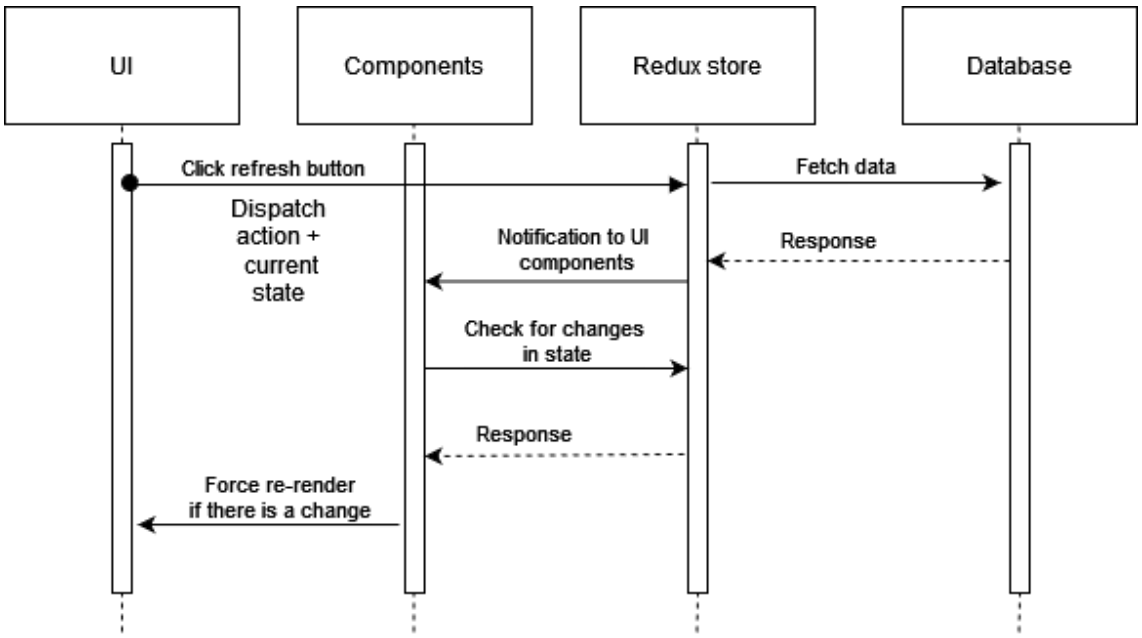


Figure 11. Functionality of refresh button.

4 DISCUSSION

The aim of this project was to have a easy to use tool to check individual signals status and display the key values and chart on the screen without the need of the user to access database by themselves.

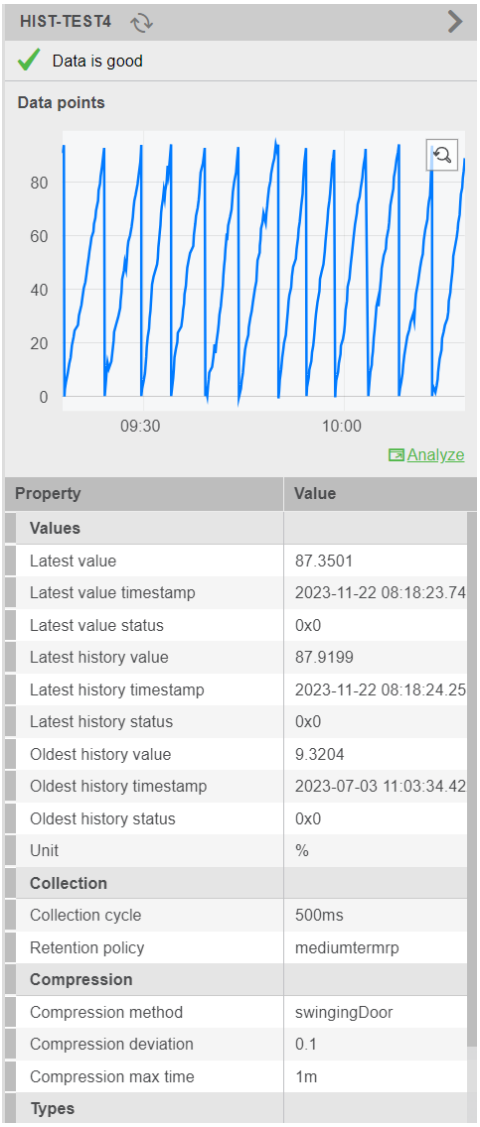
In the beginning of the project the needed amount of work was hard to estimate. The difficulty of creating needed components varied and the initial thoughts of which were easier, and which were harder to implement came through.

Starting from design to working product takes much planning and adjusting to the changes that came a long the way was expected and handled using agile approach. During sprints, implementing different components and functionalities, problems often occurred. If the problem was a blocker, it was dealt with immediately and if it was not a blocking issue a ticket for it was written in Jira and it was fixed later. Consistent daily coding and constructive feedback from pull requests, guidance from seniors has contributed to developing better coding skills and problem-solving skills. Sprint reviews and meetings have helped with soft skills development. Which is a necessity for a software developer.

The layout of the different parts inside the diagnostics tool gave a lot of things to think about. Having two components side by side and the other one having three different parts column-wise required many elements in the HTML side. At the beginning it was hard but having to do that many times it got easier. Thinking of them as boxes, inside other boxes, inside other boxes and so on until the final items were properly laid out was the trick used.

In paragraph 2.2.2 Functional requirements there are several use cases which the final product is wanted to meet. The final product can meet all those requirements. For example, use case 1. Is about that user can verify that the data collection is working or use case 2. That the collection options are correct. Both cases are easily verified by looking at the status bar and seeing the signals status or at the table where the options and values are shown.

After all the functionalities were made a meeting with the UI team was held to inspect how the tool looked and felt. It was given a pass with some minor visual updates here and there to be made and the final stage is shown in figure 12.



Property	Value
Values	
Latest value	87.3501
Latest value timestamp	2023-11-22 08:18:23.74
Latest value status	0x0
Latest history value	87.9199
Latest history timestamp	2023-11-22 08:18:24.25
Latest history status	0x0
Oldest history value	9.3204
Oldest history timestamp	2023-07-03 11:03:34.42
Oldest history status	0x0
Unit	%
Collection	
Collection cycle	500ms
Retention policy	mediumtermrp
Compression	
Compression method	swingingDoor
Compression deviation	0.1
Compression max time	1m
Types	

Figure 12. Signal diagnostics tool.

5 FUTURE WORK

In current implementation the chart is not very dynamic. It could be made to refresh automatically instead of just when refresh button is pressed.

Having an option to print or otherwise export the tools collected values could bring value to some users as then they could save them and come back to them later to see how they compare to the current situation.

Currently the chart is drawing lines from all values, and it does not indicate if there are faulty values in it. The chart could be improved by using a small mark to show faulty values in the lines.

REFERENCES

Collimator website 2023. What are control systems? Read on 5.11.2023. https://www.collimator.ai/reference-guides/what-are-control-systems

Influxdata website. N.d Data Historian. Read on 23.11.2023. https://www.influxdata.com/glossary/data-historian/

Redux website 2023. Redux state container. Read on 26.11.2023. https://redux.js.org/introduction/getting-started

Sass website N.d. SCSS stylesheet language. Read on 16.11.2023. https://sass-lang.com/documentation/

TypeScript website. N.d. Typescript programming language. Read on 15.11.2023. https://www.typescriptlang.org/

Valmet website. N.d. Valmet DNA engineering environment tools. Read on 20.11.2023. https://www.valmet.com/automation/distributed-control-system/engineering-maintenance-tools/

Visual studio code website. N.d. Visual Studio Code. Read on 15.11.2023. https://code.visualstudio.com/