NTNU
Norwegian University of
Science and Technology

Master's thesis

2019

Odd Harald Sjursen Sande, Andreas Børhaug

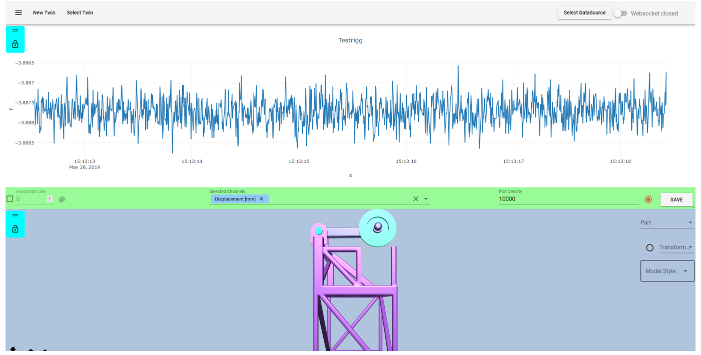
NTNU
Norwegian University of
Science and Technology
Faculty of Engineering
Department of Mechanical and Industrial Engineering

Master's thesis

Odd Harald Sjursen Sande
Andreas Børhaug

Developing a Client for a Digital Twin Cloud Platform

June 2019



Horizontal Line: 0
Selected Channels: Displacement [mm]
Plot Density: 10000

NTNU
Norwegian University of
Science and Technology

NTNU
Norwegian University of
Science and Technology

Developing a Client for a Digital Twin Cloud Platform

Odd Harald Sjursen Sande
Andreas Børhaug

Engineering and ICT
Submission date: June 2019
Supervisor: Terje Rølvåg
Co-supervisor: Bjørn Haugen

Norwegian University of Science and Technology
Department of Mechanical and Industrial Engineering

Sammendrag

Det er forventet at selskaper bruker en stadig større mengde ressurser på å implementere Digitale Tvillinger hvert år, spesielt med tanke på prediktivt vedlikehold og overvåking av strukturell integritet. På det nåværende tidspunkt eksisterer det ikke noen åpen kildeplattform for å visualisere data fra Digitale Tvillinger. Denne avhandlingen beskriver utviklingen av en slik platform, spesielt med fokus på frontend og det grafiske brukergrensesnittet.

Prototypen som er blitt utviklet støtter plotting av sanntidsdata i form av tidsserier samt visualisering av en 3D modell. 3D modellen speiler bevegelsen til den fysiske tvillingen som er valgt, basert på informasjon fra en Functional Mock-up Unit (FMU). Prototypen er generalisert til å støtte en vilkårlig Digital Tvilling så lenge den følger FMU standarden. Ved å implementere forskjellige komponenter for plotting og visualisering lar prototypen brukeren lage fleksible og modifiserbare oppsett. Brukeren kan videre definere prosessorer for å transformere data, for eksempel Fast Fourier Transform, Butterworth filtre og FMUer for simulasjon.

Summary

Companies are predicted to allocate a greater amount of resources to implement Digital Twins in their business every year. especially in regards to predictive maintenance and monitoring structural integrity. However, currently there exists no non-proprietary cloud platforms for visualizing data from Digital Twins. This thesis documents the development of such a platform, especially the front end and the Graphical User Interface.

The prototype developed, supports plotting real-time data as time series and visualizing a 3D model. The 3D model replicates the movement of the physical twin selected, based on output from an FMU. The prototype is generalized to support any Digital Twin following the FMU standard. By implementing different components for plotting and visualization, the prototype allows the user to create flexible and customizeable layouts. The user can also define processors for transforming data, such as Fast Fourier Transform, Butterworth filters and FMUs for simulation.

Preface

This Master's thesis is written on behalf of the Department of Mechanical and Industrial Engineering (MTP) as part of the study program Engineering and ICT. The project was initialized and completed during the spring 2019 semester as a continuation of a specialization project completed the prior semester. Supervisor *Terje Rølvåg* proposed this project with the aim to bring Digital Twins into the Cloud. *Terje Rølvag* along with our co-supervisor *Bjørn Haugen* has been providing assistance and guidance throughout the project period. External assistance has been provided from two companies, *Ceetron* and *Fedem Technologies* (Now part of SAP). *Fedem Technologie*s has assisted with both software to do calculations on models as well as human resources to help in our utilization of their software. *Ceetron* has provided us with code excerpts and software to visualize these models in 3D directly in the browser, as well as being available to answer relevant questions regarding their software. This thesis assumes the reader has a general understanding within the field of IT development, mechanical engineering and signal processing.

Table of Contents

List of Tables

List of Figures

Listings

Acronyms

API Application Programming Interface. vii, ix, 3, 8, 14, 15, 36, 40

CBMS Cloud Based Monitoring System. 2

CSS Cascading Style Sheets. 9

CSV Comma Separated Values. 30

FFT Fast Fourier Transform. 3, 8, 46

FMI Functional Mock-up Interface. 9

FMU Functional Mock-up Unit. i, viii, 9, 25, 46, 48, 49

GUI Graphical User Interface. 10, 13, 28, 46

HTML Hypertext Markup Language. 9

HTTP Hypertext Transfer Protocol. 8, 15

IDE Integrated Development Environment. 14

JSON JavaScript Object Notation. viii, 15, 38

MTP Department of Mechanical and Industrial Engineering. ii, 1, 3

PLM Product Lifecycle Management. 5

TCP Transmission Control Protocol. 8

UI User Interface. 38

USG Unstruct Surface Grid. 25

Chapter 1

Introduction

This thesis describes the development of a client to interact and utilize backend developed in the companion project [7] to this thesis. As a development project, the focus will be both on the result as well as the road from initial conception to finished product. This chapter presents the background, scope and outline of the thesis.

1.1 Background and Motivation

Following previous projects at MTP in the field of Digital Twins, a need to establish an ecosystem in the cloud where Digital Twins may be accessed and configured has emerged. Such a system would eliminate the need for direct access to powerful hardware with large enterprise programs installed to do the necessary calculations for your digital twin. This would move the load over to a centralized server that can be utilized by any sanctioned device at any location. As a direct consequence, new opportunities to use the Digital Twin may be realized such as running calculations or creating new views using nothing but a device with internet access. During maintenance, skilled operators may be able to use and update the Digital Twin on-site to help them complete their tasks, without having to report irregularities back to a central hub that controls the Digital Twin.

The supervisor of this project, *Terje Rølvåg* at MTP has been cooperating with *Fedem Technologies* regarding Digital Twins. Along with our co-supervisor *Bjørn Haugen*, *Rølvag* proposed that a project to launch the Digital Twin into the cloud. At it's core, the idea was to create an application for modelling Digital Twins in the cloud that could support a wide range of different fields. Input data from two concurrent projects was to be made available for testing as well as the the Torsion Bar Suspension Rig used in B.

1.2 Research Goals

The complete project has been sectioned into multiple individual but dependent projects: The twins that will be monitored and the CBMS. *Terje Rølvåg* defined five goals for the CBMS to help give direction to the project as a whole. These five goals were as follows:

1. Identify structural failure modes to be detected by CBMS (fatigue, yield, buckling, instability etc.). Collect inputs from the generator and crane master students.
2. Identify the functional requirements for monitoring of the most critical failure modes. Collect requirements from the generator and crane master students.
3. Implement a generic configuration system in the cloud solution for easy adaption to other digital twin applications (other sensors, actuators, streaming analytics etc.)
4. Implement required software functionality in the cloud solution to support the requirements from task 2 (streaming analytics, curve plotting, 3D visualization, event trigging, report generation)
5. Setup and benchmark the CBMS on a physical crane

The CBMS project was split into two sections, frontend and backend [7] where this thesis is covering the frontend part and how these goals are part of the bigger picture. The overall goal for the front-end was that the final product should be a user-friendly interface for Digital Twins. The prototype should include generic configurations that would suit the master students of the crane and generator.

Another major goal is to facilitate inspecting the data the Digital Twin model can provide. This would allow the user to select what is critical for the current situation, e.g yield. To complete these goals, a few basic requirements must be met in regards to performance. As a result, performance has continuously been evaluated.

1.3 Research Scope

1.3.1 Objectives

While the overall goals were described in the previous section, the objectives will describe implementation steps to achieve those goals and provide discussion points for why changes happen during development.

1. Choose a framework for quick prototyping.
2. Choose and implement a visualization tool for graphs and charts.
3. Implement a layout creator and selector where the user can set up and save layouts for re-use.
4. Implement functionality to allow user to subscribe and receive both raw and transformed sensor data.
5. Set up Ceetron 3D visualization to show displacement of the 3D model currently being inspected in real time.
6. Extend the graphing tool to allow for different types of inputs, e.g Fast Fourier Transform (FFT).
7. Set up event triggers to notify users when a critical point of a Digital Twin is reached.
8. Create automatic reports showing a breakdown of critical events.
9. Allow for look-ups on saved data for statistics and long term analysis.

1.3.2 Limitations

The focus of the project has not been to create a commercial solution ready to use. The currently supported functionality will only be as accurate as the model. During development, a majority of the testing has been done on the *Testrig* located at MTP. While the general functionality should work for any Digital Twin, some verification and tweaks may be needed depending on how different the twin is. Furthermore, this project is reliant on what data and functionality is provided by the backend API. Additional functionality must both be added to the backend and the frontend before it can be accessed by the user.

The 3D Visualization using Ceetron's technology does not provide an exact real time replica of the physical model. Delay from calculations, transfer of data and the rendering on the screen adds a noticeable delay. However, the visualization does follow the model adequately and can be in most cases used as if it was real time.

1.3.3 Thesis Structure

Chapter/Appendix	Description
Introduction	Gives an introduction to the background and goals for the project
Theory	Describes concepts and technologies used in this project
Implementation	Describes how the project has been solved
Results	Showcases the end result of the project
Discussion	Discussion of the results and where it can be taken next
Conclusion	A critical view of the project

Table 1.1: Overview of thesis structure

Chapter 2

Theory

2.1 Digital Twins

2.1.1 Definitions of Digital Twins

There is not a single, fully accepted definition of Digital twin. The definition varies in what field it's used, and in what context. The concept as it is known today was first introduced back in 2002 by Michale Grieves, in the context of a presentation regarding Product Lifecycle Management (PLM). Grieves later went on to define Digital Twin in a paper written along with John Vickers in 2017:

> Digital Twin is a set of virtual information constructs that fully describes a potential or actual physical manufactured product from the micro atomic level to the macrogeometrical level. At its optimum, any information that could be obtained from inspecting a physical manufactured product can be obtained from its Digital Twin. [4]

Another commonly referred to definition from Glaessgen of NASA is:

> A Digital Twin is an integrated multiphysics, multiscale, probabilistic simulation of an as-built vehicle or system that uses the best available physical models, sensor updates, fleet history, etc., to mirror the life of its corresponding flying twin. [3]

Richard Howells of SAP [1] defines Digital Twin as a "digital representation of a real world object, product or asset" [5]

In a recent paper by Kritzinger et al. [9], the authors used a combination of the aforementioned definitions to define three levels of integration. A common theme in the definitions

[1] SAP - Systems, Applications and Products in Data Processing, a german ERP company

of a Digital Twin is that a Digital Twin is a digital counterpart to physical objects. Within that space, there is much room for interpretation. The three levels of integration introduced in the paper is an attempt to differentiate inside this space.

The three levels of integration are as follows: A digital model, A digital shadow and a digital twin. They all fulfill the basic requirement of being a digital counterpart to physical objects, however they do it to different degrees. A Digital Model is a manually created model of the physical asset. Any updates to the model is done manually. In a Digital Shadow data flows from the physical object to the digital representation to keep it up to date. A Digital Twin in their paper refers then to a further extension of a Digital Shadow, in which data flows both ways. State change in the Digital Twin will impact its physical counterpart and vice versa.

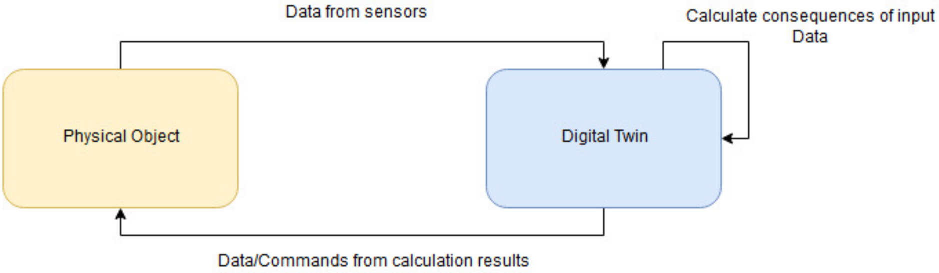

Data from sensors
Calculate consequences of input Data
Physical Object
Digital Twin
Data/Commands from calculation results

Figure 2.1: A Digital Twin as Kritzinger's paper defines it. Information flows automatically both to and from the Digital model and the physical asset. A Digital Shadow will not have the automatic communication back to the physical asset and a Digital Model will have no automatic communication at all

2.1.2 Benefits of Digital Twins

The main use cases of Digital twins are in regards to predictive maintenance and monitoring of structural integrity. Digital Twin lessens the need of on-site inspections and gives a better lifetime estimation. SAP claims that implementing Digital Twins gives a "25% reduction in the cost of quality defects with digital twins and that 65% of manufacturing businesses will be using Digital Twins by 2020. " [11]. Thomas Kaiser, SAP Senior Vice President of IoT, put it this way: "Digital twins are becoming a business imperative, covering the entire lifecycle of an asset or process and forming the foundation for connected products and services. Companies that fail to respond will be left behind."

Siemens claims that Digital Twins will reduce product defects and production costs as well as shorten the time to market [12].

Similar claims can be found from many manufacturing businesses and also in other fields. However Digital Twin is still not a mature technology. One of the reasons for this is that the cost of implementation and operation has been very high. While sensors have decreased in price over the years, the amount of data being transferred from a physical object may require expensive equipment to handle.

2.1.3 Examples of Digital Twins in practice

Digital Twins have been used in various fields to both different degrees of success and correctness. Many companies claim to use Digital Twins, however what has actually been implemented varies in a great degree. The McLaren group, for example talks positively about the impact their implementation of Digital Twins has had on performance in a sport where every second counts. Dr Peter van Manen, a former managing director and vice president for McLaren Applied Technologies is quoted saying:

> It's just the sort of thing a digital twin is perfect for helping with, Formula 1 is all about time management. Every second counts so when you can shave them off by learning key insights about the inner workings of your car, it really matters. We used a digital twin – though we just called it a computer simulation – to help us do that [13].

This shows that while the Digital Twin concept can be difficult to implement to its full definition, it is not required to provide value within competitive fields such as Formula 1 racing. Furthermore, Dr. van Manen has an interesting perspective of their experiences implementing a Digital Twin.

> For the first three to five years, we were just playing around with it, trying to work out what it could do – it's important to have this phase. Digital twins are not going to be perfect straight away – they're a bit like a puppy at Christmas – it's great but you have to keep taking care of it if you want to reap the benefits [13].

Another practical example comes from Brazil where Stara, a manufacturer of tractors has implemented Digital Twins. In cooperation with SAP they have implemented IoT sensors with the goal of providing live updates on how their tractors operate. Consequently, farmers using Stara's tractors have reported a 21% less usage of seeds and 19% less usage of fertilizers [6].

2.2 Data visualization

Visualization of data can come in many forms, charts, plots, maps, models etc. The classic visualization challenge is figuring out which visualization fits for your static data set. As mentioned in the previous section, a Digital Twin uses real time data, which is not a static data set. It is continuously updated with data from the physical asset and is closer to needing techniques derived from what the IT sector calls Big Data Visualization. In a Digital Twin, the amount of data will quickly become too great to keep in a database. Only some of the data can be stored. The data must to be visualized on the fly and decisions must be made whether or not the data should be stored for further analysis.

As a consequence, the first challenge is to choose what data must be visualized. For a Digital Twin the equivalent is to ask: Do you need to visualize the entire model at once? Another issue is encountered when attempting to manipulate the incoming data:

The computer cannot keep up. Each action must be repeated on each data point. The problem may be mitigated by increasing computational power, however this is expensive. Instead it might be advisable to limit the amount of actions to keep costs down.

2.3 Signal Analysis

2.3.1 Fast Fourier Transform

FFT, is an algorithm for efficiently computing the discrete Fourier series of a sequence. Cohen et al. [2] describes it as the following:

> The fast Fourier transform is a computational tool which facilitates signal analysis such as power spectrum analysis and filter simulation by means of digital computers. It is a method for efficiently computing the discrete Fourier transform of a series of data samples (referred to as a time series)

The discrete Fourier series of a sequence is commonly used to analyze a signals properties. Through identifying the higher frequency components of a signal, it may be possible to pinpoint the source of the unwanted vibration or noise. FFT is often used in combination with filters to remove the unwanted frequencies once the range has been identified.

2.3.2 Butterworth Filter

In signal processing a filter is typically used to remove noise, i.e unwanted frequency components, from the signal. A low pass filter, as the name states, lets only frequencies below a certain cut-off frequency pass through, while a high pass one only lets frequencies higher than the cut-off pass through. The range of frequencies that pass through a filter is called the bandpass, and an ideal filter should have a flat as possible passband as Butterworth states in his paper [1]: “An ideal electrical filter should not only completely reject the unwanted frequencies but should also have uniform sensitivity for the wanted frequencies.”

The Butterworth filter fulfills these conditions, more importantly the latter one, meaning it does not affect the frequencies that it should let pass. The higher order the filter has the flatter the frequency response from the passband becomes, however as the order increases, so does the latency of the signal. This is one of the filter that are available in the prototype.

2.4 Functional Mock-up Interface

> Functional Mock-up Interface (FMI) is a tool independent standard to support both model exchange and co-simulation of dynamic models using a combination of xml-files and compiled C-code.

> ...
> Models are described by differential, algebraic and discrete equations with time-, state- and step-events. [10] (from Modelica, the association behind FMI)

The models used in the interface are packaged in zip files with an .fmu extension, short for Functional Mock-up Unit. These FMUs contain several files. The definition of all variables and other information pertaining to the model is contained in an xml file, such that the target machine will not need to specify these. In addition a small set of C-functions provided in binary or as source files, which expose the model equations in a simple manner. An FMU might also contain additional data, especially maps and tables needed by the specific model.

2.5 Technologies and Frameworks

2.5.1 JavaScript

JavaScript is the core language used to create web-pages as it can be run directly in the browser with no additional installations. Together with HTML and CSS, it forms the most common foundation of modern web development in client side programming.

2.5.2 Vue

Vue.js[2] is JavaScript front-end framework for building user interfaces. Vue is component-based, meaning it splits Graphical User Interface (GUI) into encapsulated elements holding their own logic, template and styling. The application itself is built by combining these loosely coupled elements, called components.

Comparison with React

Vue, similar to React[3], which is backed by Facebook, focuses on the view layer of an application. However, unlike React, Vue offers a more complete solution for build Web Applications/Single Page Applications. While React relies on other third party libraries for advanced features such as Redux for state management, Vue supports it with their own library Vuex[4]. The Vue team also provides a command line tool, Vue CLI[5] for minimal configuration and instant prototyping.

[2] https://vuejs.org/

[3] https://reactjs.org/

[4] https://vuex.vuejs.org/

[5] https://cli.vuejs.org/

Additionally, Vue's documentation[6] states that Vue scales down just as well as up. This means that the framework should easily fit an application growing in size. Coupled with the Vue CLI, this makes the threshold for getting started with Vue.js low and a good choice for rapid development. Examples of large companies using Vue are Alibaba, Gitlab[7], Expedia and Adobe. [8]

Usage

The component approach is quite similar to that of a class in Object-Oriented Programming and likewise a component is instantiated inside a parent component with parameters.

A basic Vue component and it's usage is shown in listing 2.1 and 2.2 respectively. The styling in the bottom of the listing is written in CSS, the template is HTML and the logic is JavaScript within the script tag.

```
<template>
  <div class="basic-component">
    {{ text }}
  </div>
</template>
<script>
  export default {
    name: 'BasicComponent',
    props: {
      text: {
        type: String,
        default: 'Hello␣World'
      }
    }
  }
</script>

<style scoped>
  .basic-component{
    background: blue;
  }

</style>
```

Listing 2.1: A Basic Vue Component

```
<BasicComponent text="Hello␣There"></BasicComponent>
```

Listing 2.2: Using a Basic Vue Component

[6] https://vuejs.org/v2/guide/comparison.html

[7] https://about.gitlab.com/2016/10/20/why-we-chose-vue/

[8] https://github.com/vuejs/awesome-vue

2.5.3 WebSocket

WebSocket is a computer communication protocol, allowing communication in both directions simultaneously (full-duplex) over a single TCP (Transmission Control Protocol) connection. Comparatively, HTTP (Hypertext Transfer Protocol) connections are half-duplex, meaning they only allow communication in one direction at a time, similarly to walkie-talkies and most handheld radios. In the WebSocket protocol, message exchange is allowed while keeping the connection open and the server can send content to the client without receiving a client request first.

As a result the interaction between a web server and a browser has less overhead compared to HTTP polling, therefore placing less burden on the server. HTML5 WebSockets can also traverse proxies and firewalls, which many application have problems with [14]. Altogether it makes real time data transfer between server and client easily achievable and the WebSocket API[9] is therefore a good fit for streaming data fast and efficiently.

[9] https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API

Chapter 3

Implementation

This chapter covers the current implementation of the prototype. Since the main objective is to create the basis for a Digital Twin Cloud Platform, no heavy optimization is done pertaining to scalability or accessibility. Building a working model with clear paths towards further development has been prioritized instead.

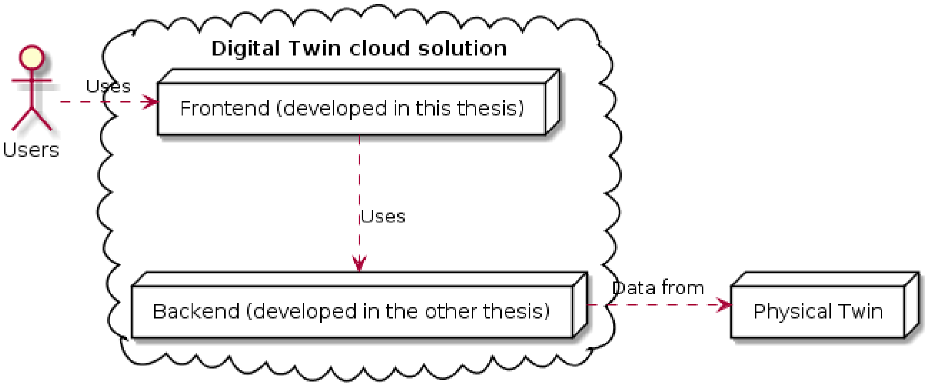

Digital Twin cloud solution
Users
Uses
Frontend (developed in this thesis)
Uses
Backend (developed in the other thesis)
Data from
Physical Twin

Figure 3.1: Overview of the CBMS with its coupling to the physical twins

Note, the chapter will mainly focus on implementation details and the code behind the GUI, so it's mostly directed at those who intend to develop the prototype further.

3.1 Requirements and Reasoning

In applications where large volumes of data are visualized, the main requirement is to be able to provide visualizations the user actually wants. Choosing between technologies and frameworks often becomes a question of scalability and availability. If the framework can't handle the necessary volumes of data, the solution is either to start again with a different framework or limit the functionality the user needs.

The technology needed to handle communication with the backend part of the project, both through request-based communication like REST or SOAP as well as a technology for receiving continuous data from the backend. Furthermore, it is desired for the project to be platform independent, requiring no extra installations on the user's part.

Due to this overarching requirement, and based on experience from prior projects, our front-end framework had the following requirements:

1. Quick prototyping
    - Sometimes you don't know if things will work out. Being able to retry without spending a month working on it is crucial.
2. Support development in Integrated Development Environment (IDE)
    - Framework needs to be well-established with development support in IDE for further timesaving and refactoring purposes.
3. Libraries and examples for graphical elements and standardized components
    - Most problems in the realm of graphical web applications have already been solved. Having a library with examples to choose and edit to fit the needs of the project can be a massive time saver.

3.1.1 Integration Requirements

At the frontend it is required to use libraries that work with the API exposed by the backend. Any frontend being developed including this project is therefore limited to select technologies that work within that scope. However, since both of these projects were developed simultaneously and as there is no additional projects depending on the backend, the backend can be adapted to support new technologies.

Current integration requirements are as follows:

1. WebSocket. WebSocket (section 2.5.3) is the current technology exposed to handle streaming of data from the Digital Twin(s)
2. REST. A REST inspired API is exposed allowing for standard http methods to request information from server. SOAP is thus not supported

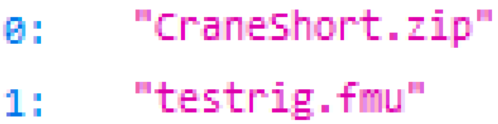
3.2 Back-end Communication

3.2.1 Resource requests

The frontend communicates with the backend via HTTP requests, such as GET, POST and DELETE. Sending a GET request to an endpoint returns a resource as a JSON-object, which can easily be read in the Vue/JavaScript frontend. An example is a list of running subscribeable sources. POST requests are used if the client needs to submit data with the request, for instance submitting a definition of sensors, ID, address and port in order to create a new data source. The methods used to make requests to the server are asynchronous functions in order to not halt the user interface during execution. Most of these can be found in the `APIHelper.js` file (Appendix C).

```
export async function getFMUs () {
  return getJSONResponse(rootAPI + /fmus/)
}
```

Listing 3.1: An example of a get request from the API helper file



```
0:    "CraneShort.zip"
1:    "testrig.fmu"
```

Figure 3.2: API response for get fmus request

3.2.2 Data Subscription

Data needed for visualization is continuously received through a WebSocket connection to the server. The process for opening a WebSocket to the server is depicted in figure 3.3. The client can then start sending subscribe and unsubscribe requests in order to receive data through the WebSocket (figure 3.4). These requests can also be executed before before the WebSocket connection has opened.

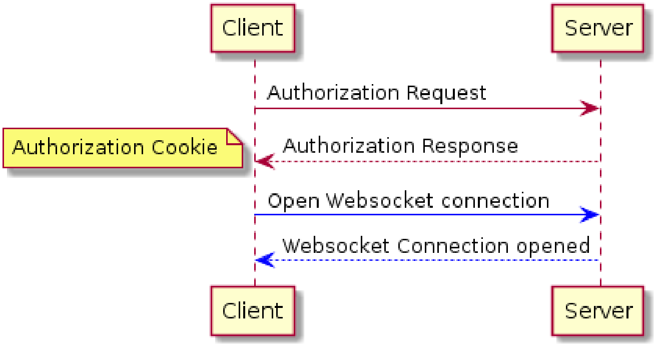


Figure 3.3: Initiating a WebSocket connection to the server

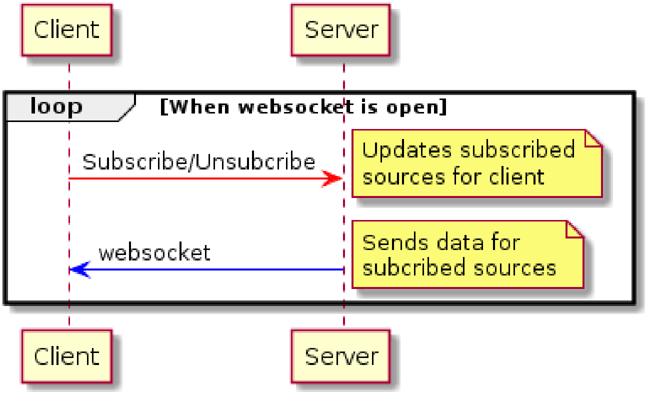


Figure 3.4: Subscription flow

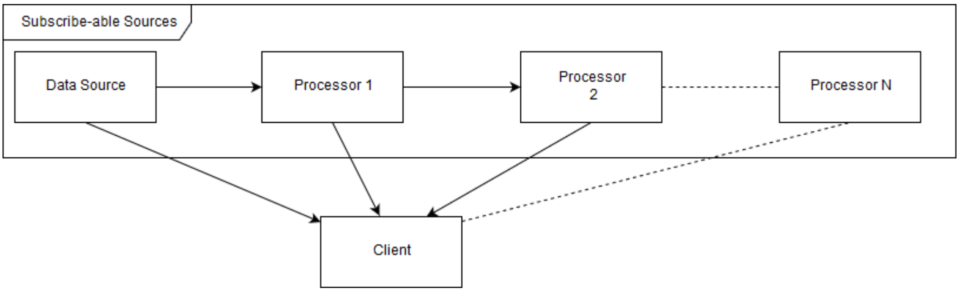


Figure 3.5: Subscribeable sources: Data sources and processors. Data sources output data directly from external source, meaning the physicals assets sensors, while processors transforms data from a data source or another processor. This means that processors can be chained, creating a pipeline with a source (the data source) and a sink (the client)

3.3 Data Flow

The incoming data from the WebSocket is received in the `channelParser` script, which parses the incoming data and bundles it in a buffer. When a set number of packets have been received the parser then emits a `newData` event with the buffer data. Visualization components that listen to this event will then receive all the buffer data and retrieve the relevant data for visualization. The plot component will for instance retrieve data for its selected channels (listing 3.4).

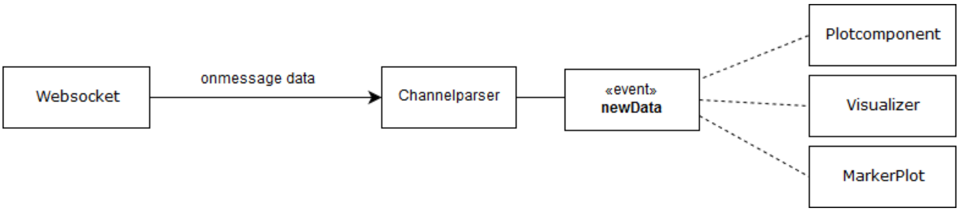


Figure 3.6: Simple overview over data flow in the client

3.3.1 Channel/Source Handling

The prototype uses Vuex[1] for state management. The Vuex store can be thought of as an in-memory database, a central store that holds the common state between components in the application. The visualizer, plot and markerplot component all need information on what channels have been subscribed to in order to visualize data. The subscribed channels are therefore stored in a dictionary located in the state object of the channel module in the application's Vuex store as displayed in listing 3.2.

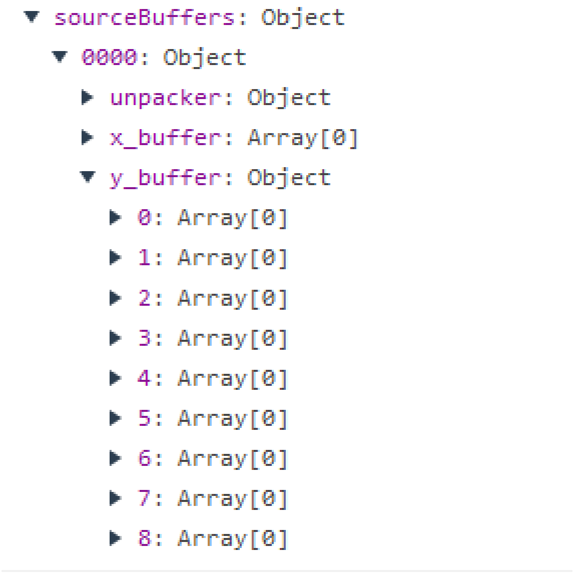
```
state: {
  // Dictionary of sourceIDs, example:
  /* sourceDict: {
    0000: {
      byteFormat: '<ddddddddd'
      name: 'testrig',
      channels: [
        {
          id: 1,
          name: 'Load [N]'
        }
      ]
    }
  } */
  sourceDict: {}
},
```

Listing 3.2: The channel module's state object

[1] https://vuex.vuejs.org/

3.3.2 Parsing data

The `sourceDict` consists of all channels the user have subscribed to, and is used to create a dictionary for parsing incoming data. Selecting all channels for the testrig data source results in the `sourceBuffers` dictionary shown in figure 3.7. The `unpacker` object depicted comes from the struct.js[2] library and is used to iteratively parse the incoming data. As listing 3.3 illustrates, the unpacked data is put into the x and y buffers related to the `sourceID`.



```
▼ sourceBuffers: Object
  ▼ 0000: Object
    ▶ unpacker: Object
    ▶ x_buffer: Array[0]
    ▼ y_buffer: Object
      ▶ 0: Array[0]
      ▶ 1: Array[0]
      ▶ 2: Array[0]
      ▶ 3: Array[0]
      ▶ 4: Array[0]
      ▶ 5: Array[0]
      ▶ 6: Array[0]
      ▶ 7: Array[0]
      ▶ 8: Array[0]
```

Figure 3.7: Snapshot of `sourceBuffers` for testrig with all channels selected (Vue Devtools). The X buffer holds to timestamp while Y buffer holds value

Every 100 milliseconds an event is emitted with the data received since last event. This event is called `newData` and sends a copy of the current `sourceBuffers` as shown in listing 3.3 line 19-25. Note that a copy of the `sourceBuffers` object is emitted in the event and not the object itself (line 21, listing 3.3). If the object itself had been passed, the `resetBuffers` method would start manipulating the `sourceBuffers` object while the event listeners process the data. In other words, data would be overwritten and lost.

```
1    parseData (data, sourceID) {
2      const sourceBuffer = this.sourceBuffers[sourceID]
3      if (sourceBuffer === undefined) {
4        return
5      }
6      this.packetCounter++
```

[2] `https://github.com/lyngklip/structjs`

```
    const unpacker = sourceBuffer.unpacker
    const unpackIterator = unpacker.iter_unpack(data)
    let unpacked = unpackIterator.next().value
    while (unpacked) {
      sourceBuffer.x_buffer.push(new Date(unpacked[0] * 1000))
      const channelsIds = this.subscribedSources[sourceID].channels.map
          ((it) => it.id)
      channelsIds.forEach(channelID => {
        sourceBuffer.y_buffer[channelID].push(unpacked[channelID + 1])
      })
      unpacked = unpackIterator.next().value
    }
  },
  async pushData () {
    if (this.packetCounter > 0) {
      EventBus.$emit(EVENTS.newData, deepCopy(this.sourceBuffers))
      this.resetBuffers()
      this.packetCounter = 0
    }
  },
  initParser () {
    this.initBuffers()
    this.pushDataIntervalID = setInterval(this.pushData, 100)
  },
```

Listing 3.3: The parseData and pushData method

3.3.3 Extracting data for Visualization

The code that runs upon receiving the event in plot component is displayed in listing 3.4. When the plot component receives data it makes a new call to the `requestAnimationFrame` [3] with `updatePlot` as a callback function, telling the browser to make the callback on the next available screen repaint. This ensures a smoother update animation of the plot, as the callback is executed when the user's computer is ready to make changes to the screen, not after a set time as the `setTimeout`[4] method does.

The `updatePlot` method loops through the selected channels and unpacks the corresponding data from the `newData` object, in other words the passed copy of the `sourceBuffers`. Lastly it calls Plotly's `extendTraces` to plot the new points. Note: `indicesToUpdate` is simply an array of what indices in the plot to update, i.e if three channels are selected the array would be: [0, 1, 2]. This array is created every time the user selects or deselects a channel:

```
this.indicesToUpdate = [...Array(this.selectedChannels.length).keys()].
```

The logic is somewhat the same in the Visualizer (see appendix C).

```
  dataReceivedCallback (newData) {
    this.newData = newData
```

[3] https://developer.mozilla.org/en-US/docs/Web/API/window/requestAnimationFrame

[4] https://www.w3schools.com/jsref/met_win_settimeout.asp

```
      requestAnimationFrame(this.updatePlot)
    },
    updatePlot () {
      let newXValues = []
      let newYValues = []
      const newData = this.newData
      for (let i = 0; i <this.selectedChannels.length; i++) {
        const sourceChannelID = this.selectedChannels[i].id
        const newChannelData = newData[sourceChannelID[0]]
        newYValues.push(newChannelData.y_buffer[sourceChannelID[1]])
        newXValues.push(newChannelData.x_buffer)
      }
      this.$refs.plotlyDiv.extendTraces(
        {
          y: newYValues,
          x: newXValues
        },
        this.indicesToUpdate,
        this.maxPoints)
    }
```

Listing 3.4: The parsing methods in Plot component

3.4 Vue

This section describes how the frontend has been implemented in Vue, with a more detailed explanation of the flexible layout and how functionality has been separated.

The application's functionality is split into routes utilizing the Vue-router library. Vue-router [5] makes it easy to create navigable routes that remember their state for when the user re-visits the route. It also enables efficient code splitting [6] of the application. JavaScript web applications can become quite large when using a bundler, resulting in increased page load time. The Vue-router leverages this by making it trivial to lazy-load the route components (section 3.4.2).

3.4.1 Vuetify

The GUI itself utilizes Vuetify.js[7], which is a Material UI component framework for Vue. It currently provides over 80 specialized components such as dropdown menus, buttons and text fields. The framework follows Google's Material Specification[8], and leverages the tedious task of maintaining a consistent UI.

[5] https://router.vuejs.org/

[6] https://webpack.js.org/guides/code-splitting/

[7] https://vuetifyjs.com/en/

[8] https://material.io/design/

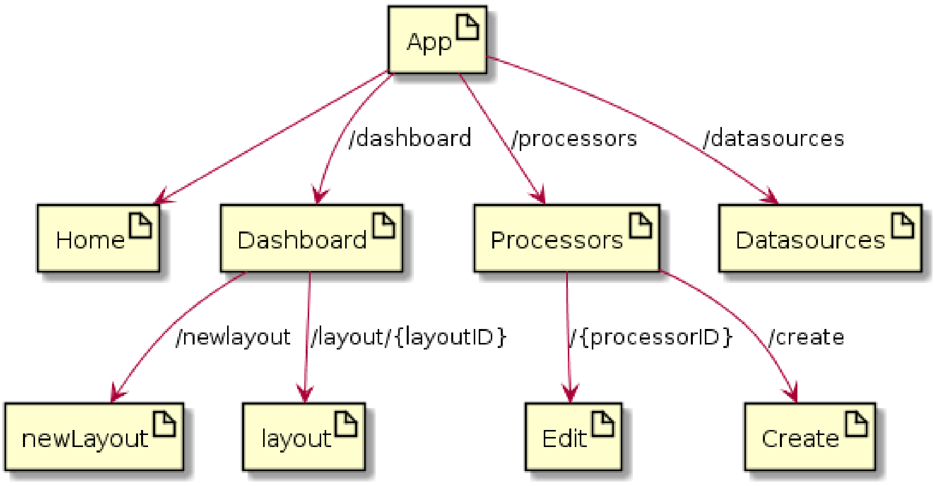


Figure 3.8: Overview of Application routes (dynamic routes are surrounded by {})

3.4.2 Lazy Loading

Lazy loading involves splitting the code into separate chunks that are loaded when needed. For instance instead of loading the whole web application, only the code needed for the destination page is loaded when the user navigates to the website. A simple lazy-loading of a component or JavaScript module can be done by combining Vue's async componenent feature[9] and webpack's code splitting feature[10] (listing 3.5 and 3.6)[11].

```
const Foo = () => Promise.resolve({ /* parameters for component */ }

import('./Foo.vue') // returns a promise
```

Listing 3.5: Async component and dynamic import

```
const Foo = () => import('./Foo.vue')

const router = new VueRouter({
  routes: {
    {path: '/foo', component: Foo}
  }

}
```

Listing 3.6: Combining Async component and dynamic import

[9] https://vuejs.org/v2/guide/components-dynamic-async.html#Async-Components

[10] https://webpack.js.org/guides/code-splitting/

[11] https://router.vuejs.org/guide/advanced/lazy-loading.html#grouping-components-in-the-same-chunk

Iterating on the above, all the views in the prototype are loaded using the `lazyLoadView` function in listing 3.7. It displays a progress component if the view takes longer than 200 milliseconds (ms) to load and a timed out component it loads for longer than the timeout field, set to 5000 ms. A similar function used for loading a component with a progress bar and timeout view can be found in appendix C.

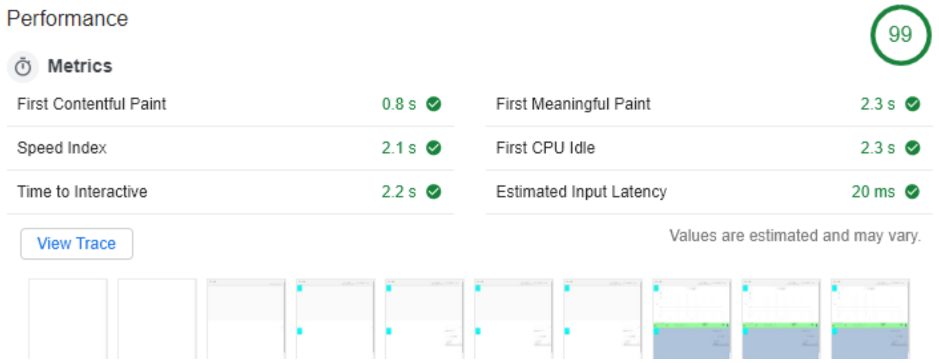
```
// Lazy-loads view components, but with better UX. A loading view
// will be used if the component takes a while to load, falling
// back to a timeout view in case the page fails to load. You can
// use this component to lazy-load a route with:
//
// component: () => lazyLoadView(import('@views/my-view'))
//
// NOTE: Components loaded with this strategy DO NOT have access
// to in-component guards, such as beforeRouteEnter,
// beforeRouteUpdate, and beforeRouteLeave. You must either use
// route-level guards instead or lazy-load the component directly:
//
// component: () => import('@views/my-view')
//
export function lazyLoadView (AsyncView) {
  const AsyncHandler = () => ({
    component: AsyncView,
    // A component to use while the component is loading.
    loading: require('../views/_loading').default,
    // Delay before showing the loading component.
    // Default: 200 (milliseconds).
    delay: 200,
    // A fallback component in case the timeout is exceeded
    // when loading the component.
    error: require('../views/_timeout').default,
    // Time before giving up trying to load the component.
    // Default: Infinity (milliseconds).
    timeout: 5000
  })

  return Promise.resolve({
    functional: true,
    render (h, { data, children }) {
      // Transparently pass any props or children
      // to the view component.
      return h(AsyncHandler, data, children)
    }
  })
}
```

Listing 3.7: Lazy-load View function

The benefit of lazy loading can be measured in a Lighthouse[12] audit. As seen in figures 3.9 and 3.10, the lazy-loaded version of the prototype loads significantly faster than the one without lazy-loading.

[12] `https://chrome.google.com/webstore/detail/lighthouse/blipmdconlkpinefehnmjammfjpmpbjk`



Performance

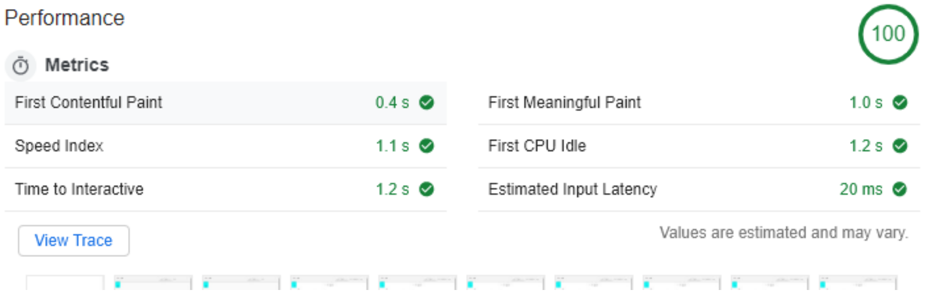
99

Metrics

First Contentful Paint	0.8 s ✓	First Meaningful Paint	2.3 s ✓
Speed Index	2.1 s ✓	First CPU Idle	2.3 s ✓
Time to Interactive	2.2 s ✓	Estimated Input Latency	20 ms ✓

Values are estimated and may vary.

Figure 3.9: Lighthouse audit of production build without lazy-loaded views



Performance

100

Metrics

First Contentful Paint	0.4 s ✓	First Meaningful Paint	1.0 s ✓
Speed Index	1.1 s ✓	First CPU Idle	1.2 s ✓
Time to Interactive	1.2 s ✓	Estimated Input Latency	20 ms ✓

Values are estimated and may vary.

Figure 3.10: Lighthouse audit of production build with lazy-loaded views

3.4.3 Layout Grid

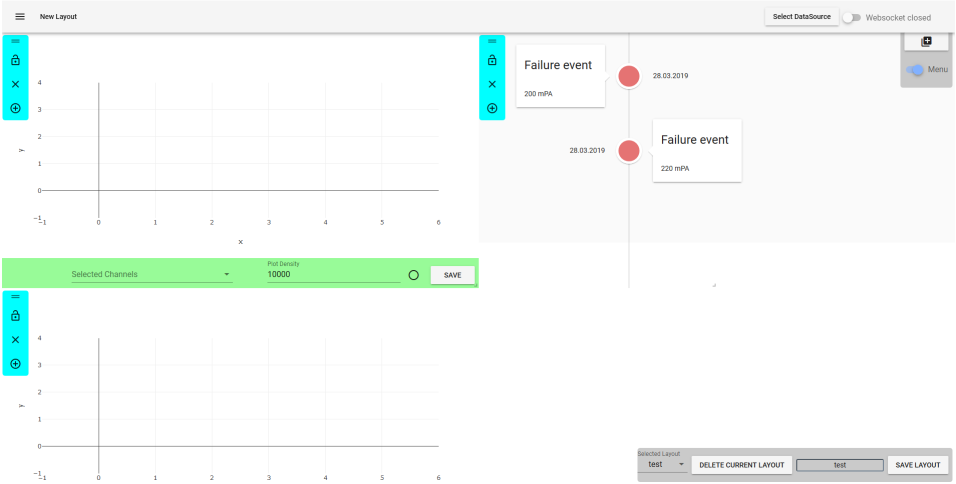
The prototype is designed to have a very flexible layout, with visualization components that can be resized, dragged and reordered in a grid. This way each user can customize their layout as they desire. This is achieved through the `LayoutGrid` component.

The `LayoutGrid` component holds moveable LayoutGridItems. The `LayoutGrid` uses a list of layout objects in order to create a complete layout of components. What component an object represents is inferred from its type field. The `LayoutGridItem` component is meant to be a simple wrapper for whatever component is put into it.

```

const gridLayout = [
  { 'x': 0, 'y': 0, 'w': 4, 'h': 1, 'i': '0', type: 'MarkerPlot', props:
      { title: 'CraneShort' } },
  { 'x': 0, 'y': 1, 'w': 4, 'h': 1, 'i': '1', type: 'Visualizer' }
]
```

Listing 3.8: Example of Layout



Plot Density
10000

Selected Layout
test

Figure 3.11: The dashboard with an example layout

The type field described in listing 3.8 is passed (as `compType`) to the computed property (listing 3.9) named `itemComp`. This coupled with Vue's dynamic component [13] allows for dynamic rendering of a component from the gridItems folder. Props can also be passed such as title to the MarkerPlot which is also depicted in the listing.

```
  computed: {
    itemComp () {
      const itemName = this.compType
      return lazyLoadComponent(import('./griditems/' + itemName + '.vue'))
    }
  }
```

Listing 3.9: The computed itemComp property

```
  <component :is="itemComp"
             class="no-drag"
             v-bind="properties"
  >
  </component>
```

Listing 3.10: LayoutGridItem: Dynamic component declaration

This means that adding a new `LayoutGridItem` is as simple as adding a new component in the projects griditems folder, and then using it by supplying a layout object as in listing 3.8 with a type field corresponding to the new components name. Note that the type field has to match the casing of the new component.

Depending on the responsiveness of the component's child component, a callback to resize might be needed. An example from the `PlotComponent` is depicted in listings 3.11

[13] `https://vuejs.org/v2/guide/components-dynamic-async.html`

and 3.12. Note that `controlsRow` is a v-layout component from Vuetify, which resizes correctly, so the callback function simply resizes the Plotly to have the same width.

```
<v-layout column v-resize="relayout">
```

Listing 3.11: PlotComponent: Resize event listening

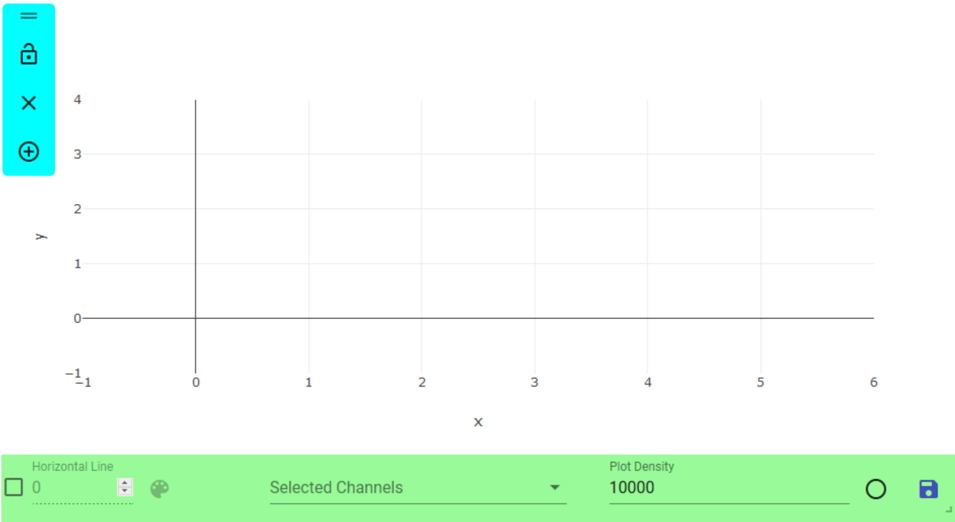
```
  // Set the plotly containers width to match controlsRow
  relayout () {
    let parentWidth = this.$refs.controlsRow.offsetWidth
    this.$refs.plotlyDiv.relayout({ width: parentWidth })
  },
```

Listing 3.12: PlotComponent: Resize callback function

3.5 Visualization

3.5.1 Plotly

Several plot libraries for JavaScript where reviewed before ending up on Plotly.js. Highcharts, Smoothie.js and Chartist.js were all suitable, but was decided against due to the lack of functionality or/and licensing compared to Plotly. Chartist, for instance, appears great for creating responsive, colorful, and visually striking plots, but not for visualizing live data. To implement Plotly in our project, a wrapper of Vue for Plotly was used [14]. It has been used in the PlotComponent as shown in listing 3.4



Horizontal Line: 0

Selected Channels

Plot Density: 10000

Figure 3.12: The Plot Component. The bar below and the blue controls on the left are additions created in Vue, the plot itself is from vue-plotly

[14] `https://github.com/statnett/vue-plotly`

3.5.2 Ceetron Cloud Components

The prototype utilizes Ceetron Cloud Components for 3D visualization, namely the Unstruct Surface Grid (USG) model functionality. The USG model does not require an additional server component, the Remote Model and Constant Remote model do,therefore the USG was deemed more suitable for a prototype. The models themselves are stored for each FMU on the backend and can be fetched on demand.

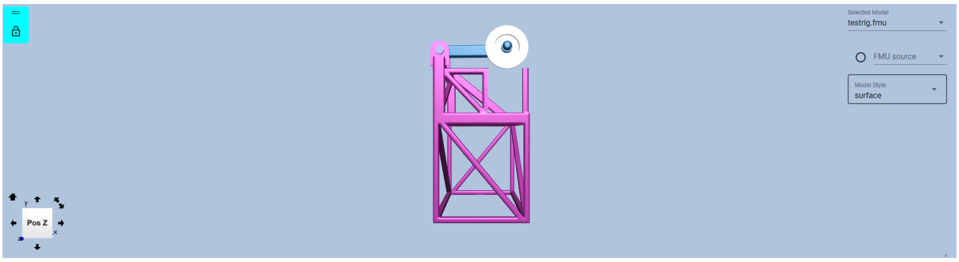

Selected Model: testrig.fmu
○ FMU source
Model Style: surface

Figure 3.13: The Visualizer component containing the Ceetron Canvas. (the menus in top right corner and the buttons in the top left corner are not part of the canvas)

3.6 Challenges

The largest obstacle during development was implementing a general parsing algorithm and a way to store the channel structure. Prior to utilizing `EventBus` to emit events, the data was parsed directly into each channel object in the store. By watching a counter value in the channel module, the plot component could then add the values currently in the channel to the plot. This method proved inefficient and resource demanding due to using the store as a buffer. The reason is that the Vuex store comes with some overhead for monitoring its state, which was not needed in our case and only brought more complexity with it.

Making the plot update smoothly was also a minor obstacle. During initial testing the browser froze over several times, due to a large amount of update calls being issued to Plotly.

Chapter 4

Results

This chapter highlights the GUI of the prototype, namely the different pages and visualization components implemented. As the prototype is a web application and navigation is done mainly through the GUI itself, the address bar has been omitted in all route screenshots except the landing page.

4.1 Graphical User Interface

As the application is split into routes, one can easily switch between different functionality. These routes can be navigated using the dropdown menu in the top left corner of the toolbar, as displayed in figure 4.1. From the toolbar it is possible to control what sources to subscribe to as well as closing or opening the WebSocket connection manually. For a detailed step by step instructions on how to use the app, see the user guide (Appendix A).

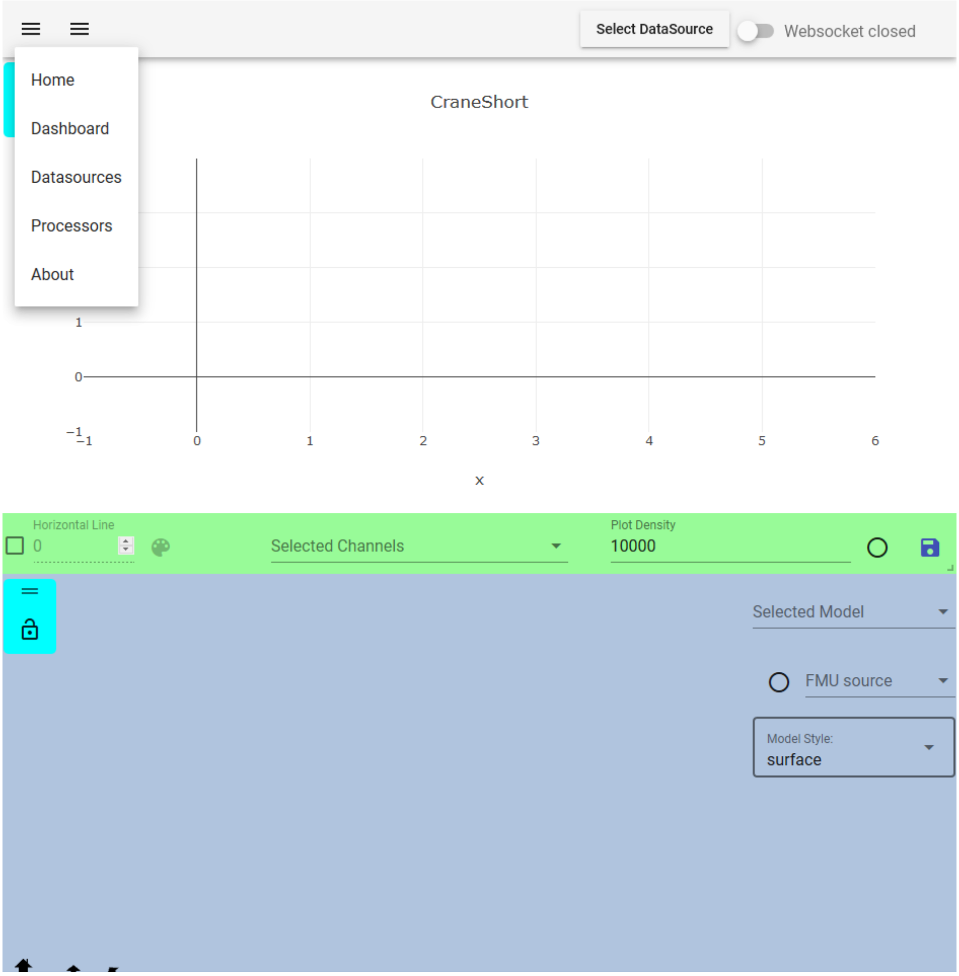
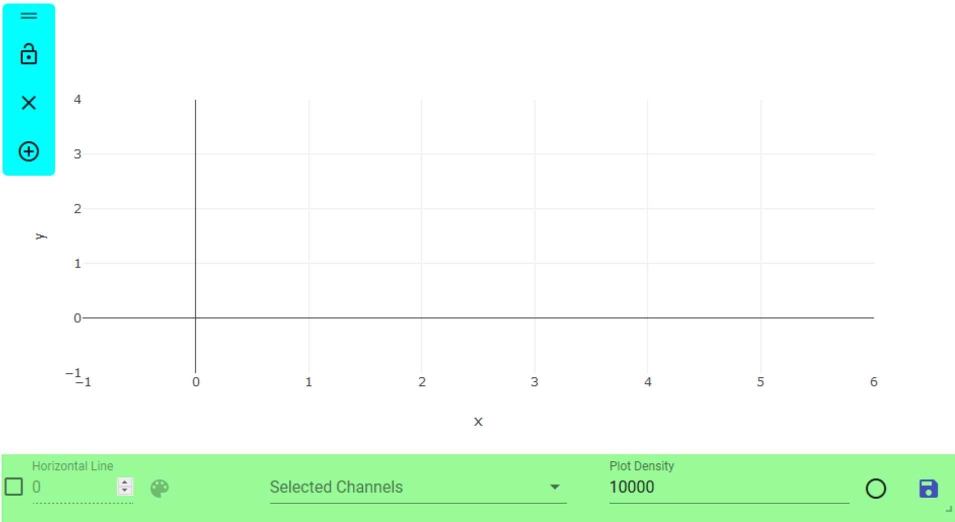


Figure 4.1: The landing page of the application: The home route

4.2 Visualization Components

4.2.1 Plot Component

The plot component is the key component for visualization (see figure 4.2). You can select what channels to plot, toggle plotting for the selected channels, and select a maximum value for how many points should be plotted before removing the oldest points. In addition, saving and downloading the current plot data as a CSV file is possible. The Plotly container inside the component provides functionality for zooming, hiding channels and saving as image to mention a few.



☐ Horizontal Line: 0

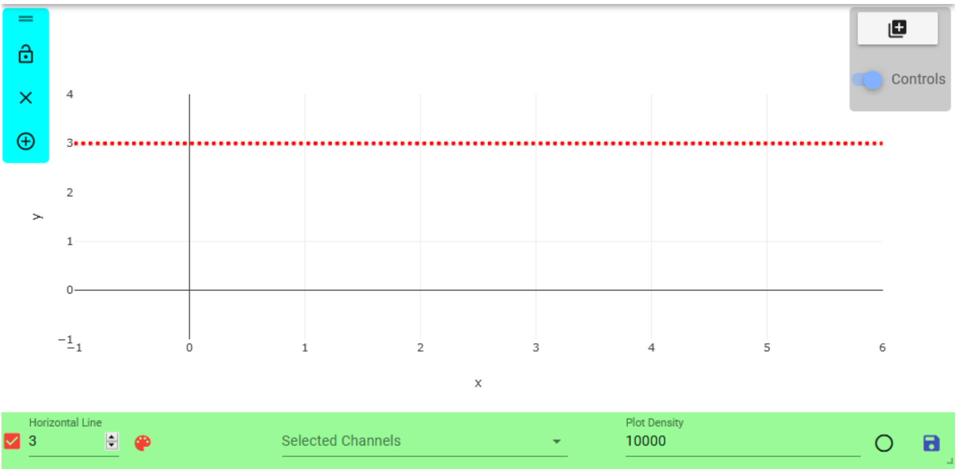
Selected Channels

Plot Density: 10000

Figure 4.2: PlotComponent: A simple component for plotting

Marker Plot

MarkerPlot (figure 4.3) is an extension of plot components functionality. It adds the possibility to create a dotted colored line on the plot for a specified value.



☑ Horizontal Line: 3

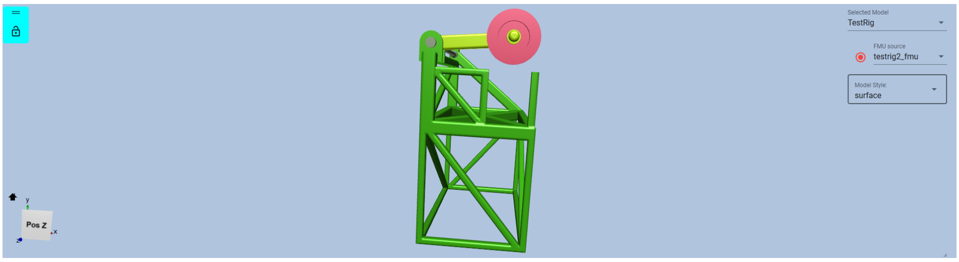
Selected Channels

Plot Density: 10000

Figure 4.3: MarkerPlot component, Extension of the plot component

4.2.2 Visualizer

The Visualizer utilizes Ceetron's Cloud Components [1] to display the model of the physical asset. Different model styles can be selected from the dropdown menu such as outline, surface and surface mesh.



Selected Model
TestRig

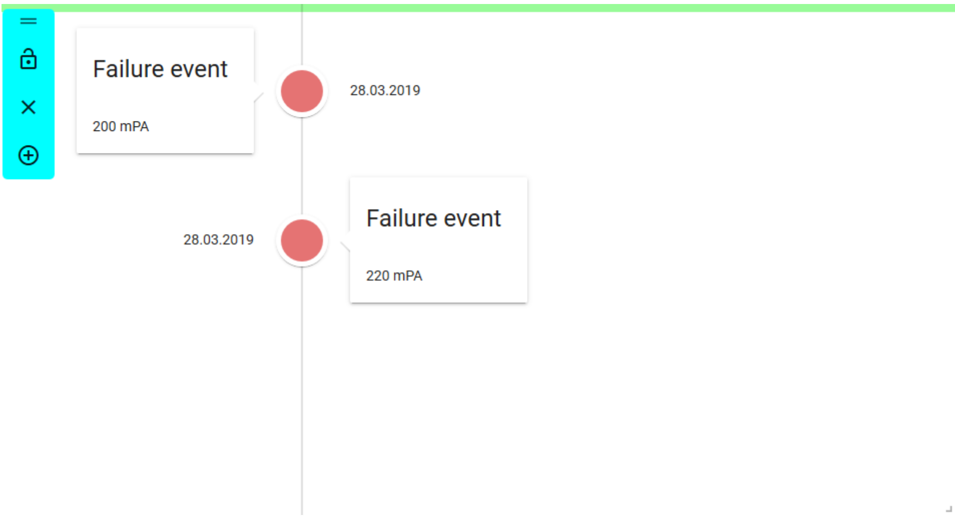
FMU source
testrig2_fmu

Model Style:
surface

Figure 4.4: The Visualizer component

4.2.3 Timeline

This component is left here as a concept, since it requires more logic both from the frontend and backend to be functional. The thought is to highlight failure events by tying it to a MarkerPlot (4.3) component, and then display events when the value selected passes a certain threshold. This can be done entirely in the client, but calculating these failure events on the backend would provide a more robust and desirable solution.



Failure event
200 mPA
28.03.2019

28.03.2019
Failure event
220 mPA

Figure 4.5: Example of the timeline component

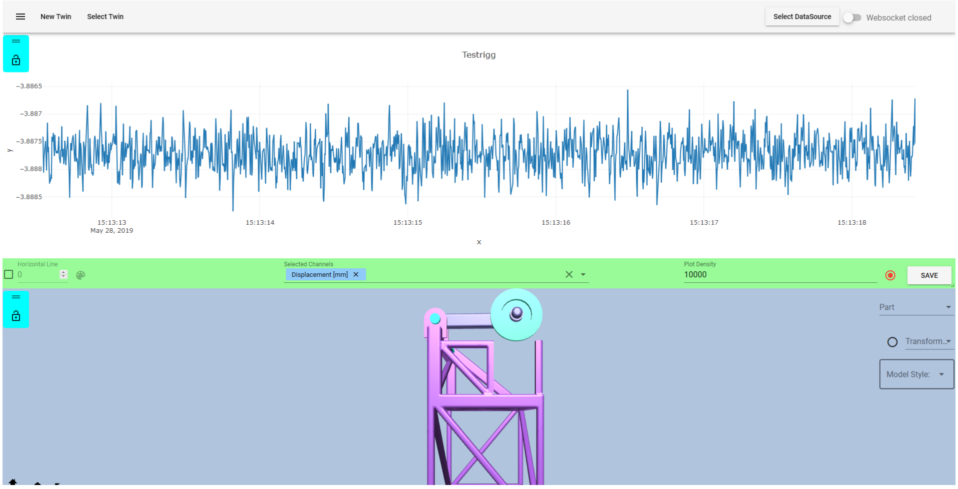
[1] https://ceetron.com/ceetron-cloud-components/

4.3 Views/Routes/Pages

This section highlights the several pages the user interface has been split into. All routes have the toolbar in common, with some slight variations on it depending on the route. The common options are selecting data sources, the navigation drop-down menu and toggling the WebSocket connection.

4.3.1 Home

A simple layout with a plot component (section 4.2.1) and a visualizer component (section 4.2.2. The intention is to have the most important functionality readily available, such that user can subscribe and view data in a few clicks. One can select data sources to subscribe to through the dialog displayed on pressing the button labeled "Select Datasource" in the toolbar.



Horizontal Line: □ 0

Selected Channels: Displacement [mm]

Plot Density: 10000

Figure 4.6: The Home Page, with the model of the testrig selected and displacement plot

The page supports relocation of components through drag and drop.

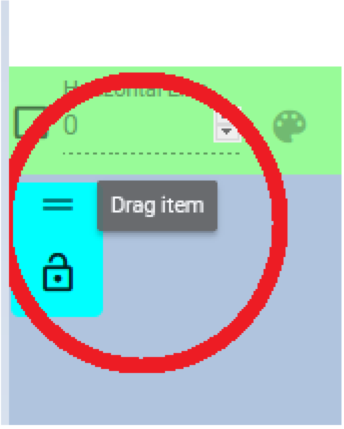
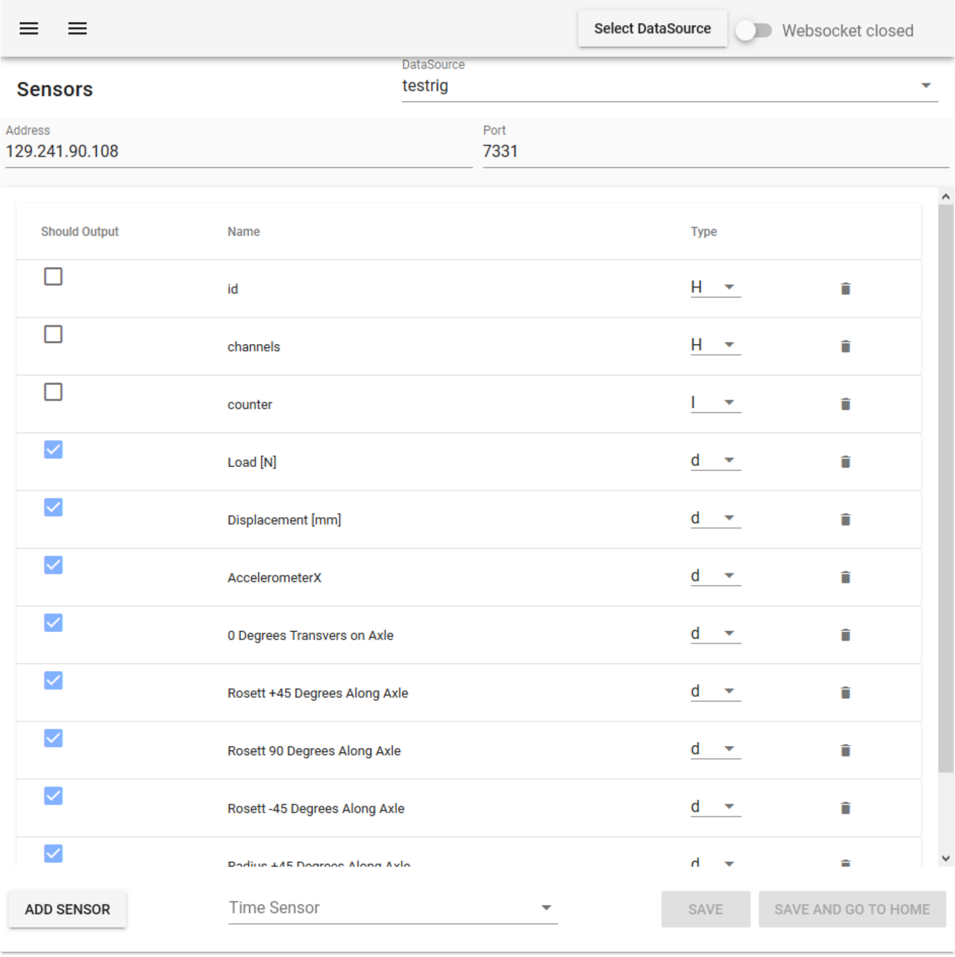


Figure 4.7: Component relocation using drag/drop

4.3.2 Data Sources

The datasources page is used to edit and create new data sources. A data source is a generalization of receiving data from a physical asset. It has a set of sensors with a name and a data type, currently allowing d, H, I, symbols for double, unsigned short and unsigned integer respectively[2]. To simplify the rest of the application's logic, especially in regards to the parsing logic explained in section 3.3.2, the user has to specify a sensor to be used as time input. The select output column in the table, specifies which channels that should be possible to subscribe to and view data from after creation. The setup in figure 4.8 results in the options depicted in figure 4.11.

[2] https://github.com/lyngklip/structjs



DataSource: testrig

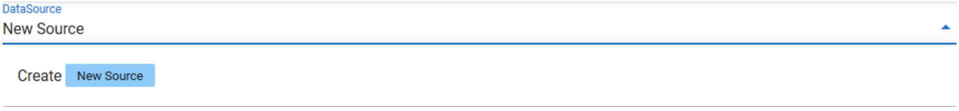
Address: 129.241.90.108

Port: 7331

Should Output	Name	Type
□	id	H
□	channels	H
□	counter	I
☑	Load [N]	d
☑	Displacement [mm]	d
☑	AccelerometerX	d
☑	0 Degrees Transvers on Axle	d
☑	Rosett +45 Degrees Along Axle	d
☑	Rosett 90 Degrees Along Axle	d
☑	Rosett -45 Degrees Along Axle	d
☑	Radius +45 Degrees Along Axle	d

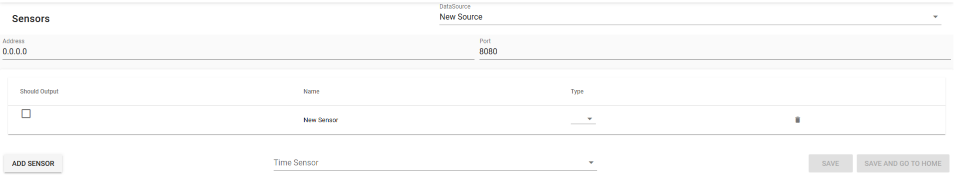
Figure 4.8: The landing page for the Datasources route

The selector labeled ”DataSource” is searchable and can also be used to create a new data source as shown in figures 4.9 and 4.10.



DataSource: New Source

Figure 4.9: Creating a new data source: Typing a new name and hitting enter will open the template for creating a new data source


Sensors

DataSource: New Source

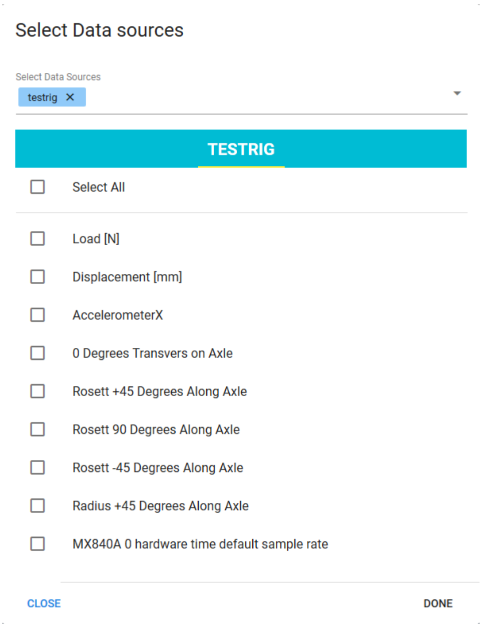
Address: 0.0.0.0

Port: 8080

Should Output	Name	Type
☐	New Sensor	

Time Sensor

Figure 4.10: The empty form for a new data source


Select Data sources

Select Data Sources: testrig ✕

TESTRIG

☐ Select All

☐ Load [N]

☐ Displacement [mm]

☐ AccelerometerX

☐ 0 Degrees Transvers on Axle

☐ Rosett +45 Degrees Along Axle

☐ Rosett 90 Degrees Along Axle

☐ Rosett -45 Degrees Along Axle

☐ Radius +45 Degrees Along Axle

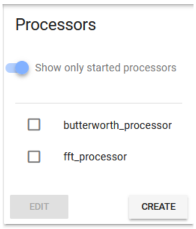
☐ MX840A 0 hardware time default sample rate

Figure 4.11: Subscribeable channels for testrig using the setup displayed in figure 4.8

4.3.3 Processors

The processor page is the landing page for handling processors. A processor in this case is a data processor which takes input from either other processors or a data source and transforms the input data. This data is made available for the user through outputs which can be subscribed to. The processor design is an attempt to generalize and standardize data extraction making it so if you support the ”standard” processor API, only small touches may occasionally be needed to adopt it for new types of processors.

This page allows further access for the user to either edit an existing process, or create a new one.



Processors

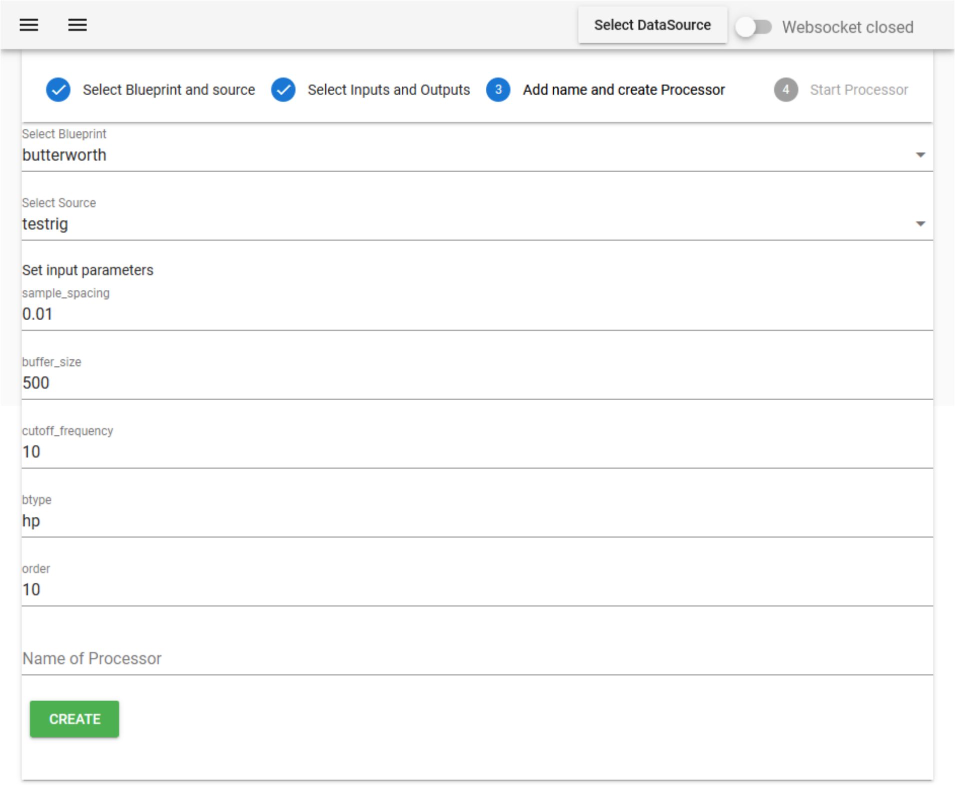
Show only started processors

□ butterworth_processor

□ fft_processor

Figure 4.12: Selecting a processor to edit or create a new one

Create Processor



Select DataSource
Websocket closed
Select Blueprint and source
Select Inputs and Outputs
3
Add name and create Processor
4
Start Processor

Select Blueprint: butterworth

Select Source: testrig

Set input parameters

sample_spacing: 0.01

buffer_size: 500

cutoff_frequency: 10

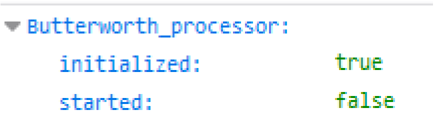
btype: hp

order: 10

Name of Processor

Figure 4.13: Create Processor page

Upon creating a processor, a request is sent to the backend to initialize the processor that has been setup. This may be confirmed by accessing the backend directly or by looking at the landing page for Processors.

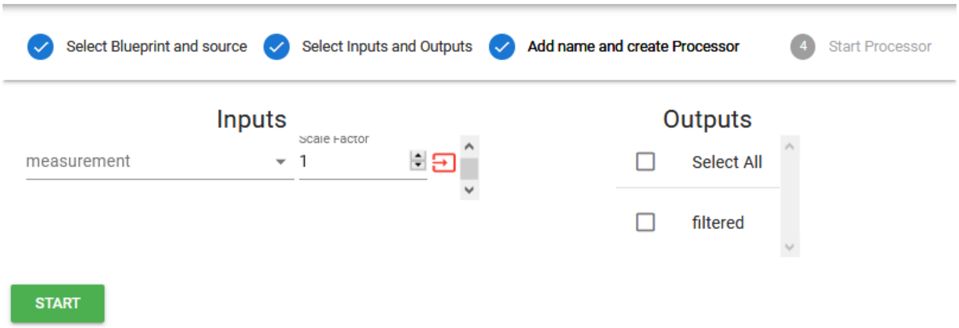


```
▼ Butterworth_processor:
    initialized:        true
    started:            false
```

Figure 4.14: JSON-object response from /processors/ showing that the processor has been created

Start Processor

As shown in the previous section, a created processor is not ready until it has been started. After pressing `Create` the user is shown start parameters as shown in figure 4.15. This is a simplified version of the User Interface (UI) shown in figure 4.18. as the only option from this page is to start the specific processor that has been created.



Select Blueprint and source
Select Inputs and Outputs
Add name and create Processor
4
Start Processor
Inputs
Outputs
START

measurement

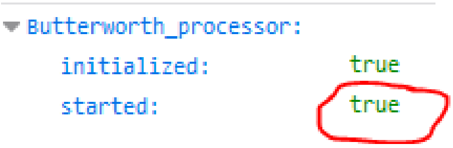
Scale Factor
1

□ Select All

□ filtered

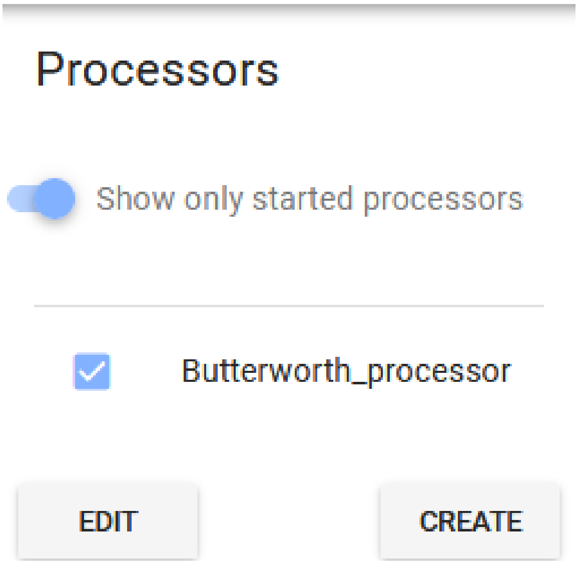
Figure 4.15: Final step of creating a processor

After pressing start, a request is sent and the processor should be running. This can again be confirmed in both the backend and in the UI directly.



```
▼ Butterworth_processor:
    initialized:        true
    started:            true
```

Figure 4.16: JSON-object response from /processors/ showing that the processor has been created and started



Processors

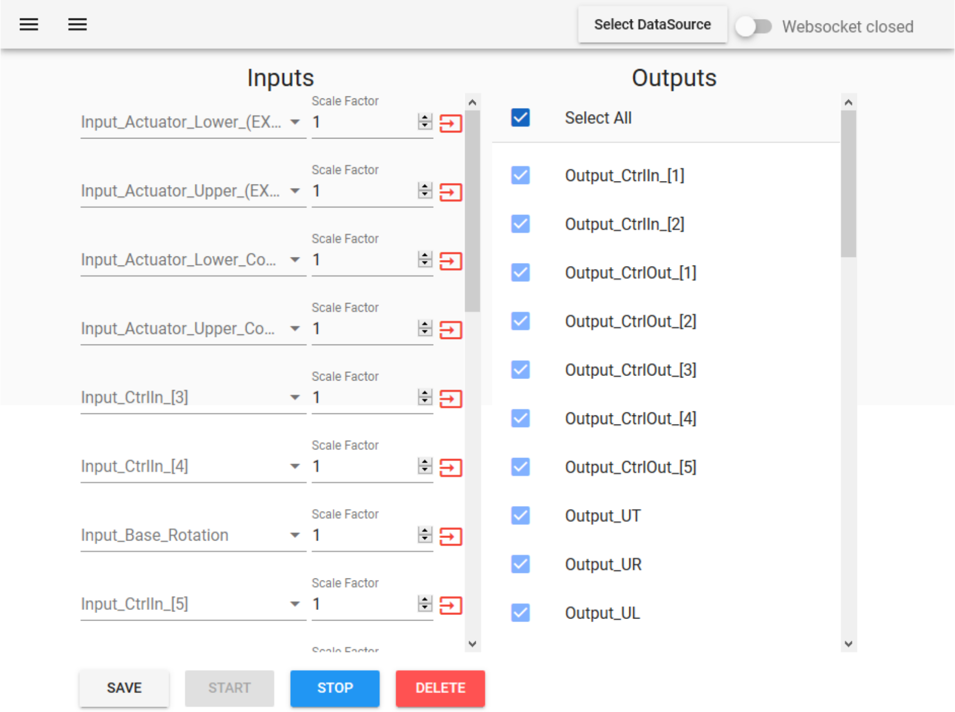
Show only started processors

☑ Butterworth_processor

Figure 4.17: Landing page of Processors now showing the started processor. Note the switch to show only started processor is on

Edit Processor

The edit processor page allows the user to edit inputs and outputs for a running processor. If the processor has not been started yet, it is possible to edit its start parameters from this page, similar to the ”Create Processor” page in figure 4.13. Running processors can be stopped or deleted as well.



Inputs

	Scale Factor
Input_Actuator_Lower_(EX...	1
Input_Actuator_Upper_(EX...	1
Input_Actuator_Lower_Co...	1
Input_Actuator_Upper_Co...	1
Input_CtrlIn_[3]	1
Input_CtrlIn_[4]	1
Input_Base_Rotation	1
Input_CtrlIn_[5]	1

Outputs

☑ Select All

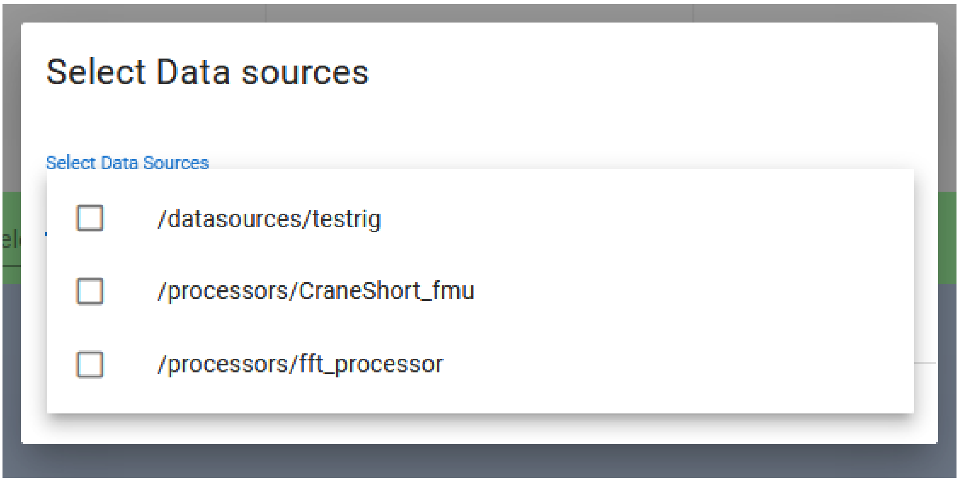
☑ Output_CtrlIn_[1]
☑ Output_CtrlIn_[2]
☑ Output_CtrlOut_[1]
☑ Output_CtrlOut_[2]
☑ Output_CtrlOut_[3]
☑ Output_CtrlOut_[4]
☑ Output_CtrlOut_[5]
☑ Output_UT
☑ Output_UR
☑ Output_UL

Figure 4.18: Screenshot: Editing a processor

Subscribe to topics

Topics is a term from Kafka[3] that is used in the backend as the API endpoint for any place data is published. This translates to any processor or datasource that is currently both created and started. To request data from topics, one has to run a subscribe request to the backend of the chosen topic. The select datasource button shown in figure 4.1 opens a view of all processors and datasources that are available for subscription. Upon opening the view, it is possible to subscribe to any multitude of outputs from different topics as seen in figures 4.19 and 4.20

[3] https://kafka.apache.org/



Select Data sources

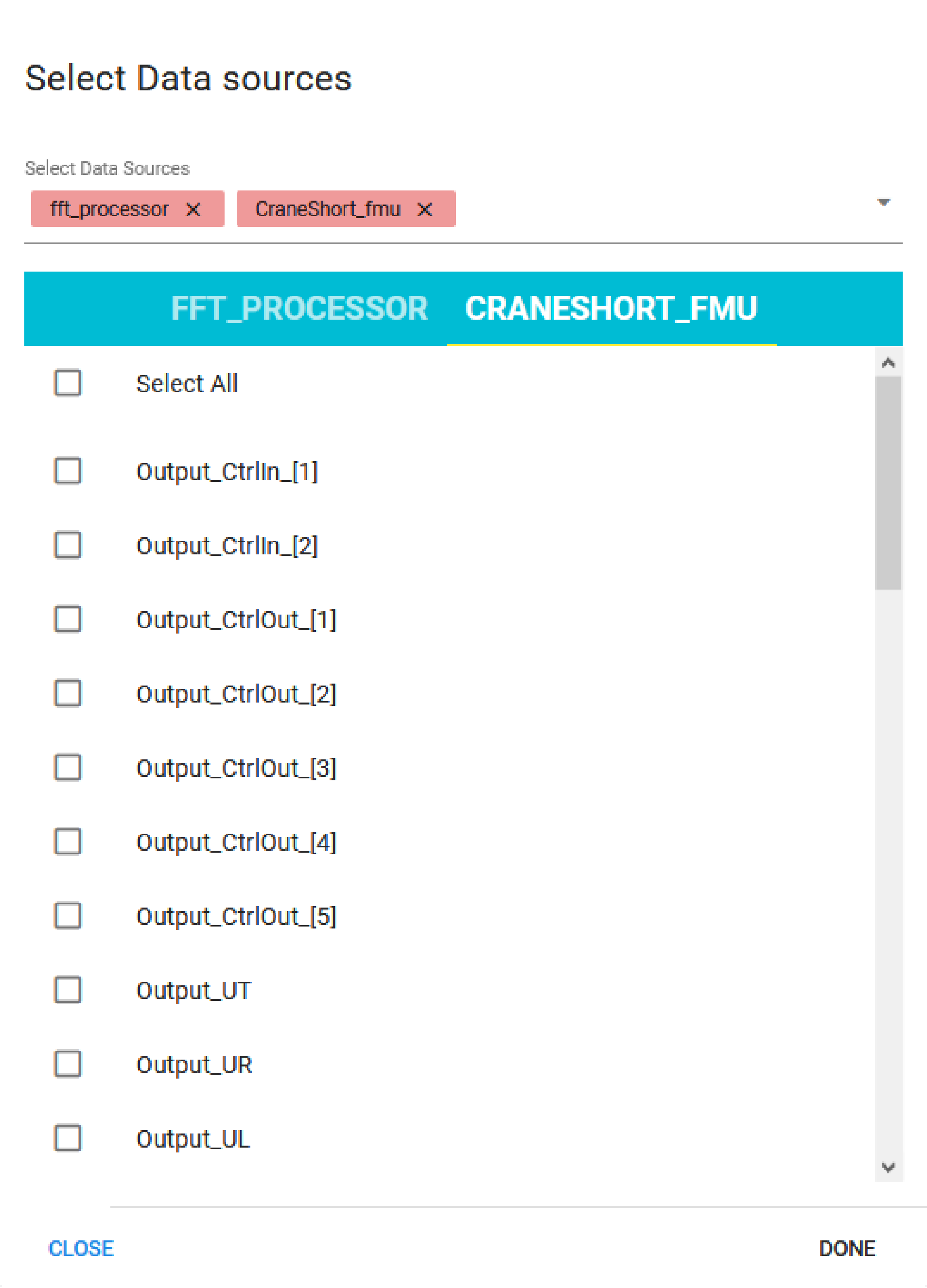
Select Data Sources

□ /datasources/testrig

□ /processors/CraneShort_fmu

□ /processors/fft_processor

Figure 4.19: Screenshot: List of topics one can select. After selecing a topic one can select which output(s) that is desired to visualize.



Select Data sources

Select Data Sources

fft_processor ✕ CraneShort_fmu ✕

- □ Select All
- □ Output_CtrlIn_[1]
- □ Output_CtrlIn_[2]
- □ Output_CtrlOut_[1]
- □ Output_CtrlOut_[2]
- □ Output_CtrlOut_[3]
- □ Output_CtrlOut_[4]
- □ Output_CtrlOut_[5]
- □ Output_UT
- □ Output_UR
- □ Output_UL

Figure 4.20: Screenshot: List of outputs from selected datasource. One can select outputs from different topics by pressing the corresponding tab(s)

4.3.4 Dashboard

The Dashboard page is meant for editing and creating custom layouts as well as viewing data in a custom layout. Layouts can be selected from the dropdown menu to the far left in figure 4.22. The current layout can be edited and then saved or deleted from the controls displayed in the same figure. To create a new layout, press the button to the right of the navigation menu, labeled ”New Layout” (figure 4.21).

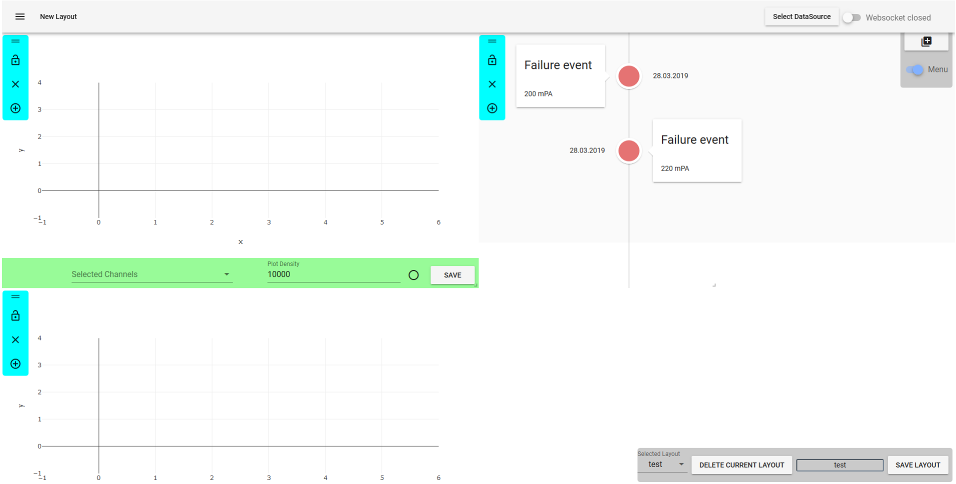


Figure 4.21: The Dashboard page, choose a layout or create a new one

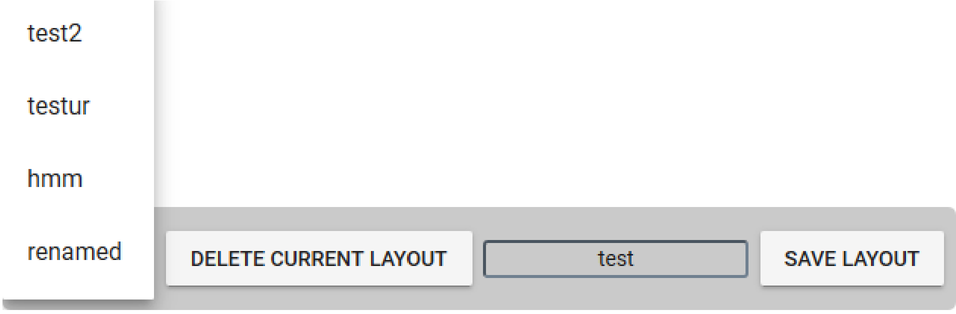


Figure 4.22: The Layout Controls for dashboard: Select, delete or save a layout

Chapter 5

Discussion and further work

In this chapter we discuss the prototype capabilities and several improvements and possible advisory tasks for further work. In particular, the chapter will focus on the choices made to facilitate further development of the project.

5.1 Cooperation with related projects

This project was as mentioned in the introduction launched alongside several dependent projects. In practice, during the project there's been close cooperation with the backend side of the CBMS project and some occasional interaction with the Crane Project. In the case of the Crane Project [8], one of the major reasons for the lack of significant cooperation is related to the stage each project was in.

On the other hand, the backend side can interact with the frontend side on a much earlier stage of development. However, two of the research goals included a suggestion to collect inputs from the crane and generator students. As this was difficult partly due to the progress of each project, instead these goals were satisfied in part by allowing the user to configure the interface.

5.2 Impact of choosing to use Vue.js

Vue has worked out very well and it is recommended to continue using Vue unless there is a need for advanced capabilities not provided by the framework. There are several reasons why we recommend Vue. It has offered us out of the box solutions that merely need to be modified for the current usage. Furthermore, it has detailed documentation to help people who do not have prior experience get started. However, while we think Vue is the best choice right now, future development regarding visualization applications might lead to a different conclusion.

5.3 Plotting

Compared to the implementation from Autumn 2018's, specialization project (Appendix B), the current prototype supports user selection of channels to plot. However, time series is still the only plot type available to the user. An FFT plot for instance, would require the values for a channel to be plot along a single "x-axis" showing frequencies. This is now fairly trivial to implement and is a natural expansion of the current plotting functionality.

On the other hand, it should be fairly simple to create a new component for plotting that utilizes another plotting library than Plotly. This comes as a result of a flexible and customizable layout coupled with a reasonably generalized parsing method. Extending the current plot implementation should also be relatively simple, which the MarkerPlot (figure 4.3) component is proof of.

5.3.1 Visualization/3D Model

Our goal with the development of the 3D model was to generalize the implementation from the specialization project. The resulting prototype, was hard-coded to show a real time 3D model of the Torsion Bar Suspension Rig. That prototype could not be used for different models (or FMUs).

The new implementation fulfills the same requirements as the prototype from Autumn 2018, but it is not hard-coded for the Testrig. It uses FMUs to show the output and in theory it only requires a model to show real time changes on any other Digital Twin. During testing however, we did not have access to the crane, nor does it send data over internet. Testing had to be done using output data from the Torsion Bar Suspension Rig (Appendix B) to move the crane model. While this did confirm that the Crane changes position by looking at the graphs, it was difficult to verify any movement on the 3D model of the Crane. Further testing is required to prove a moving model, either by tuning the output from the Torsion Bar Suspension Rig and applying it at specific inputs, or by setting up the Crane to give output.

The 3D Model visualization goal (section 1.3.1) is therefore not fully proven to apply to multiple models. To fully prove that the goal has been completed it would be necessary to prove the model's movement visually as well as in the graphs. What has been proven is that the model's parameters change as the input changes. This indicates that the 3D model accepts the current generic configuration for any model following the FMU standards. There is also currently no user-friendly way of creating simulated movement without getting output data from an asset. One way to do it would be to create a simulated data source that sends for example a sine time series. The effect of that would depend on the model, and as mentioned significant knowledge of the model would be necessary for it to have a use.

5.4 Bottlenecks/Improvements

This section outlays several areas of the prototype application that may be improved, not touched on previously. These areas will vary in importance and it is always important to not optimize prematurely. At the same time, it is important during development to spend the time to get a decent, working solution instead of creating temporary solutions at every turn.

5.4.1 Plotting Performance

During implementation of the curveplotting component, performance quickly deteriorated as more data points were rendered simultaneously. Research into the issue did not give clear-cut answers. Examples from other applications and visualizations revealed that the problem should not be lying in the amount of points rendered. However, there was still a clear difference between their visualization and ours. Their visualizations were based on static data points, not real time streamed data. Alternatives like Smoothie.js, a lightweight JavaScript graphing framework, was considered. To fix this, initially a lazy solution to limit the number of points on screen was chosen. However, this proved to be unsatisfactory.

It was discovered later on that the problem itself mainly was related to how data was plotted, not the framework. Initially the plot-call overwrote all current data points in the plot, simply adding the new points since last call improved performance significantly. There is now no significant lag using the system and the user can now set how many points should be retained in the plot. In other words, it is possible to set the points threshold unusually high and the application will start slowing down the browser, but this will vary depending on the computer's hardware. Therefore it's better to have an adjustable threshold, catering to each user's need.

5.4.2 Dataparsing

We discovered that issuing plot calls to Plotly was clearly the most resource demanding part of the prototype. The parsing logic remains relatively simple and with minimal optimization efforts as a result. Should it be discovered as a bottleneck later on, it would be advisable to use a web worker [1] to move work away from the main thread/process. A web worker is simply put another script running in the background, not interrupted by other scripts responding to user-interactions.

[1] https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API/Using_web_workers

Another drawback with the current data parsing implementation is the unnecessary load put on the Visualizer and Plot components. As mentioned in the data parsing section 3.3.2, the whole buffer of parsed data is sent to all these components where the desired data is filtered out. Emitting events and sending large amounts of data in this way is not ideal. There are multiple solutions to these issues, but figuring out a solution that actually results in a better data parser requires additional experience.

As a side note, it might help using WebAssembly[2] to implement a new parser, especially if higher performance is needed. It enables you to use already existing C/C++ code and compile it to modules usable in JavaScript, which is beneficial if there exists a C/C++ library that fits this usage. There is however a cost to issuing calls to these WebAssembly modules, which means that depending on the implementation the performance might not improve as much as desired.

5.5 Graphical User Interface

5.5.1 Home

The home page's design is in two parts as shown in 4.6. The objective of this page has been to provide the user a view that can show both the selected 3D model as well as a graph of real time data. One issue however is that depending on the 3D model, users may sometimes wish for more space dedicated for the 3D model. With the current implementation resizing the 3D visualizer is not supported and a user is limited to the space dedicated for the 3D model. A mitigating factor for this problem is that the visualizer component itself supports zoom, but ideally resizing should also be a supported feature.

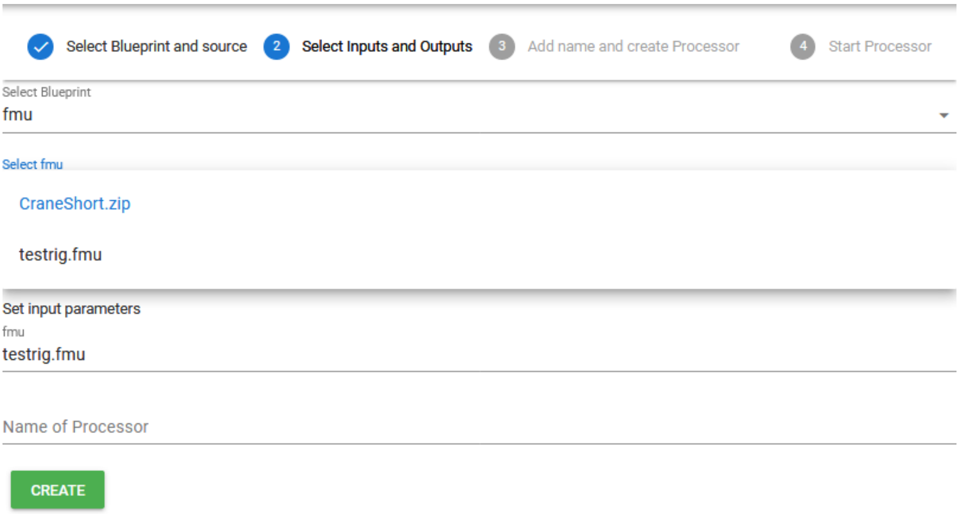
5.5.2 Processors

The Processors page has a simple design using a stepper function. We believe the steps itself of creating a process is shown in a good way on the processor page. Nonetheless, improvements can be made. At current implementation, documentation is only shown when the user is active in the input field. A more elaborate documentation might improve experience for both new and experienced users. Furthermore, if the type of processor the user wants to create is a FMU, an extra dropdown menu is currently required. The reason for this is to show the user which FMUs are currently available from the backend. The ”real” selection is performed in the field below where it says ”testrig.fmu” as default. This may be confusing for some users.

A more logical implementation would be for the dropdown menu to actually select which FMU to use.The reason that solution was not chosen is to fulfill of the major goals of the project, which was to create a generic configuration. The FMU is a special case,

[2] `https://webassembly.org/`

as the backend for it is slightly different to how for example different filters have been implemented.


Select Blueprint and source
2 Select Inputs and Outputs
3 Add name and create Processor
4 Start Processor

Select Blueprint
fmu

Select fmu
CraneShort.zip
testrig.fmu

Set input parameters
fmu
testrig.fmu

Name of Processor

Figure 5.1: Create Processor page during selection of FMU

5.5.3 Datasources

The Datasources page is a mostly static page where the user can start a data source from a template or create a new one. A problem with the current design is that the way to create a new data source is not immediately apparent to a new user. A create button or a stepper option similar to the one on the Processor page might make it clearer. The interface is missing the option to start and stop data sources. However, at its current stage there is very little need to create new sources, since the only available source has been the Torsion Bar Suspension Rig (Appendix B) therefore it has not been a priority.

5.6 Further Work

5.6.1 Historical data

Late in the development cycle, the ability to view historical data was added to the backend. Viewing historical data may be crucial for utilizing a Digital Twin to its fullest. Without a way of looking at old behaviour, it is difficult to find trends and utilize what is supposed to be one of Digital Twins biggest advantages - predictive maintenance.

5.6.2 Event Trigger

As mentioned in section 4.2.3, the Timeline component was made as a way to visualize events where the structure being monitored hit any critical modes. The back end has an event trigger processor implemented, but there is currently no automatic report generation. The easiest solution would be to do the automatic report generation entirely in the client. However, unlike the server, the user might not want to have the client running for longer periods of time, therefore it would be better to do it on the server side, as the server should always be up and running. The MarkerPlot component may be built upon to provide an intuitive interface for defining the thresholds that trigger these interactions.

5.6.3 Future Visualization Components

Once automatic report generation is implemented, there will be a need for new visualization components tailored to the data from such a report. The Timeline component (section 4.2.3) is a good start and can be further iterated upon. Another useful component would be a statistics component displaying a quick breakdown of the report and also more detailed information for a specific time period. Additionally it would be quite beneficial to have some sort of component that displays sensor status or sensor value color mapped to match it's value spectrum. In other words, green if the sensor is online or the value is within a defined accepted range. The suggestions have been summarized below:

- Status Bits: Red or Green boxes showing for instance overvoltage or sensor status.
- Statistics component, breakdown or report of a failure event
- Tailored visualization components for viewing generated reports

Chapter 6

Conclusion

A framework has been chosen for fast prototyping and a prototype has been developed. Plotly has been chosen as the library for plotting and a simple plot component has been implemented. The prototype has a flexible and customizeable layout for visualization and user-defined layouts can be created and saved. Raw and transformed real time data can be subscribed to and viewed in the user interface. The 3D visualization implemented can display the behaviour of the physical asset transmitting data.

The current prototype has not been tested sufficiently with any other digital twins than the Torsion Bar Suspension Rig. Consequently, it is difficult to say how well it will function with other twins, but the implementation should be general enough to support any digital twin, most likely without any tweaking. The solution for data parsing and extracting data for visualizations is not ideal, as a lot of unnecessary data gets passed through events. It's however without a doubt a feasible solution at the current time, but the data parsing solution might have to be tweaked or re-written when scaling up the prototype.

We were able to create a solution that fulfills some of the initial research goals as well as our main development objective which was to create a solution that had a user friendly interface and a generic configuration applicable to any Digital Twin. While there is room to grow in regards to both the UI, and testing on additional Digital Twins, current solution provides an adequate basis for having a Digital Twin in the Cloud.

Further development should focus on extending and testing the current prototype up against other digital twins as well as implementing new functionality. There is especially a need for a specialized GUI tailored to view historical data, and support for automatic report generation.

Bibliography

[1] Butterworth, S., 1930. On the theory of filter amplifiers. Wireless Engineer 7 (6), 536–541.
URL https://www.changpuak.ch/electronics/downloads/On_the_Theory_of_Filter_Amplifiers.pdf

[2] Cochran, W., Cooley, J., Favin, D., Helms, H., Kaenel, R., Lang, W., Maling, G., Nelson, D., Rader, C., Welch, P., 1967. What is the fast fourier transform? Proceedings of the IEEE 55 (10), 1664–1674.
URL https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=1447887

[3] Glaessgen, E., Stargel, D., 2012. The Digital Twin Paradigm forFuture NASA and U.S. Air Force Vehicles. NASA.

[4] Grieves, M., Vickers, J., 2017. Digital twin: Mitigating unpredictable, undesirable emergent behavior in complex systems. Transdisciplinary Perspectives on Complex Systems, 85–113.

[5] Howells, R., 02 2018. Should businesses be scared to meet their digital twin? Forbes.
URL https://www.forbes.com/sites/sap/2018/02/28/should-businesses-be-scared-to-meet-their-digital-twin/#75a2154363a1

[6] Howells, R., 2019. The digital twin effect: Four ways it can revitalize your business.
URL https://www.forbes.com/sites/sap/2018/06/22/the-digital-twin-effect-four-ways-it-can-revitalize-your-business/#704a494e5835

[7] Jensen, S. N., 6 2019. Building an extensible prototype for a cloud based digital twin platform. Master's thesis, NTNU.

[8] Johansen, C., 6 2019. Digital twin of offshore knuckle boom crane. Master's thesis, NTNU.

[9] Kritzinger, W., Karner, M., Traar, G., Henjes, J., Sihn, W., 2017. Digital twin in manufacturing: A categorical literature review and classification. Sciencedirect.

[10] Modelica, 2010. Tool news: Functional mockup interface (fmi).
URL https://www.modelica.org/publications/newsletters/2010-1/index_html#item8

[11] SAP, 2019. Bridge digital and physical worlds with digital twin technology.
URL https://www.sap.com/products/digital-supply-chain/digital-twin.html

[12] Taylor, D., 2017. Advantages of the digital twin in your manufacturing business.
URL https://community.plm.automation.siemens.com/t5/Digital-Twin-Knowledge-Base/Advantages-of-the-digital-twin-in-your-manufacturing-business/ta-p/432983

[13] Todd, F., 2019. Digital twin examples: Simulating formula 1, singapore and wind farms to improve results.
URL https://www.compelo.com/digital-twin-examples-formula1-singapore

[14] Websocket.org, (n.d). About html5 websockets.
URL https://www.websocket.org/aboutwebsocket.html

Appendices

Appendix A

User Guide

A user guide has been created showing the central actions a user can take. Each action has its own youtube video. The list below provides links to each video.

- How to use the Dashboard
- How to handle Datasources
- How to handle Processors
- How to subscribe and visualize data

Appendix B

Digital Twin Specialization Project Autumn, 2018

Project Thesis
Cloud Software For Digital Twin Modeling And Monitoring

Christian Johansen, Simen Norderud Jensen, Andreas Børhaug,
Odd Harald Sjursen Sande, Kia Brekke

Fall 2018

NTNU

Kunnskap for en bedre verden

Summary

The objective of the project is to explore and decide upon possible solutions to create a cloud-based digital twin solution with FEDEM software assisting in simulation and processing of FE models. The development of the project has been in cooperation with SAP and Ceetron, under supervision of MTP represented by Terje Rølvåg and Bjørn Haugen.

A user guide has been made to facilitate a quick-start in new environments, or as documentation together with the system overview. A prototype containing the key features required has been developed. The chosen solution is based on a local server receiving relevant data from a data acquisition system. The data is received by a server and analysed with FEDEM. The FE results are forward to a web application where motion of the asset is reproduced in a 3D model.

Contents

List of Figures

Listings

1 Introduction

The purpose of this project is to explore, test and evaluate possibilities regarding cloud based software solutions for Digital Twins using the FEDEM software for simulation and processing. The development of the project has been in cooperation with SAP and Ceetron, under supervision of MTP represented by Terje Rølvåg and Bjørn Haugen.

1.1 Background

The concept behind digital twins is to have a software replica of a physical object or process (physical twin) that can be used to better understand the system. However, the term is used loosely and its meaning varies depending on the physical twin it is representing. In this project a digital twin refers to a finite element (FE) model of a physical asset that through FE simulations, based on sensor data, can replicate the assets behaviour in real-time.

Multiple industries are looking to make use of digital twins because the development of the Internet of Things (IoT) has made sensors less expensive. The main use cases are predictive maintenance and monitoring of structural integrity. Benefits include better lifetime estimation, less need for on-site maintenance inspections and overall cost saving. To that purpose software companies are working on improving and creating new digital twin solutions to meet the demands of these industries. However, currently there are no non-proprietary digital twin solutions accessible. The Department of Mechanical and Industrial Engineering (MTP) at NTNU has a goal to develop a cloud based software solution that supports the digital twin applications both NTNU and SAP are currently developing. This project thesis lays the ground work for developing such software.

1.2 Problem Formulation

There are three main objectives in this project.

1. Write functional requirements for development of digital twin software. These should be based on hands-on experience and knowledge about technology.
2. Identify and select state-of-the-art software solutions. This includes exploration and evaluation based on usability, cost and ability to satisfy the functionality requirements.
3. Develop a prototype to test how well the requirements can be satisfied with the chosen solution.

This report will present the requirements, a system overview and a user manual on how to set up some of the parts. Furthermore, the results from prototypes developed will be displayed and explained. Finally there will be a discussion around technology options, challenges and further work.

2 Requirements

This section describes the different components needed for the digital twin cloud software, and the desired functionality that the end-user can experience.

Minimum Functionality Requirements

Physical twin

1. Measure relevant physical attributes
2. Transmit data to external server

Server

1. Receive measurement data
2. Sensor based real-time FE simulation and analysis
3. Transmit results to clients

Client

1. Be available through a browser
2. Visualise data from server in real-time
3. Save data from server to local file-system

Desired Functionality

- Real-time 2D plot of sensor data
- Real-time transformation of 3D model mirroring the physical twin
- Real-time video stream of the physical twin
- Stress analysis visualisation
- Fatigue analysis (S-N Curve)
- Possibility to save sensor values for further analysis
- Fast Fourier Transform
- Rewind in 3D visualisation and live-plot in case of interesting events

Hardware Components for Physical Twin

- Sensors
- Data Acquisition Board
- Computer(s)

3 System Overview

This section describes the system, including the physical asset, as is.

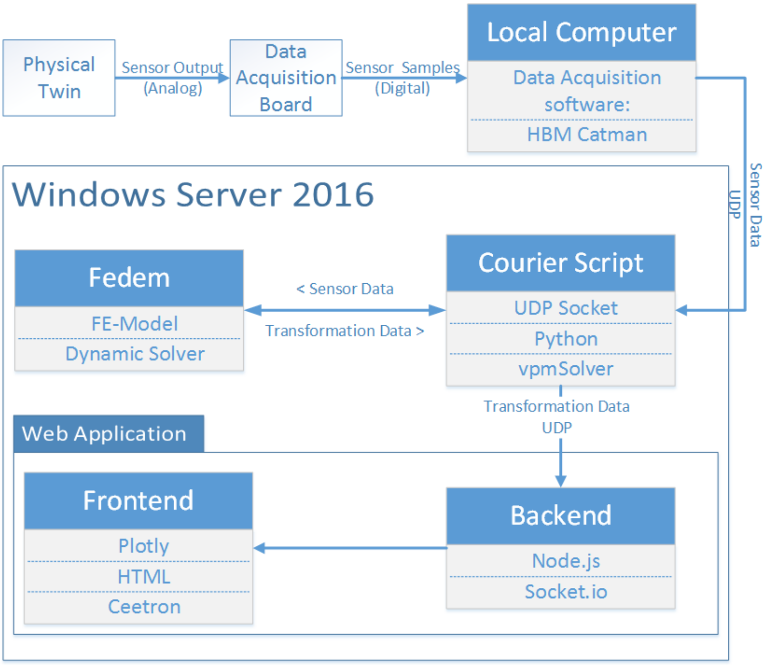


Figure 1: System overview

3.1 Physical Twin

The physical twin used in this project is the Torsion Bar Suspension Rig, which is equipped with eight sensors:

1. Load Cell
2. Displacement
3. Accelerometer
4. 0° Strain Gauge
5. +45° Rosette
6. 90° Rosette
7. −45° Rosette
8. +45° in Radius

The sensor values are sampled with an HBM data acquisition board and transferred to a computer located on the rig using an ethernet connection. More detailed information on the Torsion Bar Suspension Rig is included in appendix A.

3.2 Data Acquisition Software

The samples arriving to the computer on the Torsion Bar Suspension Rig are captured using the data acquisition software Catman. Catman is then used to map the samples values from voltage to the corresponding physical measurements. After the data is processed it is sent to the server using the remote connection option. This allows for sending data over the internet using the user datagram protocol (UDP). The remote connection option sends the data as a byte stream of 104 bytes for each time step. The mapping of the values to the bytes is shown in table 1.

Variable	**Bytes**
ID	[0:1]
Number of channels	[2:3]
Sequence counter	[4:7]
Time 1 - default sample rate	[8:15]
Time 1 - slow sample rate	[16:23]
Time 1 - fast sample rate	[24:31]
Load [N]	[32:39]
Displacement [mm]	[40:47]
AccelerometerX	[48:55]
0 Degrees Transvers on Axle	[56:63]
Rosett +45 Degrees Along Axle	[64:71]
Rosett 90 Degrees Along Axle	[72:79]
Rosett -45 Degrees Along Axle	[80:87]
Radius +45 Degrees Along Axle	[88:95]
MX840A_0 hardware time default sample rate	[96:103]

Table 1: Catman Output Format for `rigTimestamp.MEP`

3.3 Server

The server hosts the software used to represent the digital twin. For this project a virtual machine (VM) with Windows Server 2016 has been provided by NTNU IT. The following sections describe the components on the server in more detail.

3.3.1 Courier Script

The Python script (**RigSolver.py**) works as a courier between the physical twin, FEDEM and the web application. It receives sensor data from the physical twin and forwards this to FEDEM. When FEDEM is done with the dynamic analysis the results are returned and sent to the web application.

The code for **RigSolver.py** can be found in listing 1. The main functionality of the code is described below:

1. Initiate communication with the Torsion Bar Suspension Rig and the Web Application (Line 10-13)
2. Initiate communication with FEDEM solver (Line 16-20)
3. Listen for new sensor data from the Torsion Bar Suspension Rig (Line 26)
4. Unpack the sensor data to a FEDEM-friendly format (Line 30 and 33)
5. Solve dynamic analysis (Line 46)F
6. Get transformation data (Line 49)
7. Send transformation data and time stamp to the web application (Line 58)

```python
import struct
import socket

from fedem.fedemdll.vpmSolverRun import VpmSolverRun

# DT setup parameters
fedem_model_path = 'TestRig.fmm'

# Configure UDP Socket
PHYSICAL_TWIN_ADDRESS = ("0.0.0.0", 7331)
WEB_SERVER_ADDRESS = ("localhost", 8001)
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind(PHYSICAL_TWIN_ADDRESS)

# Initate VpmSolverRun object
with VpmSolverRun(fedem_model_path) as twin:

 # Initialization of solver (Needed for fedem functions)
 for n in range(2):
  twin.solveNext()

 # Continously receive data, solve, and forward result through UDP
 while(True):

  # Receive datagram
  data, _ = sock.recvfrom(32000)

  # Unpack displacement in mm from bytes 40:48 of datagram
  # Multiply with 0.001 to go from millimeters to meters
  displacement = 0.001*struct.unpack('<d',data[40:48])[0]

  # Rounding up the displacement value
  rounded_displacement=round(displacement, 4)

  # Print the sensor value
  print("Sensor␣value:␣␣{}␣meters".format(rounded_displacement))

  # Get current time. Needed for Fedem
  time = twin.getCurrentTime()

  # Connects sensor input to correct channel (Model spesific)
  # Set extfunc channel '2' as time 'time' with data 'rounded_displacement'.
  twin.setExtFunc(1, time, rounded_displacement)

  # Solves dynamic analysis for this time step based on sensor input
  twin.solveNext()
```

```

    # Get transformation data for all triads and parts
    transformationData = twin.save_transformation_state()

    # Retrieve timestamp from received datagram
    timestamp = data[96:104]

    # Assemble message with timestamp and transformationData
    message = timestamp + transformationData

    # Sends timestamp and transformation data to web client
    sock.sendto(message, WEB_SERVER_ADDRESS)
```

Listing 1: RigSolver.py

3.3.2 FEDEM

FEDEM is used to run dynamic analysis on the FE-model of the physical twin. The analysis is based on the sensor input from the physical twin and outputs transformation data for the triads and parts in the model. This is made possible by the external functions option in FEDEM. The output is an array containing the data type *double*. The format of the output array is shown in table 2.

Variable	**Element**
Time step	[0]
Time	[1]
Step length	[2]
Triad/Part	[3:17]
Triad/Part	[18:32]
⋮	⋮
Triad/Part	[End-14:End]

Table 2: Transformation Data array

Each sub array "Triad/Part" is on the format shown in table 3. "Object-Type" equals "1" for triads and "2" for parts.

Variable	**Element**
ObjectType	[0]
BaseID	[1]
Rotation Matrix $\begin{bmatrix} 2 & 3 & 4 \\ 5 & 6 & 7 \\ 8 & 9 & 10 \end{bmatrix}$	[2:10]
Translation vector $\begin{bmatrix} 11 \\ 12 \\ 13 \end{bmatrix}$	[11:13]

Table 3: Triad/Part Transformation Data array

FEDEM is also used to create the surface model used for 3D visualisation from a volume model of the Torsion Bar Suspension Rig.

3.3.3 Web Application

The `Node.js` script **index.js** (Listing 2) receives the transformation data from **RigSolver.py** (Listing 1) and parses it. It uses `socket.io` to send the relevant parsed data via WebSockets to **index.html** (Listing 3).

The code for **index.js** can be found in listing 2. The main functionality of the code is described below:

1. Initialise HTTP server (Line 10-12 and 17-20)
2. Serve files required by visualisation module (Line 14-15)
3. Parse and forward incoming data (Line 32-60)
4. Listen for new data (Line 63)

```
 1 // Import and initialise libraries
 2 const express = require('express');
 3 const app = express();
 4 const http = require('http').Server(app);
 5 const io = require('socket.io')(http);
 6 const dgram = require('dgram');
 7 const struct = require('python-struct');
 8 
 9 // Serve index.html when users visits the page
10 app.get('/', function(req, res) {
11     res.sendFile(__dirname + '/index.html');
12 });
13 
14 app.use('/ceetron', express.static('ceetron'));
15 app.use('/js', express.static('js'));
16 
17 // Start the http server for serving index.html
18 http.listen(1337, function(){
19     console.log('listening␣on␣*:1337');
20 });
21 
22 // Create socket listening for new data from solver
23 fedemSocket = dgram.createSocket('udp4');
24 
25 // Print to console when ready to listen for new data
26 fedemSocket.on('listening', function(){
27     const address = fedemSocket.address();
28     console.log('listening␣on␣' + address.address + ':' +
           address.port);
29 });
```

```

// Function for parsing new data from solver
fedemSocket.on('message', function(message, remote){

    // Extract timestamp from message
    const timestamp = struct.unpack('<d', message)[0]*1000;

    // Iterate over the bytes from the solver
    // Skip the timestamp and the first 3 doubles (24 bytes)
    // The bytes represents an array of doubles with a size of 8 bytes each
    // Iterate over the remaining doubles, 14 doubles at a time (112 bytes)
    for (var i = 32; i < message.length-111; i += 112) {
        // Read the baseId as the second of the 14 doubles
        const baseId = struct.unpack('<d', message.slice(i+8));
        if (baseId[0] === 318) {
            // Read the vertical displacement of the element from the message
            const displacement = struct.unpack('<d', message.slice(i + 96))[0];
            // Send the vertical displacement to the client
            io.emit('new_data', [timestamp, displacement]);
        } else if (baseId[0] === 316) {
            const t = struct.unpack('<12d', message.slice(i+16));
            const m = [
                t[0], t[1],  t[2],  0,
                t[3], t[4],  t[5],  0,
                t[6], t[7],  t[8],  0,
                t[9], t[10], t[11], 1
            ];
            io.emit('transformation', m);
        }
    }
});

// Start listening for new data from solver
fedemSocket.bind(8001, '0.0.0.0');
```

Listing 2: index.js

index.html (Listing 3) is used to plot the sensor data in the browser client and display a 3D model of the torsion bar suspension rig that replicates the movement of the asset. `Socket.io` is used to receive data sent by **index.js** (listing 2), `Ceetron Cloud Components` is used for 3D graphics and `plotly` is used for plotting.

The code for **index.html** can be found in listing 3. The main functionality of the code is described below:

1. Add toolbox for configuring 3D model draw style (Line 11-23)
2. Initialise connection to HTTP server (Line 34)
3. Initialise 3D visualisation from **usg.ts** (Listing 4) (Line 37-76)
4. Initialise plot (Line 85-98)
5. Plot live datastream (Line 103-115)
6. Update 3D model (Line 17-124)
7. Add save functionality (Line 126-152)

```
1 <!doctype html>
2 <html lang="en">
3 <head>
4     <title>Digital Twin</title>
5     <link rel="style.css">
6     <script src="/socket.io/socket.io.js"></script>
7     <script src="https://cdn.plot.ly/plotly-latest.js" charset="
        utf-8"></script>
8 </head>
9 <body style="margin:␣0;␣height:100vh;␣display:␣grid;␣grid:␣
    minmax(400px,␣50%)␣minmax(200px,␣50%)␣/␣minmax(400px,␣100%)">
10
11 <div style="display:␣flex">
12     <div id="chartContainer" style="width:␣100%"></div>
13     <div style="display:␣flex;␣flex-direction:␣column">
14         <button onclick="save()">Save</button>
15         <div style="flex-grow:␣1"></div>
16         <span>Model Style</span>
17         <button onclick="myApp.setDrawStyle('surface')">Surface<
            /button>
18         <button onclick="myApp.setDrawStyle('surface_mesh')">
            Surface Mesh</button>
19         <button onclick="myApp.setDrawStyle('outline_mesh')">
            Outline Mesh</button>
        <button onclick=”myApp.setDrawStyle(’lines’)”>Lines</button>
        <button onclick=”myApp.setDrawStyle(’points’)”>Points</button>
        <button onclick=”myApp.setDrawStyle(’outline’)”>Outline</button>
    </div>
</div>

<div style=”line−height:␣0”>
    <canvas id=”CeetronCanvas”></canvas>
</div>

<script src=”ceetron/require.js”></script>
<script>

    // Initialise connection to server
    var socket = io();

    // Initialise USG module
    var myApp = null;
    require([”js/usg”], function(appModule) {
        myApp = appModule.startApp(”CeetronCanvas”);

        // Retrieve arm geometry
        var oReq = new XMLHttpRequest();
        oReq.onload = armLoaded;
        oReq.open(”get”, ”/js/arm.json”, true);
        oReq.send();
    });

    function armLoaded(e) {
        // Send arm geometry to visualiser
        data = JSON.parse(this.responseText);
        myApp.addArmGeometry(data);

        // Retrieve torsion rod geometry
        var oReq = new XMLHttpRequest();
        oReq.onload = rodLoaded;
        oReq.open(”get”, ”/js/TorsionRod.json”, true);
        oReq.send();
    }

    function rodLoaded(e) {
        // Send torsion rod geometry to visualiser
        data = JSON.parse(this.responseText);
        myApp.addRodGeometry(data);

        // Retrieve frame geometry
        var oReq = new XMLHttpRequest();
        oReq.onload = frameLoaded;
        oReq.open("get", "/js/Frame.json", true);
        oReq.send();
    }

    function frameLoaded(e) {
        // Send frame geometry to visualiser
        data = JSON.parse(this.responseText);
        myApp.addFrameGeometry(data);
    }

    // Store reference to container for plot
    var graphContainer = document.getElementById('chartContainer');

    // Container for displacement plot data
    var displacements = {x:[[]], y:[[]]};

    // Initialise plot
    Plotly.newPlot(
        graphContainer,
        [{y:[]}],
        {
            title: 'Displacement',
            xaxis: {
                title: 'Displacement (mm)'
            },
            yaxis: {
                title: 'Timestamp'
            }
        },
        {responsive: true}
    );

    // Counter for how many data points has been received
    var dataRecievedCount = 0;

    // Update plot with new data for every 100 new data points
    socket.on('new data', function(msg){
        displacements.x[0].push(new Date(msg[0]));
        displacements.y[0].push(msg[1]);
        // If 100 data points recieved since last update
        if (dataRecievedCount++ % 100 === 0) {
            // Remove points received more than 1000 points ago
            displacements.x[0] = displacements.x[0].slice(-100000);
            displacements.y[0] = displacements.y[0].slice(-100000);
            // Update plot
            Plotly.restyle(graphContainer, displacements);
        }
    });

    // Update transformation of model for every 100 new data points
    socket.on('transformation', function(msg){
        if (dataRecievedCount % 100 === 0) {
            if (myApp !== null) {
                myApp.updateDisplacement(msg);
            }
        }
    });

    // Create download dialog for currently plotted data
    function save() {
        var saveData = "Timestamp,␣displacement(mm)\r\n";
        for (var i = 0; i < displacements.x[0].length; i++) {
            saveData += displacements.x[0][i].valueOf() + ",␣" + displacements.y[0][i] + "\r\n"
        }
        download(saveData, "twin_" + new Date().toISOString() + ".csv", "text/csv");
    }

    // Downloading data to a file
    function download(data, filename, type) {
        var file = new Blob([data], {type: type});
        if (window.navigator.msSaveOrOpenBlob) // IE10+
            window.navigator.msSaveOrOpenBlob(file, filename);
        else { // Others
            var a = document.createElement("a"),
                url = URL.createObjectURL(file);
            a.href = url;
            a.download = filename;
            document.body.appendChild(a);
            a.click();
            setTimeout(function() {
                document.body.removeChild(a);
                window.URL.revokeObjectURL(url);
            }, 0);
        }
    }
</script>
</body>
</html>
```

Listing 3: index.html

The module **usg.ts** (Listing 4) is used to visualise movement in the torsion bar suspension rig through a 3D model. The model is translated and rotated according to the transformation given in the update method (which is calculated in the FEDEM solver). The drawing style of the geometry is changed through the setDrawStyle method.

The code for **usg.ts** can be found in listing 4. The main functionality of the code is described below:

1. Import Ceetron USG module used for creating, transforming and displaying the geometry. (Line 1)
2. Initialisation (Line 4-10)
3. Define class used to handle the visualisation (Line 13-141)
4. Initialise the visualisation state (Line 16-50)
5. Create the geometry representing the torsion arm (Line 65-70)
6. Create the geometry representing the torsion rod (Line 72-86)
7. Create the geometry representing the frame (Line 88-94)
8. Display statistics about the geometry in bottom left corner (Line 96-111)
9. Update arm geometry according to transformation (from FEDEM) (Line 113-127)
10. Change the drawing style of the visualisation (Line 130-141)

```
import * as cee from ”../ceetron/CeeCloudClientComponent”;

// Initialiser for Ceetron module of application
export function startApp(canvasElementId: string): App {
    let canvas = document.getElementById(canvasElementId);
    if (!(canvas instanceof HTMLCanvasElement)) {
        throw(”Could␣not␣get␣canvas␣element”);
    }
    return new App(canvas);
}

// Class containing Ceetron Cloud Client Component state
export class App {

    // Ceetron Cloud Client Component state
    private cloudSession: cee.CloudSession;
    private view: cee.View;
    private model: cee.usg.UnstructGridModel;
    private state: cee.usg.State;

    // Canvas containing visualisation
    private canvas: HTMLCanvasElement;

    constructor(canvas: HTMLCanvasElement) {
        this.canvas = canvas;

        // Initialise Ceetron Cloud Client Component
        this.cloudSession = new cee.CloudSession();
        let viewer = this.cloudSession.addViewer(canvas);
        if (!viewer) {
            throw(”No␣WebGL␣support”);
        }
        this.view = viewer.addView();
        this.model = new cee.usg.UnstructGridModel();
        this.view.addModel(this.model);
        this.state = this.model.addState();
        this.state.geometry = new cee.usg.Geometry();

        // Hide infoBox initially
        this.view.overlay.infoBoxVisible = false;

        // Listen for resize events
        window.addEventListener('resize', () => this.
            _handleWindowResizeEvent());

        // Manually run resize function once
        this._handleWindowResizeEvent();

        // Update view every browser frame
        window.requestAnimationFrame((time: number) => this._myAnimationFrameCallback(time));
    }

    // Adjust view dimension (called when window is resized)
    private _handleWindowResizeEvent() {
        let canvasWidth = window.innerWidth;
        let canvasHeight = this.canvas.parentElement.offsetHeight;
        this.cloudSession.getViewerAt(0).resizeViewer(canvasWidth, canvasHeight);
    }

    // Update view (called every browser frame)
    private _myAnimationFrameCallback(highResTimestamp:number) {
        this.cloudSession.handleAnimationFrameCallback(highResTimestamp);
        window.requestAnimationFrame((time: number) => this._myAnimationFrameCallback(time));
    }

    // Create the torsion arm geometry
    addArmGeometry(data) {
        let geometry = this.state.geometry.addPart();
        geometry.mesh = new cee.usg.Mesh(data.nodeArr, data.elementTypeArr, data.elementNodeIndexArr);
        geometry.settings.color = new cee.Color3(.1,.1,.1);
    }

    // Create the torsion rod geometry
    addRodGeometry(data) {
        let geometry = this.state.geometry.addPart();
        geometry.mesh = new cee.usg.Mesh(data.nodeArr, data.elementTypeArr, data.elementNodeIndexArr);
        geometry.settings.color = new cee.Color3(.8, .8, .8);

        // Transform to global coordinate system
        const c = cee.Mat4.fromElements(
            1, 0, 0, -0.02407066,
            0, 1, 0, -0.02722985,
            0, 0, 1, 0.27199998,
            0, 0, 0, 1
        );
        this.state.setPartTransformationAt(1, c);
    }

    // Create the frame geometry
    addFrameGeometry(data) {
        let geometry = this.state.geometry.addPart();
        geometry.mesh = new cee.usg.Mesh(data.nodeArr, data.elementTypeArr, data.elementNodeIndexArr);
        geometry.settings.color = new cee.Color3(.2, .2, .7);
        this.showStatistics(this.state.geometry);
    }

    private showStatistics(geometry) {
        // Generate statistics on geometry
        let nodeCount = 0;
        let elementCount = 0;
        for (let part of geometry.getPartArray()) {
            nodeCount += part.mesh.nodeCount;
            elementCount += part.mesh.elementCount;
        }

        // Log generated statistics
        console.log("Initial␣state␣loaded,␣nodeCount=" + nodeCount + ",␣elementCount=" + elementCount);

        // Draw generated statistics in bottom right corner
        this.view.overlay.infoBoxVisible = true;
        this.view.overlay.setInfoBoxContent(`Elements: ${elementCount} elements \nNodes: ${nodeCount} nodes`);
    }

    updateDisplacement(transformationMatrix: number[]) {
        // Create Ceetron matrix from transformation data
        const m = cee.Mat4.fromArray(transformationMatrix);

        const localToGlobalTransformation = cee.Mat4.fromElements(
            1, 0, 0, -0.00000001,
            0, 1, 0, -0.00000000,
            0, 0, 1, 0.00199997,
            0, 0, 0, 1
        );
        const transformation = cee.Mat4.multiply(m, localToGlobalTransformation);

        // Apply transformation to armGeometry
        this.state.setPartTransformationAt(0, transformation);
    }

    // Change drawing style for geometries
    setDrawStyle(ds: string) {
        const geometry = this.model.getStateAt(0).geometry;
        for (let part of geometry.getPartArray()) {
            if      (ds === "surface")          part.settings.drawStyle = cee.usg.DrawStyle.SURFACE;
            else if (ds === "surface_mesh")         part.settings.drawStyle = cee.usg.DrawStyle.SURFACE_MESH;
            else if (ds === "outline_mesh")         part.settings.drawStyle = cee.usg.DrawStyle.SURFACE_OUTLINE_MESH;
            else if (ds === "lines")                part.settings.drawStyle = cee.usg.DrawStyle.LINES;
            else if (ds === "points")               part.settings.drawStyle = cee.usg.DrawStyle.POINTS;
            else if (ds === "outline")              part.settings.drawStyle = cee.usg.DrawStyle.OUTLINE;
        }
    }
}
```

Listing 4: usg.ts

4 Web Application Prototype

The web application prototype is available at `http://tvilling.digital:1337` when connected to the NTNU network. Figure 2 shows a digital representation of the physical asset.

The upper half of the web browser consists of a live 2D plot of the torsion arm displacement. Extra functionality for the plot window such as zoom and pan can be found in the toolbox at the top-right of the plotting window. To the right of the toolbox there is a save button. By pressing this button you can download a CSV-file to your own computer containing the previous 100 000 data points and their associated timestamp. The timestamp is saved using the Unix time standard, which is number of seconds elapsed since 1st of January 1970. A visualisation of the torsion bar suspension rig is shown on the bottom half. A model of the torsion bar suspension rig moves according to the movement of the torsion arm calculated in FEDEM. It is possible to change the zoom and camera position by scrolling and dragging, and the draw style can be changed with the buttons above on the right.

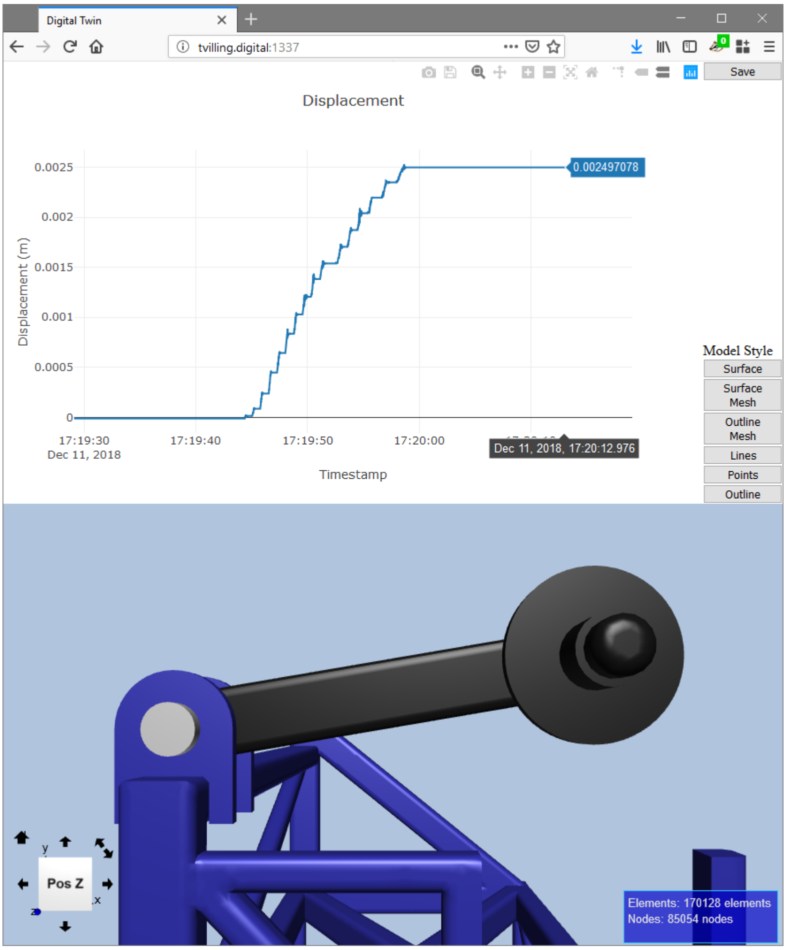


Figure 2: Digital Twin

5 User guide

This section is a user guide on how to setup the digital twin cloud software with the Torsion Bar Suspension Rig. Each subsection describes a part of the system and how to configure it.

5.1 Ethernet

The computer on the Torsion Bar Suspension Rig needs to be connected to the data acquisition board and with the WIN.NTNU.NO network through a common ethernet connection. This can be achieved by using an ethernet switch. On the Torsion Bar Suspension Rig the ethernet connection is already set up.

5.2 Catman configuration for Torsion Bar Suspension Rig

NOTE:

- The username and password for the computer is written on top of it
- Catman must be run in Administrator mode for the remote connection to work properly

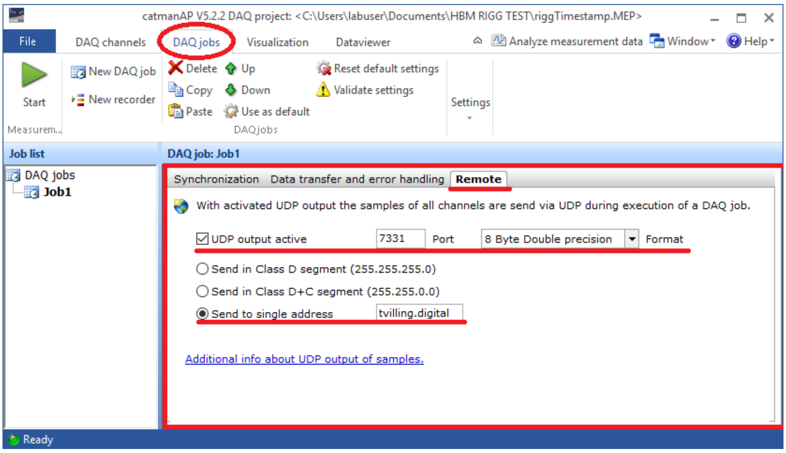
5.2.1 Initialisation and Calibration

First navigate to the directory: *C:\Users\labuser\Documents \HBM RIGG TEST* and run the file `riggTimestamp.MEP`. This will open Catman with the correct setup. Next you need to calibrate the sensors. The calibration procedure is explained in the Torsion Bar Suspension Rig Manual found in appendix A.
NOTE: This manual is designed for the project file `RIGGOPPSETT.MEP` and some of the functionality it describes is not available for the project file `riggTimestamp.MEP`.

5.2.2 Remote Connection

To set up the remote connection to the server you need to:
Go to **DAQJobs** in the header > Choose **Advanced** and then **Remote**.
The window should look like figure 3.
In this window you need to:

- Check the option for *UDP output active*
- Fill in the server port number (7331)
- Choose the format *8 Byte Single precision*
- Choose *Send to single address* and fill in the IP-address of the server (tvilling.digital or 10.212.25.104)



With activated UDP output the samples of all channels are send via UDP during execution of a DAQ job.

☑ UDP output active 7331 Port 8 Byte Double precision Format

○ Send in Class D segment (255.255.255.0)

○ Send in Class D+C segment (255.255.0.0)

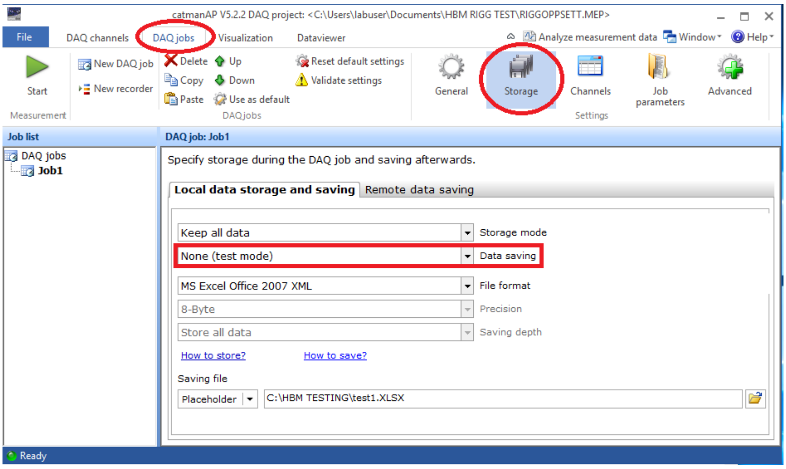
● Send to single address tvilling.digital

Additional info about UDP output of samples.

Figure 3: Remote Connection in Catman

5.2.3 Storage

If not specified, Catman will locally store all data recorded. To avoid this: Go to **DAQJobs** in the header > Choose **Storage** and then **Local data storage and saving** > Click on **Data saving** and choose *None (test mode)*. The window should look like figure 4.



Specify storage during the DAQ job and saving afterwards.

Storage mode: Keep all data

Data saving: None (test mode)

File format: MS Excel Office 2007 XML

Precision: 8-Byte

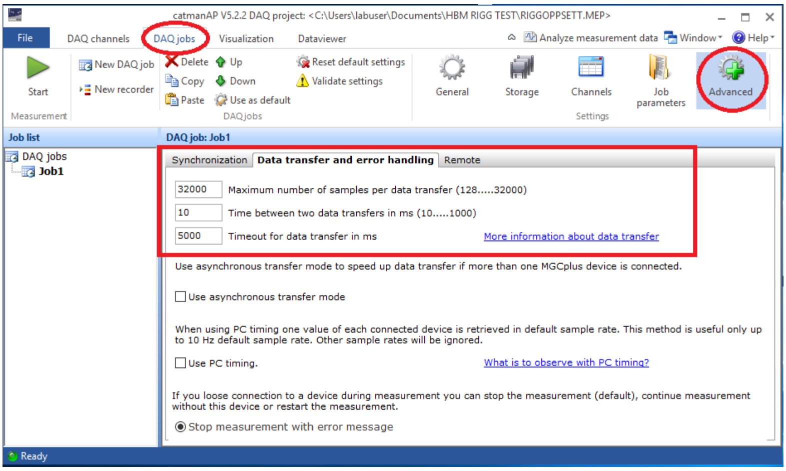
Saving depth: Store all data

Saving file: Placeholder C:\HBM TESTING\test1.XLSX

Figure 4: Storage management in Catman

5.2.4 Transfer

The size and frequency of data transmissions can be managed. To do this you need to:
Go to **DAQJobs** in the header > Choose **Advanced** and then **Data transfer and error handling**.
The window should look like figure 5.



32000 Maximum number of samples per data transfer (128.....32000)

10 Time between two data transfers in ms (10.....1000)

5000 Timeout for data transfer in ms More information about data transfer

Use asynchronous transfer mode to speed up data transfer if more than one MGCplus device is connected.

□ Use asynchronous transfer mode

When using PC timing one value of each connected device is retrieved in default sample rate. This method is useful only up to 10 Hz default sample rate. Other sample rates will be ignored.

□ Use PC timing. What is to observe with PC timing?

If you loose connection to a device during measurement you can stop the measurement (default), continue measurement without this device or restart the measurement.

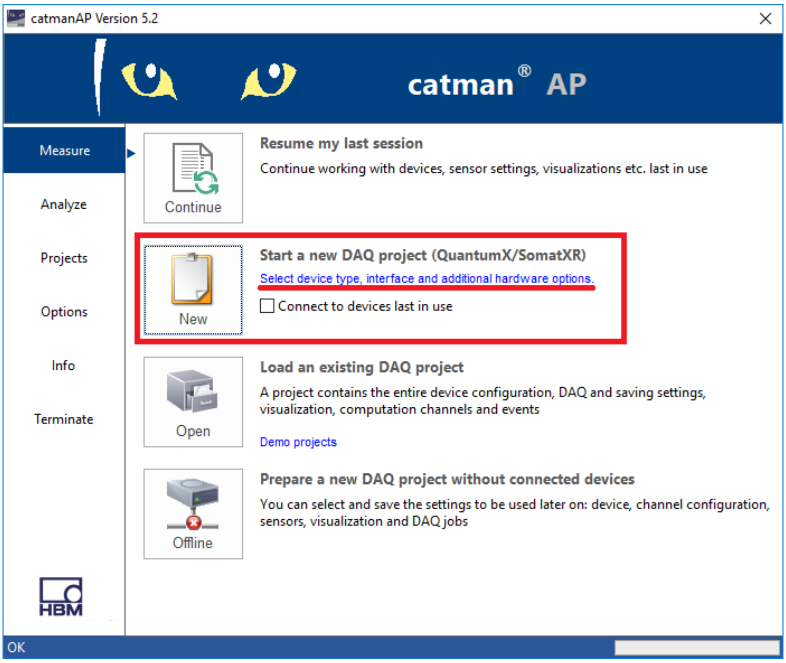
● Stop measurement with error message

Figure 5: Transfer management in Catman

5.2.5 Create New Project (OPTIONAL)

To create a new project file (`.MEP`): open Catman AP (See figure 6) > Click on ”**Select device type, interface and additional hardware options**” > In this new window (See figure 7) Click on **Hardware time channels**, choose **Create hardware time channels** and click **OK** > Click on **Start a new DAQ project** > In this new window (See figure 8) click **Connect**.

Note that the data acquisition board must be connected to the sensors and the computer with Catman for this to work properly.



Resume my last session

Continue working with devices, sensor settings, visualizations etc. last in use

Start a new DAQ project (QuantumX/SomatXR)

Select device type, interface and additional hardware options.

□ Connect to devices last in use

Load an existing DAQ project

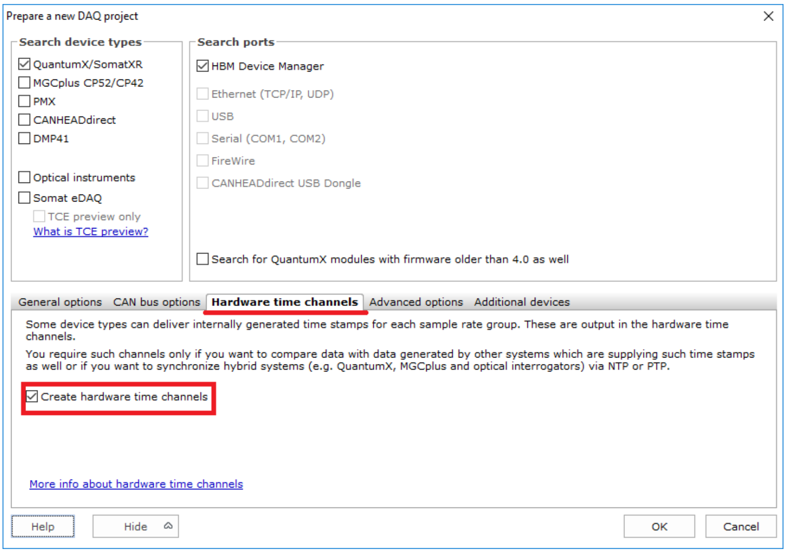
A project contains the entire device configuration, DAQ and saving settings, visualization, computation channels and events

Demo projects

Prepare a new DAQ project without connected devices

You can select and save the settings to be used later on: device, channel configuration, sensors, visualization and DAQ jobs

Figure 6: New Project in Catman AP (1)



Prepare a new DAQ project

Search device types

☑ QuantumX/SomatXR
□ MGCplus CP52/CP42
□ PMX
□ CANHEADdirect
□ DMP41

□ Optical instruments
□ Somat eDAQ
□ TCE preview only
What is TCE preview?

Search ports

☑ HBM Device Manager

□ Ethernet (TCP/IP, UDP)
□ USB
□ Serial (COM1, COM2)
□ FireWire
□ CANHEADdirect USB Dongle

□ Search for QuantumX modules with firmware older than 4.0 as well

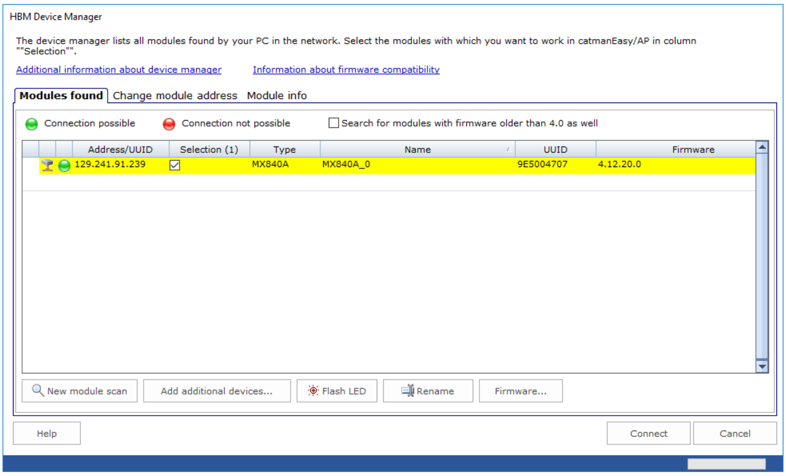
Some device types can deliver internally generated time stamps for each sample rate group. These are output in the hardware time channels.

You require such channels only if you want to compare data with data generated by other systems which are supplying such time stamps as well or if you want to synchronize hybrid systems (e.g. QuantumX, MGCplus and optical interrogators) via NTP or PTP.

☑ Create hardware time channels

More info about hardware time channels

Figure 7: New Project in Catman AP (2)



HBM Device Manager

The device manager lists all modules found by your PC in the network. Select the modules with which you want to work in catmanEasy/AP in column ""Selection"".

Additional information about device manager   Information about firmware compatibility

Modules found | Change module address | Module info

● Connection possible   ● Connection not possible   □ Search for modules with firmware older than 4.0 as well

Address/UUID	Selection (1)	Type	Name	UUID	Firmware
129.241.91.239	☑	MX840A	MX840A_0	9E5004707	4.12.20.0

Figure 8: New Project in Catman AP (3)

5.3 Server

This section explains how to set up the cloud software on the server from scratch.
NOTE:

- Before you start you need to install Python and Node with NPM on the server.
- The servers Firewall may have to be configured to allow for UDP communication on ports 8001 and 7331, and TCP communication on port 1337.
- To gain access to the folder *DT_Example* you must sign a non-disclosure agreement (NDA) with SAP.
- The *udpplotter* can be retrieved from the Github repository: "https://github.com/simennj/udpplotter". It is currently private because of license restrictions.
- You need to start a job in Catman for the plotting to commence. To do so simply press the `Start`-button found in the top-left corner in the Catman window.

- Catman has to be set up to send the data to the new server, see 5.2.2.

Procedure:

1. Install all necessary Python packages. A complete list of the packages can be found in Appendix B.
2. Navigate to the directory of the *udpplotter* folder (see notes) in the terminal and type `npm install`.
3. Run the web application by typing `node index.js` in the terminal.
4. In a new terminal window navigate to the *DT_Example* folder (see notes) and run the command: `python RigSolver.py`

The server should now be set up properly. If you have configured the rest of the system according to sections 5.1 and 5.2 you should now be able open the web application if you type `localhost:1337` in the server's web browser. If the firewall is set up correctly, the web application should then be available on `<server address>:1337` on other computers.

6 Discussion and Evaluation

6.1 Technologies

6.1.1 Data Acquisition System

A data acquisition system consists of three parts: sensors, data acquisition boards and data acquisition software. The sensors capture and quantify a physical phenomena through a voltage which is then sampled by a data acquisition board. The samples are read by the data acquisition software and the voltage value is translated into a corresponding engineering unit. Examples of DAQ software is Catman by HBM and LabVIEW by National Instruments.

At the beginning of the project, a previous setup was available using a data acquisition board from HBM and Catman, and there was no immediate need for changes. However, in the early stages of the project the license for Catman expired. An alternative to purchasing license based data acquisition software is to develop an in-house software solution. In addition to cost savings, an in-house software solution is more transparent and can offer more control than Catman. The possibility of an in-house software solution was explored and a prototype was developed. This prototype was able to retrieve raw data from the data acquisition board.

While the prototype is able to retrieve the raw data from the board, there are still two obstacles. The first is interpreting data from the data acquisition board; what values are received and which sensor they originate from. The second is mapping the voltage values to an engineering unit. Both of these issues require access to documentation of the sensors and the data acquisition board in order to be solved.

At that point there were two clear ways forward, either continue working on the prototype or renewing the Catman license. After discussing the options with Terje Rølvåg it was decided to renew the Catman licence. This was due to time constraints and uncertainty of successfully finishing the prototype without access to the proper documentation. However, we would like to stress that the digital twin cloud software is not locked to Catman.

The choice of data acquisition solution should be assessed in the case of instrumenting a new physical asset. As long as the solution is able to send the measured values as doubles through UDP, it should be compatible with the digital twin solution. This could potentially reduce costs spent on hardware and software licenses.

6.1.2 Server Architecture

One of the sub-goals of the project is to be able to host digital twin software externally in an application. During development, two options have been considered: Self hosting and renting space at cloud computing service companies such as Amazon Web Services (AWS), Microsoft Azure or NTNU IT. Hosting at a cloud service required less work than self hosting and was therefore preferable. After researching the cloud services we discovered that while AWS and Azure are expensive, hosting at NTNU IT would not cost anything and still provide the necessary features. The chosen solution was to host a local VM provided by NTNU IT.

6.1.3 Data Communication

Two protocols for sending raw data over the internet were considered: Transmission Control Protocol (TCP) and User Datagram Protocol (UDP). An assessment was done in order to choose which would be the best for the digital twin cloud software. For this project, the most important difference between the protocols is that UDP simply sends the packets without checking if they are received while TCP re-sends the packet if it is not received. It was decided after discussions with Terje Rølvåg that in the case of a lost packet it would be better to continue transmitting new packets instead of halting the stream to re-transmit the lost packet. The need for a high throughput with as little delay as possible is deemed more important than the occasional loss of a packet. Since the latter has no noticeable effect on the simulation, UDP was chosen as the data communication protocol.

6.1.4 Visualisation tools

There is a large number of visualisation tools for web development available that offer 2D graphics, both open-source and closed-source. However, for the digital twin cloud software we needed a tool that could make a 2D-plot of a live data-stream, without stuttering. To avoid losing time on issues regarding licenses, open-source libraries were prioritised. After research and testing, the JavaScript library `Plotly` was chosen. Other tools were reviewed, but due to the successful implementation of `Plotly` we chose not to go any further with other options.

The number of visualisation tools for web development that offer 3D graphics is more limited. There are a few open-source libraries such as `BabylonJS` and `Three.js` specifically made for 3D graphics, but they do not support FE-models. We were introduced to the company Ceetron by Terje Rølvåg which offers several tools for visualisation and post-processing of FE-models. A meeting was arranged with Ceetron and SAP in late November to discuss how we could use Ceetron software in our web application to visualise and animate the FE-model results. For this purpose it was suggested that we make use of the `Unstruct Surface Grid` (USG) model functionality found in `Ceetron Cloud Components`.

Ceetron also suggested an alternative solution. It required an additional server component, and was more complex. USG was therefore chosen since the additional functionality from the other solution was not required for this project. Swapping to the more complete solution was described as being a feasible option, if functionality not offered in USG is required in the future.

6.2 Challenges and limitations

In the beginning of this project we were introduced to three different physical assets: The Torsion Bar Suspension Rig, a crane located at MTP laboratories at Valgrinda and Lerkendal stadium. All three physical assets lacked the necessary hardware components for this project. The computer located on the Torsion Bar Suspension Rig had recently broken down, but a new one had been ordered. The crane at Valgrinda lacked a data acquisition board and a computer, while Lerkendal stadium lacked all the hardware components. In order to start prototyping as soon as possible we decided to start working with the Torsion Bar Suspension Rig since it required the least time to get up and running. In addition it was the asset that was most accessible and complete.

Digital Twin as a field and as a concept is still in the process of being established and developed. As a result, there are very few 'best practices' available. In discovering what tools to employ there was thus very little documentation regarding how to utilise them. This extended to FEDEM and Catman, where the complexity of the programs and lack of proper documentation of the relevant functionality have been a challenge. An example of this was during our first attempt at streaming the incoming data through FEDEM. The Dynamic Link Library (DLL) for the FEDEM solver exposed only the name of the functions with no explanation of their input parameters, types or purposes. Since there was no documentation or header files available we were unable to use it directly and had to use a wrapper from SAP, which was not immediately available. Catman had similar issues with documentation, especially regarding the physical wiring needed for the remote connection option. It was eventually solved by trial and error.

Another challenge was selecting which tools to employ and when. While there is no established best practice in cloud software for digital twins, there are plenty of tools that advertise as being helpful. There are many streaming analytics tools which claim to 'process continuous streams of event data in real time and act on the results'. During development, some of these tools were tested (SAP Analytics Cloud for instance). However it was decided that for now we would not utilise these tools as most of the analysis needed could be handled by simple statistics and plots.

6.3 Scalability

The server currently runs on a virtual machine with limited resources. This puts a limit on how many processes and script jobs that can run simultaneously. Consequently, in order to support a larger user base than the MTP department, one would need more space and processing power, especially if more complex analytic tools are needed later on. These tools will likely require the ability and space to store historical data, as currently data may only be stored client side.

Additional resources could be granted from NTNU IT if necessary. Moving the solution to a different host with more resources is also possible.

6.3.1 Adding a new digital twin

Our digital twin cloud software is tailored towards the Torsion Bar Suspension Rig and there is currently no functionality to simply add new models. Most of the code on the server can be reused (Listing 1, 2, 3 and 4) for a new model. However, there are lines of code that are model specific and these will mainly depend on:

- Number and types of sensors
- Output format for sensor data (See table 1)
- Configuration of external functions in FEDEM model
- Which values should be plotted

Should the new physical asset in question be equipped with another data acquisition system than described in section 6.1.1 this should not present a problem. As long as the data acquisition system uses UDP to send sensor values as a byte stream, the system will work with only minor adjustments on the server side.

6.4 Further work

As mentioned in section 6.2 there were two additional assets that could be used. An advisable task would be to instrument at least one of these assets. This will be beneficial for two reasons: First, if the instrumentation process is documented well, the documentation can be used as guide for setting up data acquisition systems for other physical assets later. Second, it will make it possible to test the robustness and scalability of the current digital twin cloud software.

A live video stream of the physical twin in the client is a requested feature. This feature would make it easier to verify that the digital twin behaves the same way as the physical. The system currently requires a computer at the site of the physical twin. Therefore, a solution is to connect a camera to the computer and send the live stream to the server in a similar fashion as the sensor data.

Another requested feature is event triggers to reduce the amount of uninteresting data received. In digital twins, only some of the behaviour will be of relevance, i.e during activity and under stress.

Currently the web application is tailored to visualise the Torsion Bar Suspension Rig. In the future, a more flexible visualisation setup is desired to make transition between different digital twins simpler for both the user and the developer. The visualisation should also be expanded to show deformation and stress in the form of colour change in the 3D model. The stress could in addition be visualised by a S-N curve as part of Fatigue analysis, however that would likely be separate from the current visualisation.

For digital twins equipped with accelerometers, a key feature to implement would be Fast Fourier Transform. This enables frequency analysis of the asset and can be used to detect structural changes. In addition it can be used to verify the precision of the FE model.

7 Conclusion

Cloud-based solutions for digital twin modelling have been explored and an environment has been established for developing a software solution. Requirements have been specified for developing a cloud based digital twin software solution. A prototype based on the torsion bar suspension rig has been created showcasing and satisfying most of the major points of the requirements. A user guide for how to setup each component of the prototype is available for reproducing or referencing the current system. Steps have been outlined for further iteration on this prototype to move towards a complete digital twin cloud solution.

Appendices

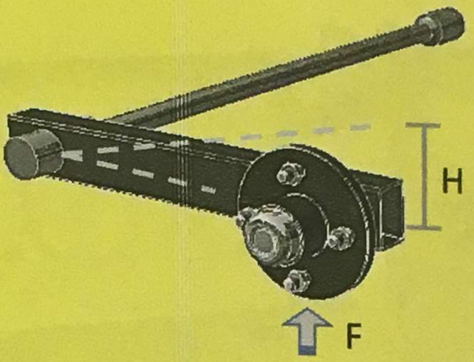
A Torsion Bar Suspension Rig Manual

Physical Test Manual

Torsion Bar Suspension Rig

Maximum Capacity

The rig is fail-proof. This means that it is OK to elevate the hydraulic jack to maximum stroke-length.

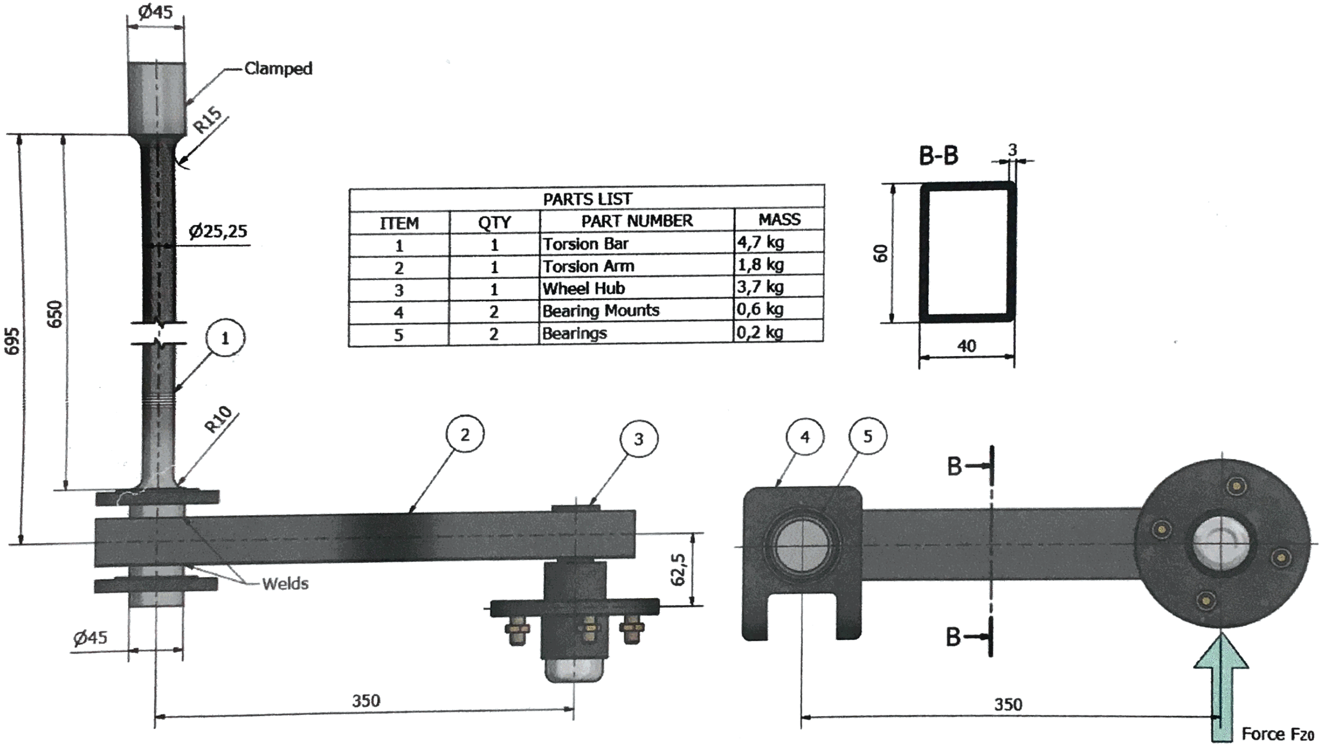


Deflection Angle	α	12,5	degrees
Elevation Height	H	81	mm
Force	F	3300	Newtons

Torsion Bar Material Data

Stainless Steel - SS2387 / S165M

E-modulus	E	210	Gpa
G-modulus	G	78	Gpa
Yield Strength	σ_{ys}	885	Mpa
Ultimate Tensile Strength	σ_{ut}	1010	Mpa
Density	δ	7700	Kg/m
Fatigue Strength Coefficient	σ'_f	1454	Mpa
Fatigue Strength Exponent	b	-0,08	
Fatigue Ductility Coefficient	ϵ'_f	1.85	
Fatigue Ductility Exponent	c	-0,72	
Cyclic Yield Strength	σ'_s	716	Mpa
Cyclic Strength Coefficient	K'	1367	Mpa
Cyclic Strain Hardening Exponent	n'	0,10	
Notch Sesitivity Factor	K_f	1,14	

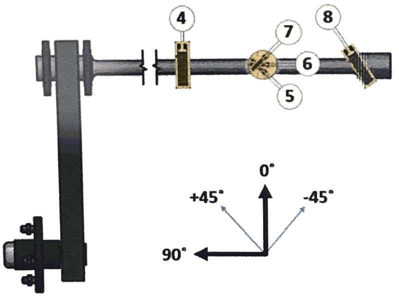


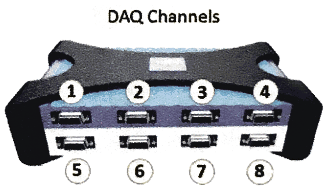
PARTS LIST			
ITEM	QTY	PART NUMBER	MASS
1	1	Torsion Bar	4,7 kg
2	1	Torsion Arm	1,8 kg
3	1	Wheel Hub	3,7 kg
4	2	Bearing Mounts	0,6 kg
5	2	Bearings	0,2 kg

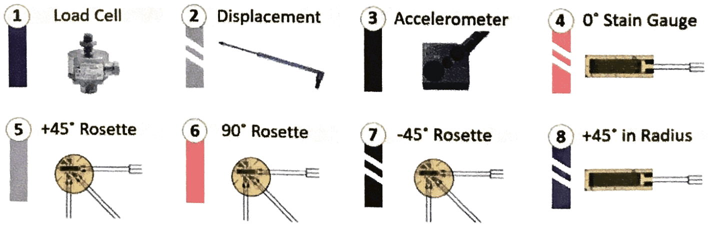
Sensors & Equipment

The rig is equipped to examine quasi-static response and the Eigen frequency.

8 sensors are installed on the rig. They are connected to a Data Acquisition box (DAQ) connected to the computer to conduct live monitoring of the tests. The applied Force is monitored by a Load Cell situated directly above the hydraulic jack. A displacement probe monitors the elevation of the wheel hub. An accelerometer is used to detect the dynamic response (Eigen Frequency) of the suspension system. Five strain gauges are situated in different locations and angles on the torsion bar. These are used to compute the torsion bar stresses.

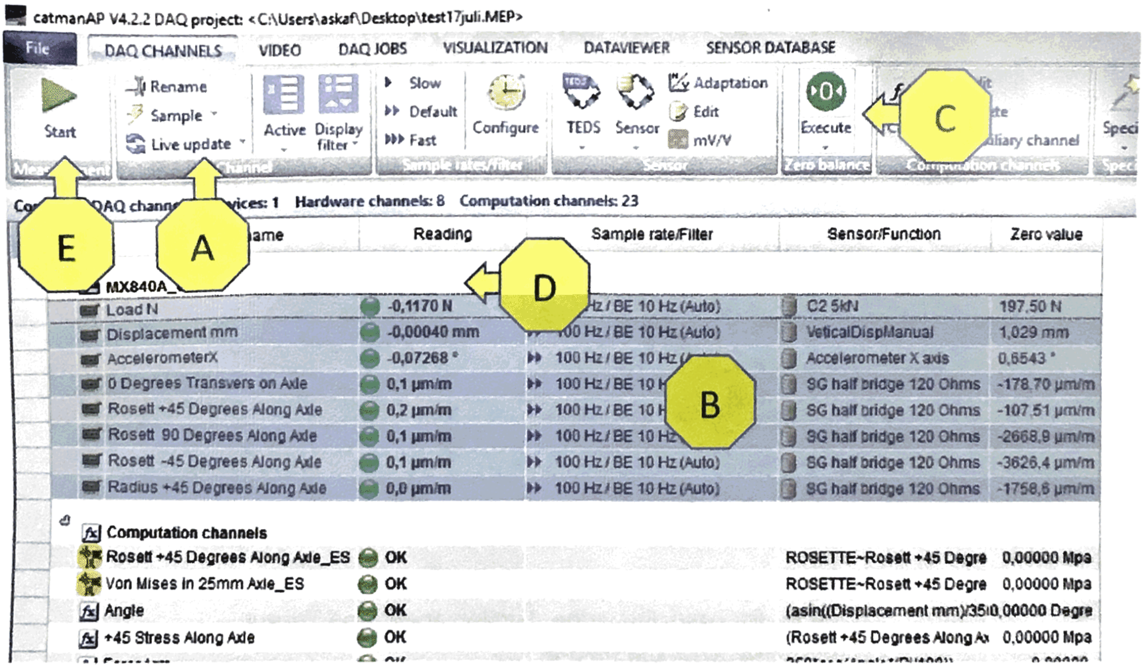






Quasi-static Test Manual

1. **Open Catman AP**
2. **Click: *Continue* (Resume my last session)**
   This opens the DAQ Channels window. Here, all the active sensors are displayed.
3. **Lower the hydraulic so the wheel hub moves freely.**
4. Before initiating a test, the sensors needs to be calibrated and zeroed. Due to the weight of the torsion arm and wheel hub (46N), this needs to be accounted for.
   Click on "A" *Live Update*. This enables live readings of the sensors. The values are visible in the Reading-column.
   Mark all the 8 sensors, "B".
   Click "C" *Execute*, to zero all the values.
   Elevate the hydraulic jack slowly, until the Load reads 46N, "D".
   Click "C" *Execute*, to zero all the values again.
5. **Click "E" *Start*** to initiate the test.

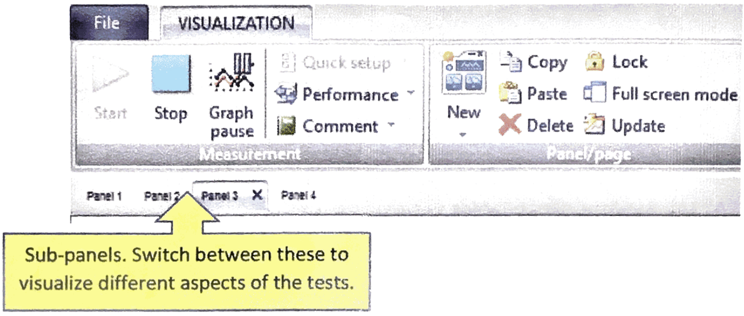


catmanAP V4.2.2 DAQ project: <C:\Users\askaf\Desktop\test17juli.MEP>

Co DAQ chann vices: 1 Hardware channels: 8 Computation channels: 23

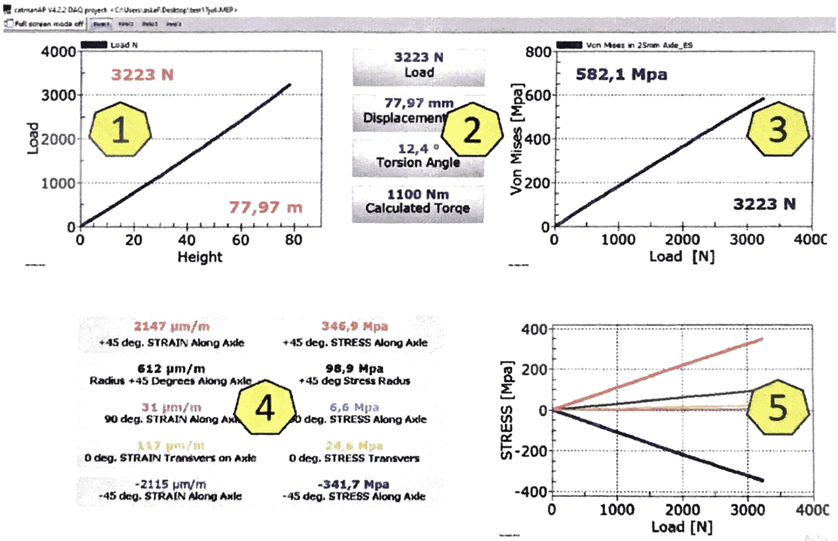
ame	Reading	Sample rate/Filter	Sensor/Function	Zero value
MX840A_				
Load N	-0,1170 N	Hz / BE 10 Hz (Auto)	C2 5kN	197,50 N
Displacement mm	-0,00040 mm	00 Hz / BE 10 Hz (Auto)	VeticalDispManual	1,029 mm
AccelerometerX	-0,07268 °	100 Hz / BE 10 Hz (	Accelerometer X axis	0,6543 °
0 Degrees Transvers on Axle	0,1 µm/m	100 Hz / BE 10 H	SG half bridge 120 Ohms	-178,70 µm/m
Rosett +45 Degrees Along Axle	0,2 µm/m	100 Hz / BE 10 H	SG half bridge 120 Ohms	-107,51 µm/m
Rosett 90 Degrees Along Axle	0,1 µm/m	100 Hz / BE 10 Hz	SG half bridge 120 Ohms	-2668,9 µm/m
Rosett -45 Degrees Along Axle	0,1 µm/m	100 Hz / BE 10 Hz (Auto)	SG half bridge 120 Ohms	-3626,4 µm/m
Radius +45 Degrees Along Axle	0,0 µm/m	100 Hz / BE 10 Hz (Auto)	SG half bridge 120 Ohms	-1758,6 µm/m
Computation channels				
Rosett +45 Degrees Along Axle_ES	OK		ROSETTE~Rosett +45 Degre	0,00000 Mpa
Von Mises in 25mm Axle_ES	OK		ROSETTE~Rosett +45 Degre	0,00000 Mpa
Angle	OK		(asin((Displacement mm)/35	0,00000 Degre
+45 Stress Along Axle	OK		(Rosett +45 Degrees Along Ax	0,00000 Mpa

6. When a test has been started, the *VISUALIZATION panel opens.* This panel gives live monitoring of the test. **Sub-panels are prepared to visualize the quasi-static testing. Switch between these to visualize different aspects of the tests.**



Sub-panels. Switch between these to visualize different aspects of the tests.

Sub-panel 1 – Stress and Overview



3223 N
Load

77,97 mm
Displacement

12,4 °
Torsion Angle

1100 Nm
Calculated Torqe

2147 µm/m
+45 deg. STRAIN Along Axle

346,9 Mpa
+45 deg. STRESS Along Axle

612 µm/m
Radius +45 Degrees Along Axle

98,9 Mpa
+45 deg Stress Radus

31 µm/m
90 deg. STRAIN Along Axle

6,6 Mpa
90 deg. STRESS Along Axle

117 µm/m
0 deg. STRAIN Transvers on Axle

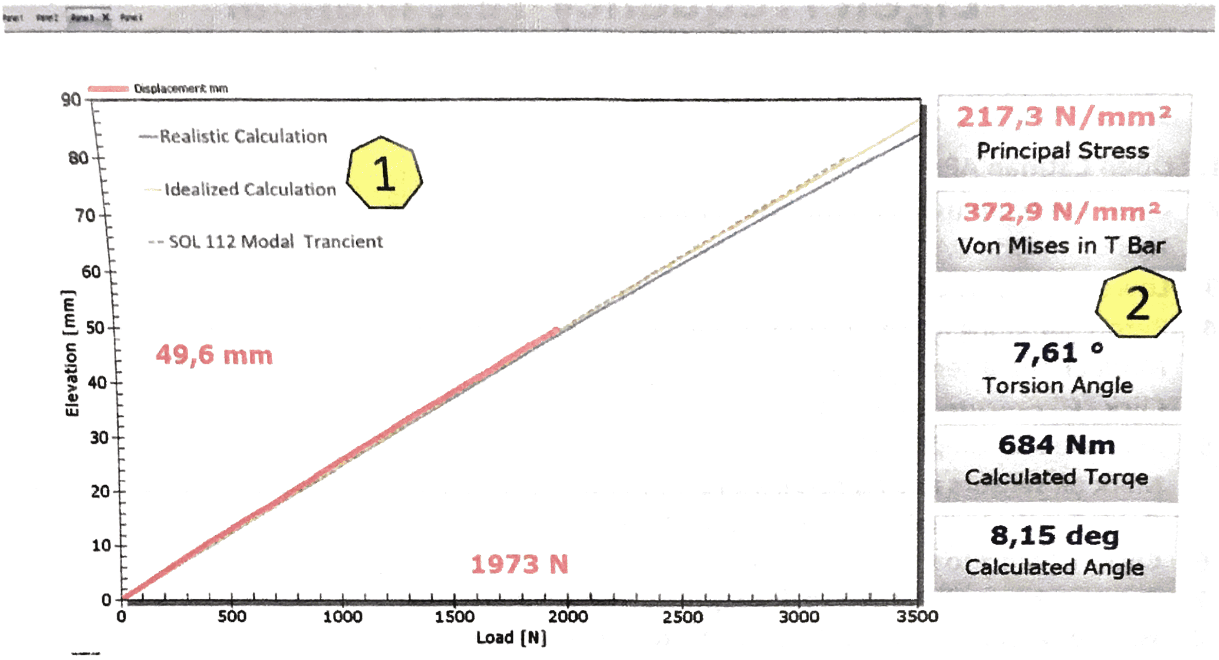
24,6 Mpa
0 deg. STRESS Transvers

-2115 µm/m
-45 deg. STRAIN Along Axle

-341,7 Mpa
-45 deg. STRESS Along Axle

(1) Elevation height/Load.
(2) Values for Load, Elevation, Torsion angle, and Torque.
(3) Von Mises Stress in torsion bar.
(4) Strain in every strain gauge and corresponding stress.
(5) Visualization of the stresses.

Sub-panel 2 Elevation vs applied load



217,3 N/mm^2
Principal Stress

372,9 N/mm^2
Von Mises in T Bar

7,61 °
Torsion Angle

684 Nm
Calculated Torqe

8,15 deg
Calculated Angle

(1) Elevation height/Load. The Background picture displays analytical and virtual solutions to estimate the height/load relationship.

(2) Values: Principal and Von Mises stresses in the torsion bar. Torsion angle, Torque and calculated torsion arm angle.

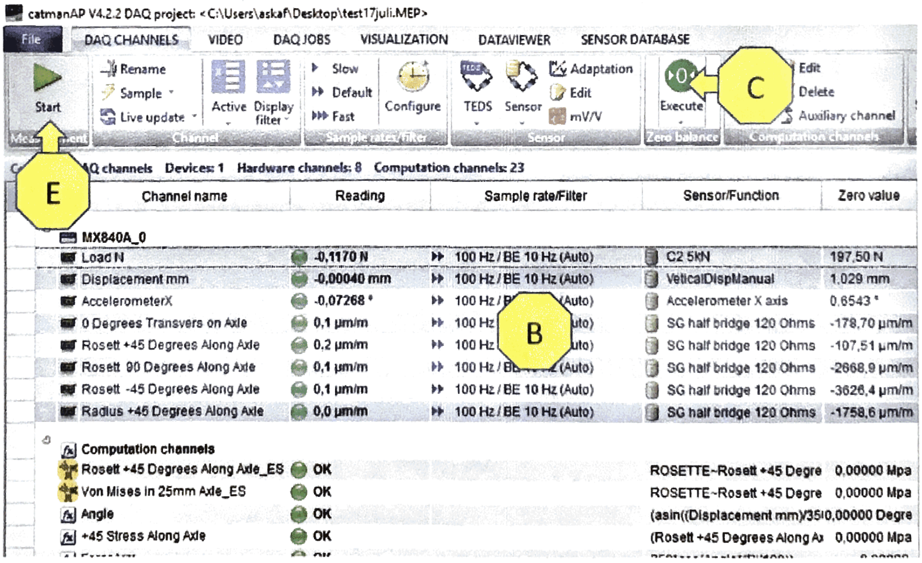
7. **Use the hydraulic jack to elevate the wheel hub.** Watch the Live monitoring.

8. **When hydraulic jack reaches the maximum position, Click *Stop* to end the test.**

9. **Test data can be exported by selecting:** *File →Save as→Save last DAQ job.* Choose desired format (e.g. Excel or Matlab)

Eigen Frequency Test Manual

1. **Open Catman AP**
2. **Click: *Continue* (Resume my last session)**
   This opens the DAQ Channels window. Here, all the active sensors are displayed.
3. **Lower the hydraulic so the wheel hub moves freely.**
4. Before initiating a test, the sensors needs to be calibrated and zeroed.
   Mark all the 8 sensors, "B".
   Click "C" *Execute*, to zero all the values.
5. Increase the sample rate:
   Mark all sensors and Right-click directly above the sample rate,"B". Click *Configure Sample Rate.* Set the sample rate to at least 300Hz.
6. **Click "E" *Start*** to initiate the test.
7. **Select sub-panel; Panel 4 to display the dynamic visualization.**
8. **Hit the wheel hub by hand repeatedly to initiate oscillation.** A sprike on the right graph will occour. This identifies the Eigen frequency.
9. **If desired: Attach the extra wheel hub weight to examine the difference.**
10. **End the test by clicking *Stop.***
11. **Reset the sample rate to 100Hz.**



catmanAP V4.2.2 DAQ project: <C:\Users\askaf\Desktop\test17juli.MEP>

AQ channels Devices: 1 Hardware channels: 8 Computation channels: 23

Channel name	Reading	Sample rate/Filter	Sensor/Function	Zero value
MX840A_0				
Load N	-0,1170 N	100 Hz / BE 10 Hz (Auto)	C2 5kN	197,50 N
Displacement mm	-0,00040 mm	100 Hz / BE 10 Hz (Auto)	VeticalDispManual	1,029 mm
AccelerometerX	-0,07268 °	100 Hz / B (Auto)	Accelerometer X axis	0,6543 °
0 Degrees Transvers on Axle	0,1 µm/m	100 Hz uto)	SG half bridge 120 Ohms	-178,70 µm/m
Rosett +45 Degrees Along Axle	0,2 µm/m	100 Hz uto)	SG half bridge 120 Ohms	-107,51 µm/m
Rosett 90 Degrees Along Axle	0,1 µm/m	100 Hz / B (Auto)	SG half bridge 120 Ohms	-2668,9 µm/m
Rosett -45 Degrees Along Axle	0,1 µm/m	100 Hz / BE 10 Hz (Auto)	SG half bridge 120 Ohms	-3626,4 µm/m
Radius +45 Degrees Along Axle	0,0 µm/m	100 Hz / BE 10 Hz (Auto)	SG half bridge 120 Ohms	-1758,6 µm/m
Computation channels				
Rosett +45 Degrees Along Axle_ES	OK		ROSETTE~Rosett +45 Degre	0,00000 Mpa
Von Mises in 25mm Axle_ES	OK		ROSETTE~Rosett +45 Degre	0,00000 Mpa
Angle	OK		(asin((Displacement mm)/35	0,00000 Degre
+45 Stress Along Axle	OK		(Rosett +45 Degrees Along A	0,00000 Mpa

B Software Packages

B.1 Node Packages

- python-struct
- dgram
- express
- http
- socket.io

B.2 Python Modules

- struct
- socket
- vpmSolverRun
- vpmSolver

Appendix C

Source Code

File - C:\Users\Odd\vue-digtwin\src\App.vue

```
<template>
  <v-app>
    <app-toolbar>
      <template #right-top-corner>
        <select-data-source-channels />
        <data-connection-monitor :isOpen="isOpen" @changed="isOpen = !isOpen" :isConnecting="isConnecting" />
      </template>
    </app-toolbar>
    <v-content>
      <keep-alive>
        <router-view />
      </keep-alive>
    </v-content>
    <popup-message ref="PopupMessage" />
  </v-app>
</template>
<script>
import AppToolbar from './components/AppToolbar'
import PopupMessage from './components/PopupMessage'
import channelParser from './mixins/channelParser'
const DataConnectionMonitor = () => import('./components/DataConnectionMonitor')
const SelectDataSourceChannels = () => import('./components/dialogs/SelectDataSourceChannels')

export default {
  components: { SelectDataSourceChannels, DataConnectionMonitor, PopupMessage, AppToolbar },
  mixins: [channelParser],
  mounted () {
    // Register method on root so all underlying components have access
    this.$root.displayPopup = message =>
      this.$refs.PopupMessage.displayMessage(message)
    this.$store.dispatch('digTwinModule/fetchModelList')
  }
}
</script>

```

File - C:\Users\Odd\vue-digtwin\src\main.js

```
import Vue from 'vue'
import './plugins/vuetify'
import App from './App.vue'
import router from './router/router'
import store from './store/store'
import VuetifyConfirm from 'vuetify-confirm'
Vue.use(VuetifyConfirm)

Vue.config.productionTip = false

new Vue({
  router,
  store,
  render: h => h(App)
}).$mount('#app')

```

```
"use strict";
Object.defineProperty(exports, "__esModule", { value: true });
var cee = require("../ceetron/CeeCloudClientComponent");
// Initialiser for Ceetron module of application
function startApp(canvasElement) {
    return new App(canvasElement);
}
exports.startApp = startApp;
// Class containing Ceetron Cloud Client Component state
var App = /** @class */ (function () {
    function App(canvas) {
        var _this = this;
        this.canvas = canvas;
        // Initialise Ceetron Cloud Client Component
        this.cloudSession = new cee.CloudSession();
        var viewer = this.cloudSession.addViewer(canvas);
        if (!viewer) {
            throw ("No WebGL support");
        }
        this.view = viewer.addView();
        this.model = new cee.usg.UnstructGridModel();
        this.view.addModel(this.model);
        this.state = this.model.addState();
        this.state.geometry = new cee.usg.Geometry();
        // Hide infoBox initially
        this.view.overlay.infoBoxVisible = false;
        // Listen for resize events
        window.addEventListener('resize', function () { return _this._handleWindowResizeEvent(); });
        // Manually run resize function once
        this._handleWindowResizeEvent();
        // Update view every browser frame
        window.requestAnimationFrame(function (time) { return _this._myAnimationFrameCallback(time); });
    }
    // Adjust view dimension (called when window is resized)
    App.prototype._handleWindowResizeEvent = function () {
        var canvasWidth = window.innerWidth;
        var canvasHeight;
        // @ts-ignore
        canvasHeight = this.canvas.parentElement.offsetHeight;
        this.cloudSession.getViewerAt(0).resizeViewer(canvasWidth, canvasHeight);
    };
    // Force the layout to resize, same reason as in plotcomponent
    App.prototype.resizeLayout = function () {
        var canvasHeight;
        // @ts-ignore
        canvasHeight = this.canvas.parentElement.offsetHeight;
        var canvasWidth;
        // @ts-ignore
        canvasWidth = this.canvas.parentElement.offsetWidth;
        this.cloudSession.getViewerAt(0).resizeViewer(canvasWidth, canvasHeight);
    };
    // Update view (called every browser frame)
    App.prototype._myAnimationFrameCallback = function (highResTimestamp) {
        var _this = this;
        this.cloudSession.handleAnimationFrameCallback(highResTimestamp);
        window.requestAnimationFrame(function (time) { return _this._myAnimationFrameCallback(time); });
    };
    // Create the torsion rod geometry
    App.prototype.addRodGeometry = function (data) {
        var geometry = this.state.geometry.addPart();
        geometry.mesh = new cee.usg.Mesh(data.nodeArr, data.elementTypeArr, data.elementNodeIndexArr);
        geometry.settings.color = new cee.Color3(.8, .8, .8);
        // Transform to global coordinate system
        var c = cee.Mat4.fromElements(1, 0, 0, -0.02407066, 0, 1, 0, -0.02722985, 0, 0, 1, 0.27199998, 0, 0, 0, 1);
        this.state.setPartTransformationAt(1, c);
    };
    App.prototype.addPartGeometry = function (data, a1, a2, a3, index) {
        var geometry = this.state.geometry.addPart();
        geometry.mesh = new cee.usg.Mesh(data.nodeArr, data.elementTypeArr, data.elementNodeIndexArr);
        geometry.settings.color = new cee.Color3(Math.random(), Math.random(), Math.random());
        var c = cee.Mat4.fromElements(a1[0], a1[1], a1[2], a1[3], a2[0], a2[1], a2[2], a2[3], a3[0], a3[1], a3[2], a3[3], 0, 0, 0, 1);
        this.state.setPartTransformationAt(index, c);
        //this.showStatistics(this.state.geometry);
    };
    App.prototype.showStatistics = function (geometry) {
        // Generate statistics on geometry
        var nodeCount = 0;
        var elementCount = 0;
        for (var _i = 0, _a = geometry.getPartArray(); _i < _a.length; _i++) {
            var part = _a[_i];
            nodeCount += part.mesh.nodeCount;
            elementCount += part.mesh.elementCount;
        }
        // Log generated statistics
        console.log("Initial state loaded, nodeCount=" + nodeCount + ", elementCount=" + elementCount);
        // Draw generated statistics in bottom right corner
        this.view.overlay.infoBoxVisible = true;
        this.view.overlay.setInfoBoxContent("Elements: " + elementCount + " elements \n Nodes: " + nodeCount + " nodes");
    };
    App.prototype.updateDisplacement = function (transformationMatrix, baseID) {
        // Create Ceetron matrix from transformation data
        var m = cee.Mat4.fromArray(transformationMatrix);
        var localToGlobalTransformation = cee.Mat4.fromElements(1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1);
        var transformation = cee.Mat4.multiply(m, localToGlobalTransformation);
        // Apply transformation to armGeometry
        this.state.setPartTransformationAt(baseID, transformation);
    };
    // Change drawing style for geometries
    App.prototype.setDrawStyle = function (ds) {
        var geometry = this.model.getStateAt(0).geometry;
        for (var _i = 0, _a = geometry.getPartArray(); _i < _a.length; _i++) {
            var part = _a[_i];
            if (ds === "surface")
                part.settings.drawStyle = cee.usg.DrawStyle.SURFACE;
            else if (ds === "surface_mesh")
                part.settings.drawStyle = cee.usg.DrawStyle.SURFACE_MESH;
            else if (ds === "outline_mesh")
                part.settings.drawStyle = cee.usg.DrawStyle.SURFACE_OUTLINE_MESH;
            else if (ds === "lines")
                part.settings.drawStyle = cee.usg.DrawStyle.LINES;
            else if (ds === "points")
                part.settings.drawStyle = cee.usg.DrawStyle.POINTS;
            else if (ds === "outline")
                part.settings.drawStyle = cee.usg.DrawStyle.OUTLINE;
        }
    };
    return App;
}());
exports.App = App;

```

```
import * as cee from "../ceetron/CeeCloudClientComponent";

// Initialiser for Ceetron module of application
export function startApp(canvasElement: HTMLCanvasElement): App {
    return new App(canvasElement);
}

// Class containing Ceetron Cloud Client Component state
export class App {

    // Ceetron Cloud Client Component state
    private cloudSession: cee.CloudSession;
    private view: cee.View;
    private model: cee.usg.UnstructGridModel;
    private state: cee.usg.State;

    // Canvas containing visualisation
    private canvas: HTMLCanvasElement;

    constructor(canvas: HTMLCanvasElement) {
        this.canvas = canvas;

        // Initialise Ceetron Cloud Client Component
        this.cloudSession = new cee.CloudSession();
        let viewer = this.cloudSession.addViewer(canvas);
        if (!viewer) {
            throw("No WebGL support");
        }
        this.view = viewer.addView();
        this.model = new cee.usg.UnstructGridModel();
        this.view.addModel(this.model);
        this.state = this.model.addState();
        this.state.geometry = new cee.usg.Geometry();

        // Hide infoBox initially
        this.view.overlay.infoBoxVisible = false;

        // Listen for resize events
        window.addEventListener('resize', () => this._handleWindowResizeEvent());

        // Manually run resize function once
        this._handleWindowResizeEvent();

        // Update view every browser frame
        window.requestAnimationFrame((time: number) => this._myAnimationFrameCallback(time));
    }

    // Adjust view dimension (called when window is resized)
    private _handleWindowResizeEvent() {
        let canvasWidth = window.innerWidth;
        let canvasHeight: number;
        // @ts-ignore
        canvasHeight = this.canvas.parentElement.offsetHeight;
        this.cloudSession.getViewerAt(0).resizeViewer(canvasWidth, canvasHeight);
    }

    // Force the layout to resize, same reason as in plotcomponent
    public resizeLayout() {
        let canvasHeight: number;
        // @ts-ignore
        canvasHeight = this.canvas.parentElement.offsetHeight;
        let canvasWidth: number;
        // @ts-ignore
        canvasWidth = this.canvas.parentElement.offsetWidth;
        this.cloudSession.getViewerAt(0).resizeViewer(canvasWidth, canvasHeight);
    }
    // Update view (called every browser frame)
    private _myAnimationFrameCallback(highResTimestamp:number) {
        this.cloudSession.handleAnimationFrameCallback(highResTimestamp);
        window.requestAnimationFrame((time: number) => this._myAnimationFrameCallback(time));
    }





    // Create the torsion rod geometry
    addRodGeometry(data: any) {
        let geometry = this.state.geometry.addPart();
        geometry.mesh = new cee.usg.Mesh(data.nodeArr, data.elementTypeArr, data.elementNodeIndexArr);
        geometry.settings.color = new cee.Color3(.8, .8, .8);

        // Transform to global coordinate system
        const c = cee.Mat4.fromElements(
            1, 0, 0, -0.02407066,
            0, 1, 0, -0.02722985,
            0, 0, 1, 0.27199998,
            0, 0, 0, 1
        );
        this.state.setPartTransformationAt(1, c);
    }

    addPartGeometry(data: any,a1: any,a2: any,a3: any,index: any){
        let geometry = this.state.geometry.addPart();
        geometry.mesh = new cee.usg.Mesh(data.nodeArr, data.elementTypeArr, data.elementNodeIndexArr);
        geometry.settings.color = new cee.Color3(Math.random(),Math.random(),Math.random());


        const c = cee.Mat4.fromElements(
            a1[0], a1[1], a1[2], a1[3],
            a2[0], a2[1], a2[2], a2[3],
            a3[0], a3[1], a3[2], a3[3],
            0, 0, 0, 1
        );

        this.state.setPartTransformationAt(index, c);
        //this.showStatistics(this.state.geometry);
    }

    private showStatistics(geometry: any) {
        // Generate statistics on geometry

        let nodeCount = 0;
        let elementCount = 0;
        for (let part of geometry.getPartArray()) {
            nodeCount += part.mesh.nodeCount;
            elementCount += part.mesh.elementCount;
        }

        // Log generated statistics
        console.log("Initial state loaded, nodeCount=" + nodeCount + ", elementCount=" + elementCount);

        // Draw generated statistics in bottom right corner
        this.view.overlay.infoBoxVisible = true;
        this.view.overlay.setInfoBoxContent(`Elements: ${elementCount} elements \nNodes: ${nodeCount} nodes`);
    }
```

```typescript

    updateDisplacement(transformationMatrix: number[],baseID: any) {
        // Create Ceetron matrix from transformation data
        const m = cee.Mat4.fromArray(transformationMatrix);
        const localToGlobalTransformation = cee.Mat4.fromElements(
            1, 0, 0, 0,
            0, 1, 0, 0,
            0, 0, 1, 0,
            0, 0, 0, 1
        );
        const transformation = cee.Mat4.multiply(m,localToGlobalTransformation);

        // Apply transformation to armGeometry
        this.state.setPartTransformationAt(baseID, transformation);
    }

    // Change drawing style for geometries
    setDrawStyle(ds: string) {
        const geometry = this.model.getStateAt(0).geometry;
        for (let part of geometry.getPartArray()) {
            if      (ds === "surface")              part.settings.drawStyle = cee.usg.DrawStyle.SURFACE;
            else if (ds === "surface_mesh")         part.settings.drawStyle = cee.usg.DrawStyle.SURFACE_MESH;
            else if (ds === "outline_mesh")         part.settings.drawStyle = cee.usg.DrawStyle.SURFACE_OUTLINE_MESH;
            else if (ds === "lines")                part.settings.drawStyle = cee.usg.DrawStyle.LINES;
            else if (ds === "points")               part.settings.drawStyle = cee.usg.DrawStyle.POINTS;
            else if (ds === "outline")              part.settings.drawStyle = cee.usg.DrawStyle.OUTLINE;
        }
    }
}


```

File - C:\Users\Odd\vue-digtwin\src\js\EventBus.js

```
import Vue from 'vue'
export const EventBus = new Vue()

export const EVENTS = {
  subscribe: 'subscribeToChannels',
  newData: 'asdf'
}

```

```
import { rootAPI } from '../api/APIHelper'

const fmuEndpoint = rootAPI + '/fmus/'
export class DigitalTwin {
  /*
      This is class retrieves all the necessary information needed for the USG.js module to create a 3D visualisation
      of the model.

      Two types of files are needed:
          - JSON Master file
              - List of parts
              - Base ID for each part
              - Local to global coordinate system transformation matrix for each part
          - JSON Part file (For each part in model)
              - Part geometry description
   */

  constructor (name, myApp) {
    // Initiate variables
    this.name = name
    this.fileName = this.name.concat('.json')
    this.directory = fmuEndpoint.concat(this.name) // /js/Name/
    this.parts = []
    this.arrays = []
    this.baseId = []
    this.myApp = myApp
    // Create Model
    this.createModel()
  }

  // Return part index
  getPartIndex (baseID) {
    /*
        To properly select a part when updating its position we need the index it has been given. The index of a
        part corresponds to the parts position in the list of parts.
        e.g:
        this.parts = [arm, base, pulley, upperArm, ....] thus the index of "pulley" is 2.
     */

    let index = -1

    for (let i = 0; i < this.baseId.length; i++) {
      if (this.baseId[i] === baseID) {
        index = i
      }
    }
    return index
  }

  // Return array with Base IDs
  getIDS () {
    return this.baseId
  }

  // Create 3D model
  async createModel () {
    // Retrieve information from JSON master file
    await this.findParts()

    // Generate 3D visualisation for each part
    for (let i = 0; i < this.parts.length; i++) {
      await this.loadParts(this.parts[i], this.arrays[i], i)
    }
  }

  async loadParts (Name, Arr, index) {
    /*
        This function loads part geometry and calls the function "addPartGeometry" which uses the Ceetron USG module
        to create the 3D visualisation of this part.
     */

    const partEndPoint = this.directory.concat('/models/').concat(Name)
    try {
      const response = await fetch(partEndPoint, { cors: 'no-cors' })
      const data = await response.json()
      this.myApp.addPartGeometry(data, Arr[0], Arr[1], Arr[2], index)
    } catch (error) {
      console.log(error)
    }
  }

  async findParts () {
    /*
         Retrieve information from JSON master file:
            - List of parts
            - Base ID for each part
            - Local to global coordinate system transformation matrix for each part
     */
    const masterFileEndpoint = this.directory.concat('/models/').concat(this.fileName)
    try {
      const response = await fetch(masterFileEndpoint, { cors: 'no-cors' })
      const data = await response.json()
      let files = data.ListOfFile
      let coordinates = data.Coordinates
      let IDs = data.baseID

      for (let i = 0; i < files.length; i++) {
        const index = i * 3
        files[i] = files[i].replace('.ftl', '.json')
        this.parts.push(files[i])
        this.baseId.push(IDs[i])
        this.arrays.push([coordinates[index], coordinates[index + 1], coordinates[index + 2]])
      }
    } catch (error) {
      console.log(error)
    }
  }
}

```

File - C:\Users\Odd\vue-digtwin\src\api\APIHelper.js

```
import { getJSONResponse } from '../utils/util'

export const rootAPI = 'http://129.241.90.187:1337'

// sourceURL is on the form /datasourcees/{id}, /processors/{id} or /topics/{id}
export async function getOutputNames (sourceUrl) {
  const sourceJSON = await getJSONResponse(rootAPI + sourceUrl)
  return sourceJSON.output_names || []
}

export async function fetchFMUs () {
  return getJSONResponse(rootAPI + /fmus/)
}

export async function fetchBlueprints () {
  return getJSONResponse(rootAPI + '/blueprints/')
}

export async function fetchTopics () {
  return getJSONResponse(rootAPI + '/topics/')
}

export async function subscribeToSource (sourceId) {
  await fetch(rootAPI + '/topics/' + sourceId + '/subscribe', { credentials: 'include' })
}

export async function unSubscribeSource (sourceId) {
  await fetch(rootAPI + '/topics/' + sourceId + '/unsubscribe', { credentials: 'include' })
}

export async function createDataSource (formData) {
  const createLink = rootAPI + '/datasources/create'
  return fetch(createLink, {
    method: 'POST',
    body: formData
  })
}

export async function createProcessor (formData) {
  const createLink = rootAPI + '/processors/create'
  return fetch(createLink, {
    method: 'POST',
    body: formData
  })
}

export async function startProcessorRequest (formData) {
  return fetch(rootAPI + '/processors/start', {
    method: 'POST',
    body: formData
  })
}

// Sends request to change inputs or outputs of running process,
// endPoint is either 'inputs' or 'outputs'
export async function editProcessorIOs (processorID, endPoint, formData) {
  const link = rootAPI + '/processors/' + processorID + '/' + endPoint
  return fetch(link, {
    method: 'post',
    body: formData
  })
}

export async function fetchDataSources () {
  const dataSourcesResponse = await getJSONResponse(rootAPI + '/datasources/')
  return dataSourcesResponse || []
}

export async function startDataSource (datasourceID) {
  const startLink = rootAPI + '/datasources/' + datasourceID + '/start'
  return fetch(startLink)
}

export async function fetchBlueprint (blueprintID) {
  return getJSONResponse(rootAPI + '/blueprints/' + blueprintID)
}

```

File - C:\Users\Odd\vue-digtwin\src\api\extractor.js

```javascript
// Returns an processor object with two lists: Inputs and outputs
import { deepCopy } from '../utils/util'

export function extractProcessorIOs (processorJSON) {
  if (processorJSON.inputs === undefined && processorJSON.outputs === undefined) {
    return false
  }
  const selectedMeasurementRefs = processorJSON.measurement_refs
  const inputs = processorJSON.inputs.map((input, index) => ({
    input_ref: index,
    name: input.name,
    measurement_ref: selectedMeasurementRefs[index] || -1,
    measurement_proportion: 1
  }))
  const matrixOutputRefs = processorJSON.matrix_outputs
  const outputRefs = processorJSON.output_refs
  let scalarOutputs = processorJSON.outputs.map((output, index) => ({
    id: index,
    name: output.name,
    selected: outputRefs.includes(index)
  }))
  if (matrixOutputRefs !== undefined) {
    const allOutputs = deepCopy(scalarOutputs)
    const matrixOutputs = Object.entries(matrixOutputRefs).map((matrixOutput) => {
      const matrixOutputIndices = matrixOutput[1]
      return {
        name: matrixOutput[0] + '_matrix',
        selected: outputRefs.includes(matrixOutputIndices[0]),
        matrixOutputNames: allOutputs.filter((output, index) => {
          if (matrixOutputIndices.includes(index)) {
            scalarOutputs.splice(index)
            return true
          }
        })
      }
    })
    return { inputs: inputs, outputs: scalarOutputs.concat(matrixOutputs) }
  }
}

export function getInitDocs (docs) {
  let newInitDocs = {}
  let docArray = docs.init_docs.split(':')
  if (docArray[0].split(' ')[0] !== 'param') {
    docArray = docArray.slice(1)
  }
  let currentParamName = ''
  docArray.forEach(line => {
    let tempArray = line.split(' ')
    if (tempArray[0] === 'param') {
      currentParamName = tempArray[1]
    } else {
      newInitDocs[currentParamName] = line
    }
  })
  return newInitDocs
}

```

File - C:\Users\Odd\vue-digtwin\src\api\formDataCreator.js

```javascript
export function createProcessorInputFormData (inputs) {
  let formData = new FormData()
  const insertedInputs = inputs.filter(input => input.measurement_ref !== -1)
  if (insertedInputs.length === 0) {
    // No inputs have any reference to a datachannel
    return false
  }
  insertedInputs.forEach(input => {
    formData.append('input_ref', input.input_ref)
    formData.append('measurement_ref', input.measurement_ref)
    formData.append('measurement_proportion', input.measurement_proportion)
  })
  return formData
}

export function createProcessorOutputFormData (outputs, formData) {
  if (formData === undefined) {
    formData = new FormData()
  }
  for (let i = 0; i < outputs.length; i++) {
    const matrixOutputs = outputs[i].matrixOutputNames
    if (matrixOutputs === undefined) {
      formData.append('output_ref', outputs[i].id)
    } else {
      matrixOutputs.forEach(output => {
        formData.append('output_ref', output.id)
      })
    }
  }
  return formData
}

export function createProcessorFormData (processorId, selectedBlueprint, initParams, source) {
  let formData = new FormData()
  formData.append('id', processorId)
  formData.append('blueprint', selectedBlueprint)
  formData.append('init_params', JSON.stringify(initParams))
  formData.append('topic', source)
  formData.append('min_output_interval', '0.01')
  return formData
}

```

```
import Vue from 'vue'
import Vuex from 'vuex'
import { channelModule } from './modules/channels'
import { digTwinModule } from './modules/digTwins'
Vue.use(Vuex)

export default new Vuex.Store({
  modules: {
    channelModule: channelModule,
    digTwinModule: digTwinModule
  }
})

```

```javascript
const savedLayouts = 'savedLayouts'
const lastSelectionKey = 'lastSelection'

let savedLayoutsDict = () => {
  let rawSavedLayouts = localStorage.getItem(savedLayouts)
  if (rawSavedLayouts !== undefined && rawSavedLayouts !== null) {
    try {
      return JSON.parse(rawSavedLayouts)
    } catch (error) {
      console.log(error)
      return {}
    }
  } else {
    return {}
  }
}

function getLayoutIds () {
  return Object.keys(savedLayoutsDict())
}

function saveLayout ({ id, newLayout }) {
  // Make an empty layoutDict if the original is undefined
  let layoutDict = savedLayoutsDict() || {}
  layoutDict[id] = newLayout
  localStorage.setItem(savedLayouts, JSON.stringify(layoutDict))
}

function deleteLayout (layoutId) {
  let layoutDict = savedLayoutsDict()
  delete layoutDict[layoutId]
  localStorage.setItem(savedLayouts, JSON.stringify(layoutDict))
  return Object.keys(layoutDict)[0]
}
function getLayoutById (id) {
  return savedLayoutsDict() ? savedLayoutsDict()[id] : undefined
}

function saveSelection (layoutId) {
  localStorage.setItem(lastSelectionKey, JSON.stringify(layoutId))
}

function getLastSelection () {
  const lastSelectedLayout = localStorage.getItem(lastSelectionKey)
  if (!lastSelectedLayout) {
    return false
  }
  try {
    return JSON.parse(lastSelectedLayout)
  }
  catch (e) {
    return ''
  }
}

function layoutExists (id) {
  return this.getLayoutById(id) !== undefined
}

export default {
  saveSelection, saveLayout, getLastSelection, getLayoutById, getLayoutIds, deleteLayout, layoutExists
}

```

```javascript
export const channelModule = {
  namespaced: true,
  state: {
    // Dictionary of sourceIDs, example:
    /* sourceDict: {
      0000: {
        byteFormat: '<dddddddddd'
        name: 'testrig',
        channels: [
          {
            id: 1,
            name: 'Load [N]'
          }
        ]
      }
    } */
    sourceDict: {}
  },
  mutations: {
    setSourceDict (state, newDict) {
      state.sourceDict = newDict
    }
  },
  actions: {
    generateDataSources ({ commit, state }, dataSources) {
      let newSourceDict = {}
      dataSources.forEach((source) => {
        const channels = source.subscribedChannels
        let newChannels = []
        for (let i = 0; i < channels.length; i++) {
          const channelName = channels[i].channelName
          const id = channels[i].id
          newChannels.push({
            id: id,
            name: channelName
          })
        }
        newSourceDict[source.id] = {
          name: source.name,
          byteFormat: source.byteFormat,
          channels: newChannels
        }
      })
      commit('setSourceDict', newSourceDict)
    }
  }
}

```

File - C:\Users\Odd\vue-digtwin\src\store\modules\digTwins.js

```
import { fetchFMUs } from '../../api/APIHelper'

export const digTwinModule = {
  namespaced: true,
  state: {
    models: ['CraneShort', 'TestRig', 'krane']
  },
  mutations: {
    setModels (state, newModels) {
      state.models = newModels
    }
  },
  actions: {
    async fetchModelList ({ commit, state }) {
      let models = await fetchFMUs()
      commit('setModels', models)
    }
  }
}

```

```
export function deepCopy (object) {
  return JSON.parse(JSON.stringify(object))
}

export async function getJSONResponse (link) {
  let jsonResponse
  try {
    const response = await fetch(link)
    jsonResponse = await response.json()
  } catch (error) {
    console.log(error)
    return false
  }
  return jsonResponse
}

```

File - C:\Users\Odd\vue-digtwin\src\utils\vueutils.js

```
// Lazy-loads view components, but with better UX. A loading view
// will be used if the component takes a while to load, falling
// back to a timeout view in case the page fails to load. You can
// use this component to lazy-load a route with:
//
// component: () => lazyLoadView(import('@views/my-view'))
//
// NOTE: Components loaded with this strategy DO NOT have access
// to in-component guards, such as beforeRouteEnter,
// beforeRouteUpdate, and beforeRouteLeave. You must either use
// route-level guards instead or lazy-load the component directly:
//
// component: () => import('@views/my-view')
//
export function lazyLoadView (AsyncView) {
  const AsyncHandler = () => ({
    component: AsyncView,
    // A component to use while the component is loading.
    loading: require('../views/_loading').default,
    // Delay before showing the loading component.
    // Default: 200 (milliseconds).
    delay: 200,
    // A fallback component in case the timeout is exceeded
    // when loading the component.
    error: require('../views/_timeout').default,
    // Time before giving up trying to load the component.
    // Default: Infinity (milliseconds).
    timeout: 5000
  })

  return Promise.resolve({
    functional: true,
    render (h, { data, children }) {
      // Transparently pass any props or children
      // to the view component.
      return h(AsyncHandler, data, children)
    }
  })
}

export function lazyLoadComponent (AsyncComponent) {
  return () => ({
    // The component to load (should be a Promise)
    component: AsyncComponent,
    // A component to use while the async component is loading
    loading: require('../views/_loading').default,
    // A component to use if the load fails
    error: require('../views/_timeout').default,
    // Delay before showing the loading component. Default: 200ms.
    delay: 200,
    // The error component will be displayed if a timeout is
    // provided and exceeded. Default: Infinity.
    timeout: 3000
  })
}
```

```javascript
// Create download dialog for currently plotted data
exports.save = (channel) => {
  const layout = channel.channelLayout
  const channelData = channel.channelData
  var saveData = layout.xaxis.title + ', ' + layout.yaxis.title + '\r\n'
  for (var i = 0; i < channelData.x.length; i++) {
    saveData += channelData.x[i].valueOf() + ', ' + channelData.y[i] + '\r\n'
  }
  this.download(saveData, 'twin_' + new Date().toISOString() + '.csv', 'text/csv')
}

// Downloading data to a file
exports.download = (data, filename, type) => {
  var file = new Blob([data], { type: type })
  if (window.navigator.msSaveOrOpenBlob) // IE10+
  { window.navigator.msSaveOrOpenBlob(file, filename) } else { // Others
    var a = document.createElement('a')

    var url = URL.createObjectURL(file)
    a.href = url
    a.download = filename
    document.body.appendChild(a)
    a.click()
    setTimeout(function () {
      document.body.removeChild(a)
      window.URL.revokeObjectURL(url)
    }, 0)
  }
}

```

File - C:\Users\Odd\vue-digtwin\src\views\Home.vue

```
<template>
  <v-layout column fill-height>
    <DynamicGrid></DynamicGrid>
  </v-layout>
</template>

<script>
// @ is an alias to /src
import DynamicGrid from '../components/StaticGrid'

export default {
  name: 'home',
  components: {
    DynamicGrid
  }
}
</script>

```

File - C:\Users\Odd\vue-digtwin\src\views\About.vue

```
<template>
  <div class="about">
    <h1>This is an about page</h1>
  </div>
</template>

```

File - C:\Users\Odd\vue-digtwin\src\views_loading.vue

```
<script>
import VProgressCircular from 'vuetify/lib/components/VProgressCircular/VProgressCircular'

export default {
  functional: true,
  components: { VProgressCircular },
  render () {
    return (
      <VLayout align-center justify-center>
        <VProgressCircular indeterminate color="primary" />
      </VLayout>
    )
  }
}
</script>

<style scoped></style>

```

File - C:\Users\Odd\vue-digtwin\src\views_timeout.vue

```
<template functional>
  <v-layout align-center justify-center>
    <h2>Timed Out</h2>
  </v-layout>
</template>

```

```
<template>
  <v-layout column fill-height>
    <router-view ref="DashboardGrid" />
  </v-layout>
</template>

<script>
// @ is an alias to /src
import DashboardGrid from '../components/DashboardGrid'
import LayoutSaver from '../store/custom/dashboardLayoutSaver'

export default {
  name: 'Dashboard',
  created () {
    this.$router.beforeEach(this.beforeEachCallback)
  },
  methods: {
    beforeEachCallback (to, from, next) {
      const isLeavingDashboard = !to.path.startsWith('/dashboard')
      if (isLeavingDashboard) {
        this.beforeExitingDashboard(from, next)
        // Removing beaforeEach hook. NB!: this works because the hook was the last one created
        this.$router.beforeHooks.shift()
      }
      next()
    },
    // Save selected layout id for when user returns to this route or ask if user wants to
    // exit if there are unsaved changes
    async beforeExitingDashboard (from, next) {
      if (from.name === 'NewDashboard') {
        let saveId = this.$refs.DashboardGrid.currentLayoutId
        // Check if layout has been saved
        if (!LayoutSaver.layoutExists(saveId)) {
          const shouldSave = await this.$confirm('Save the current layout before exiting?', {
            title: 'Warning'
          })
          if (shouldSave) {
            this.$refs.DashboardGrid.saveLayout()
          }
        }
      } else {
        // save the id of the current layout so it will load when returning to /dashboard/
        LayoutSaver.saveSelection(from.params.layoutId)
      }
    },
  }
}
</script>

```

File - C:\Users\Odd\vue-digtwin\src\views\Processor.vue

```
<template>
  <v-layout column align-center>
    <router-view />
  </v-layout>
</template>

```

File - C:\Users\Odd\vue-digtwin\src\views\Processors.vue

```
<template>
  <v-card>
    <v-card-title>
      <span class="headline">Processors</span>
    </v-card-title>
    <v-switch v-model="showStarted" :label="`Show only started processors`">
    </v-switch>
    <v-card-text>
      <SelectList
        v-model="selectedSimulation"
        :multi="false"
        :items="processorsToShow"
      />
    </v-card-text>

    <v-card-actions>
      <v-btn @click="navigateToEdit" :disabled="noProcessorselected"
        >Edit</v-btn
      >
      <v-spacer />
      <v-btn @click="navigateToCreate">Create</v-btn>
    </v-card-actions>
  </v-card>
</template>

<script>
import SelectList from '../components/lists/SelectList'
import { getJSONResponse } from '../utils/util'
import VCard from 'vuetify/lib/components/VCard/VCard'
export default {
  name: 'Simulations',
  components: {
    SelectList,
    VCard,
  },
  data: () => ({
    activeProcessors: [],
    startedProcessors: [],
    allProcessors: [],
    selectedSimulation: {},
    showStarted: true,
    showInitialized: false,
  }),
  created () {
    this.fetchProcessors()
  },
  computed: {
    noProcessorselected () {
      return this.selectedSimulation === undefined || Object.keys(this.selectedSimulation).length === 0
    },
    processorsToShow () {
      if(this.showStarted) {
        return this.startedProcessors
      } else  {
        return this.activeProcessors
      }
    }
  },
  methods: {
    async fetchProcessors () {
      const simulationJSON = await getJSONResponse(
        'http://129.241.90.187:1337/processors/'
      )
      this.activeProcessors = Object.entries(simulationJSON)
        .filter(processor => processor[1].initialized) // Filter out only initialized processors
        .map(processor => processor[0])
      this.startedProcessors = Object.entries(simulationJSON) // Filter out only started processors
        .filter(processor => processor[1].started)
        .map(processor => processor[0])
    },
    navigateToCreate () {
      this.$router.push({ name: 'ProcessorCreate' })
    },
    navigateToEdit () {
      this.$router.push({
        name: 'ProcessorEdit',
        params: {
          processorID: this.selectedSimulation
        }
      })
    }
  }
}
</script>

<style scoped></style>

```

File - C:\Users\Odd\vue-digtwin\src\views\ProcessorEdit.vue

```
<template>
  <SimulationEditor :processorID="$route.params.processorID" />
</template>

<script>
  import ProcessorEditor from '../components/ProcessorEditor'
  export default {
    name: 'SimulationEdit',
    components: { SimulationEditor: ProcessorEditor }

  }
</script>

<style scoped>

</style>

```

```
<template>
  <v-layout column>
    <v-toolbar flat color="white" style="margin-top: 4px">
      <v-toolbar-title>Sensors</v-toolbar-title>
      <v-spacer />
      <create-selector
        v-model="selectedSource.status"
        :dataSources="dataSources"
        @create-new="createNewDataSource"
      />
    </v-toolbar>
    <v-layout row>
      <v-text-field
        class="port-and-address"
        label="Address"
        v-model="selectedSource.address"
      />
      <v-text-field
        class="port-and-address"
        :rules="portRules"
        label="Port"
        v-model="selectedSource.port"
      />
    </v-layout>
    <SensorTable
      v-model="sensors"
      :isLoadingSensors="isLoadingSensors"
      :isEditing="isCreatingNewSource"
      @save-click="saveDataSource"
      @save-then-home="saveAndGoToHome"
    />
  </v-layout>
</template>

<script>
import SensorTable from '../components/SensorTable'
import DropDownSelector from '../components/selectors/DropDownSelector'
import CreateSelector from '../components/selectors/CreateSelector'
import { getJSONResponse } from '@utils/util'
import VTextField from 'vuetify/lib/components/VTextField/VTextField'
import {createDataSource, fetchDataSources, startDataSource} from '../api/APIHelper'

const apiEndPoint = 'http://129.241.90.187:1337/'
const testSensors = [
  ['ID', 'H'],
  ['Number of channels', 'H'],
  ['Sequence counter', 'I'],
  ['Time 1 - default sample rate', 'd'],
  ['Time 1 - slow sample rate', 'd'],
  ['Time 1 - fast sample rate', 'd'],
  ['Load [N]', 'd'],
  ['Displacement [mm]', 'd'],
  ['AccelerometerX', 'd'],
  ['0 Degrees Transvers on Axle', 'd'],
  ['Rosett +45 Degrees Along Axle', 'd'],
  ['Rosett 90 Degrees Along Axle', 'd'],
  ['Rosett -45 Degrees Along Axle', 'd'],
  ['Radius +45 Degrees Along Axle', 'd'],
  ['MX840A 0 hardware time default sample rate', 'd']
]
const tempSource = {
  id: 'testrig',
  address: '129.241.90.108',
  port: '7331'
}
const tempSensor = {
  name: 'New Sensor',
  type: 'D'
}
let hasOnlyNumbers = string => /^\d+$/.test(string)

export default {
  name: 'DataSourceEditor',
  components: {
    CreateSelector,
    DropDownSelector,
    SensorTable,
    VTextField
  },
  data() {
    return {
      sensors: [],
      dataSources: [],
      selectedSource: { status: { id: '', running: false } },
      isLoadingSensors: false,
      isCreatingNewSource: false,
      portRules: [
        v => !!v || 'Item is required',
        v => hasOnlyNumbers(v) || 'Only numbers allowed'
      ]
    }
  },
  async created() {
    this.dataSources = await this.fetchDataSources()
    if (this.dataSources.length > 0) {
      this.selectedSource.status = this.dataSources[0]
    }
  },
  computed: {
    selectedSourceID() {
      return this.selectedSource.status.id
    }
  },
  watch: {
    selectedSourceID(newSourceID) {
      if (newSourceID === undefined) return
      if (!this.isCreatingNewSource) {
        this.loadDataSource(newSourceID)
        return
      }
      this.isCreatingNewSource = false
    }
  },
  methods: {
    createNewDataSource(sourceName) {
      this.selectedSource = {
        status: {
          id: sourceName,
          running: false
        },
        address: '0.0.0.0',
        port: '8080'
      }
      this.isCreatingNewSource = true
      this.sensors = [tempSensor]
      this.dataSources.push(this.selectedSource.status)
    },
    async loadSensors(dataSourceJSON) {
      // split string on the form '<IdHHHddI' into a type array to be applied to sensors
      const dataTypes = dataSourceJSON.input_byte_format.substring(1).split('')
      const output_refs = dataSourceJSON.output_refs
      this.sensors = dataSourceJSON.input_names.map((it, index) => ({
        name: it,
        type: dataTypes[index],
        selected: output_refs.includes(index)
      }))
    },
    async fetchDataSources() {
      const dataSourcesResponse = await fetchDataSources()
      return Object.entries(dataSourcesResponse).map(source => ({
        id: source[0],
        running: source[1]
      }))
    },
    async loadDataSource(sourceId) {
      this.isLoadingSensors = true
      const dataSourceLink = apiEndPoint + 'datasources/' + sourceId
      const dataSourceJSON = await getJSONResponse(dataSourceLink)
      if (!dataSourceJSON) {
        this.$root.displayPopup("Couldn't fetch sensors")
        this.isLoadingSensors = false
        return
      }
      this.selectedSource.address = dataSourceJSON.addr[0]
      this.selectedSource.port = dataSourceJSON.addr[1]
      await this.loadSensors(dataSourceJSON)
      this.isLoadingSensors = false
    },
    async saveAndGoToHome (timeSensor) {
      await this.saveDataSource(timeSensor)
      this.$router.push({ name: 'Home' })
    },
    async saveDataSource(timeSensor) {
      // eslint-disable-next-line
      let formData = new FormData()
      formData.append('id', this.selectedSource.id)
      formData.append('address', this.selectedSource.address)
      formData.append('port', this.selectedSource.port)
      // Append output channels
      const indexOfTime = this.sensors.indexOf(timeSensor)
      formData.append('time_index', indexOfTime)
      this.sensors.forEach(sensor => {
        formData.append('output_name', sensor.name)
      })
      this.sensors.forEach((sensor, index) => {
        if (sensor.selected) {
          formData.append('output_ref', index)
        }
      })
      const byteFormatString =
        '<' +
        this.sensors
          .map(it => it.type)
          .toString()
          .replace(/,/g, '')
      formData.append('byte_format', byteFormatString)
      try {
        const createResponse = await createDataSource(formData)
        if (createResponse.ok) {
          let message =  'Datasource was created'
          if (this.selectedSource.status.running) {
            message += ', changes will be applied on rerun'
          } else {
            const startResponse = await startDataSource(this.selectedSourceID)
            startResponse.ok ? message += ' and started running' : ''
          }
          this.$root.displayPopup(message)
        }
      } catch (error) {
        this.$root.displayPopup('Network Error: ' + error.toString())
      }
    }
  }
}
</script>
<style scoped>
.port-and-address {
  margin: 5px 5px 0 5px;
}
</style>

```

```
import { EventBus, EVENTS } from '../js/EventBus'
import { subscribeToSource, unSubscribeSource } from '../api/APIHelper'
import { deepCopy } from '../utils/util'

const struct = require('@aksel/structjs')

let decoder = new TextDecoder('utf-8')

function byteToString (buf) {
  return decoder.decode(buf)
}

export default {
  data () {
    return {
      socketEndpoint: 'ws://129.241.90.187:1337',
      isOpen: false,
      isConnecting: false,
      sourceBuffers: {},
      pushDataIntervalID: undefined,
      packetCounter: 0,
      prevSubscriptions: []
    }
  },
  computed: {
    subscribedSources () {
      return this.$store.state.channelModule.sourceDict
    }
  },
  watch: {
    isOpen (isOpen) {
      isOpen ? this.initWebSocketConnection() : this.closeWebSocketConnection()
    }
  },
  created () {
    EventBus.$on(EVENTS.subscribe, this.subscribeDataSources)
  },
  beforeDestroy () {
    EventBus.$off(EVENTS.subscribe, this.subscribeDataSources)
  },
  methods: {
    async subscribeDataSources (sources) {
      const removedSources = this.prevSubscriptions.filter((sub) => !sources.some((source) => source.id === sub.id))
      const addedSources = sources.filter(
        (source) => !this.prevSubscriptions.some((prevSub) => prevSub.id === source.id)
      )
      removedSources.forEach((source) => {
        unSubscribeSource(source.id)
      })
      addedSources.forEach((source) => {
        subscribeToSource(source.id)
      })
      this.prevSubscriptions = sources.map((source) => ({
        id: source.id,
        url: source.url
      }))
      if (!this.isOpen) {
        await this.fetchAuthCookie()
        await this.initWebSocketConnection()
      } else {
        this.initBuffers()
      }
    },
    async fetchAuthCookie () {
      await fetch('http://129.241.90.187:1337/session', {
        credentials: 'include'
      })
    },
    async initWebSocketConnection () {
      // eslint-disable-next-line no-undef
      if (this.ws) return
      this.ws = new WebSocket(this.socketEndpoint)
      this.isConnecting = true
      this.ws.binaryType = 'arraybuffer'
      this.ws.onopen = () => {
        this.isConnecting = false
        this.isOpen = true
      }
      this.ws.onerror = () => {
        this.isConnecting = false
        this.$root.displayPopup("Couldn't reach server")
        this.isOpen = false
      }
      this.ws.onclose = () => {
        this.isOpen = false
        clearInterval(this.pushDataIntervalID)
      }
      this.initParser()
      this.ws.onmessage = (event) => {
        if (event.data.byteLength > 0) {
          const data = event.data
          const sourceID = byteToString(new Uint8Array(data, 0, 4))
          this.parseData(data.slice(4), sourceID)
        } else {
          console.log('pong')
        }
      }
    },
    closeWebSocketConnection () {
      this.ws.close(1000, 'Deliberate disconnection')
      clearInterval(this.pushDataIntervalID)
      this.ws = null
    },
    parseData (data, sourceID) {
      const sourceBuffer = this.sourceBuffers[sourceID]
      if (sourceBuffer === undefined) {
        return
      }
      this.packetCounter++
      const unpacker = sourceBuffer.unpacker
      const unpackIterator = unpacker.iter_unpack(data)
      let unpacked = unpackIterator.next().value
      while (unpacked) {
        sourceBuffer.x_buffer.push(new Date(unpacked[0] * 1000))
        const channelsIds = this.subscribedSources[sourceID].channels.map((it) => it.id)
        channelsIds.forEach((channelID) => {
          sourceBuffer.y_buffer[channelID].push(unpacked[channelID + 1])
        })
        unpacked = unpackIterator.next().value
      }
    },
    async pushData () {
      if (this.packetCounter > 0) {
        EventBus.$emit(EVENTS.newData, deepCopy(this.sourceBuffers))
        this.resetBuffers()
        this.packetCounter = 0
      }
    },
    initParser () {
      this.initBuffers()
      this.pushDataIntervalID = setInterval(this.pushData, 100)
    },
    resetBuffers () {
      for (const sourceId in this.sourceBuffers) {
        let sourceBuffer = this.sourceBuffers[sourceId]
        sourceBuffer.x_buffer = []
        Object.keys(sourceBuffer.y_buffer).forEach((channelID) => {
          sourceBuffer.y_buffer[channelID] = []
        })
      }
    },
    initBuffers () {
      this.sourceBuffers = {}
      for (const sourceId in this.subscribedSources) {
        let sourceBuffer = {}
        const byteFormat = this.subscribedSources[sourceId].byteFormat
        sourceBuffer.unpacker = struct(byteFormat)
        const channels = this.subscribedSources[sourceId].channels
        sourceBuffer.x_buffer = []
        sourceBuffer.y_buffer = {}
        channels.forEach((channel) => {
          sourceBuffer.y_buffer[channel.id] = []
        })
        this.sourceBuffers[sourceId] = sourceBuffer
      }
    }
  }
}

```

File - C:\Users\Odd\vue-digtwin\src\mixins\channelHandler.js

```javascript
export default {
  data () {
    return {
      selectedChannels: []
    }
  },
  watch: {
    channels (newChannels) {
      // Remove channels that were unsubscribed from selected channels
      this.selectedChannels = this.selectedChannels.filter(selChannel => newChannels.some(subChannel => subChannel.name === selChannel.name))
    }
  },
  computed: {
    channels () {
      const channelsDict = this.$store.state.channelModule.sourceDict
      return Object.entries(channelsDict).flatMap(source => {
        return source[1].channels.map(channel => ({
          id: [source[0], channel.id],
          name: channel.name
        }))
      })
    }
  }
}

```

File - C:\Users\Odd\vue-digtwin\src\mixins\routeNavigator.js

```
import { navRoutes } from '../router/navRoutes'

export default {
  data () {
    return {
      menu: navRoutes
    }
  },
  methods: {
    navigateTo (routePath) {
      this.$router.push(routePath)
    }
  }
}

```

File - C:\Users\Odd\vue-digtwin\src\mixins\processorLoader.js

```
import { getJSONResponse } from '../utils/util'
import { getOutputNames, rootAPI, startProcessorRequest } from '../api/APIHelper'
import { createProcessorInputFormData, createProcessorOutputFormData } from '../api/formDataCreator'
import { extractProcessorIOs } from '../api/extractor'

const processorEndpoint = rootAPI + '/processors/'

export default {
  props: {
    processorID: String
  },
  data: () => ({
    isStarted: true,
    startParams: {},
    processor: {},
    dataSourceChannels: [],
    isLoading: false
  }),
  watch: {
    processorID: {
      immediate: true,
      handler (processorID) {
        if (processorID !== '') {
          this.loadProcessor(processorID)
        }
      }
    }
  },
  methods: {
    async startProcessor () {
      let formData = this.getStartForm()
      try {
        this.isLoading = true
        const response = await startProcessorRequest(formData)
        if (!response.ok) {
          this.$root.displayPopup('Server error: ' + response.statusText)
          this.isLoading = false
          return
        }
        this.$root.displayPopup('Process Started!')
        // Get back to frontpage as you're finished
        this.$router.push({ name: 'Home' })
      } catch (error) {
        this.$root.displayPopup('Network Error' + error)
      }
      this.isLoading = false
    },
    getStartForm () {
      let formData = new FormData()
      formData = createProcessorInputFormData(this.processor.inputs)
      formData = createProcessorOutputFormData(this.processor.outputs, formData)
      formData.append('id', this.processorID)
      formData.append('start_params', JSON.stringify(this.startParams))
      return formData
    },
    async findStart (processorJSON) {
      let newParams = {}
      const blueprint = processorJSON.blueprint_id
      const startParamResponse = await getJSONResponse(
        rootAPI + '/blueprints/' + blueprint
      )
      startParamResponse.start_params.forEach(element => {
        newParams[element.name] = element.default
      })
      this.startParams = newParams
    },
    async loadProcessor (processorID) {
      const processorJSON = await getJSONResponse(
        processorEndpoint + processorID
      )
      const processorIOs = extractProcessorIOs(processorJSON)
      if (!processorJSON.started) {
        this.findStart(processorJSON)
        this.isStarted = false
      }
      if (processorIOs) {
        this.processor = processorIOs
        this.dataSourceChannels = await getOutputNames(
          '/topics/' + processorJSON.source_topic
        )
      }
    }
  }
}

```

File - C:\Users\Odd\vue-digtwin\src\router\router.js

```
import Vue from 'vue'
import Router from 'vue-router'
import { navRoutes } from './navRoutes'

Vue.use(Router)

export default new Router({
  mode: 'history',
  routes: navRoutes
})

```

```
import { lazyLoadView } from '../utils/vueutils'
import LayoutSaver from '../store/custom/dashboardLayoutSaver'

// Put routes that should be displayed in the navigation menu here
export const navRoutes = [
  {
    path: '/',
    name: 'Home',
    component: () => lazyLoadView(import('../views/Home'))
  },
  {
    path: '/dashboard',
    component: () => lazyLoadView(import('../views/Dashboard')),
    redirect: () => {
      // Redirect to new dashboard if there is no last selected layout, else load last selected layout
      const lastSelection = LayoutSaver.getLastSelection()
      if (!lastSelection) {
        return { name: 'NewDashboard' }
      }
      return { name: 'DashboardLayout', params: { layoutId: lastSelection } }
    },
    name: 'Dashboard',
    children: [
      {
        path: 'layout/:layoutId',
        name: 'DashboardLayout',
        component: () => lazyLoadView(import('@components/DashboardGrid')),
        props: true
      },
      {
        path: 'newlayout',
        name: 'NewDashboard',
        component: () => lazyLoadView(import('@components/DashboardGrid')),
        props: { isCreatingNewLayout: true, layoutId: 'new layout' }
      }
    ]
  },
  {
    path: '/datasources',
    name: 'Datasources',
    component: () => lazyLoadView(import('../views/DataSourceEditor'))
  },
  {
    path: '/Processors',
    component: () => lazyLoadView(import('../views/Processor')),
    children: [
      {
        path: '',
        name: 'Processors',
        component: () => lazyLoadView(import('../views/Processors'))
      },
      {
        path: 'create',
        name: 'ProcessorCreate',
        component: () => lazyLoadView(import('../components/forms/ProcessorCreator'))
      },
      {
        path: ':processorID',
        name: 'ProcessorEdit',
        component: () => lazyLoadView(import('../views/ProcessorEdit'))
      }
    ]
  },
  {
    path: '/about',
    name: 'About',
    // route level code-splitting
    // this generates a separate chunk (about.[hash].js) for this route
    // which is lazy-loaded when the route is visited.
    component: () => import(/* webpackChunkName: "about" */ '../views/About.vue')
  }
]

```

File - C:\Users\Odd\vue-digtwin\src\plugins\vuetify.js

```
import Vue from 'vue'
import Vuetify from 'vuetify/lib'
import 'vuetify/src/stylus/app.styl'
import VTooltip from 'vuetify/lib/components/VTooltip/VTooltip'
import VIcon from 'vuetify/lib/components/VIcon/VIcon'
import VBtn from 'vuetify/lib/components/VBtn/VBtn'
import VLayout from 'vuetify/lib/components/VGrid/VLayout'

Vue.use(Vuetify, {
  // Registering vuetify components for usage in functional components, NB!: Make sure that they're in use
  // or else unnecessary code is bundled in production
  components: { VLayout, VTooltip, VIcon, VBtn },
  iconfont: 'md'
})

```

```
<template>
  <v-layout row align-center>
    <input type="text" ref="nameInput" v-model="inputValue" @focus="$event.target.select()" />
    <v-btn @click="emitSaveEvent">Save layout</v-btn>
  </v-layout>
</template>

<script>
  export default {
    name: 'SaveBox',
    props: ['value'],
    computed: {
      inputValue: {
        get () {
          return this.value
        },
        set (newVal) {
          this.$emit('input', newVal)
        }
      }
    },
    methods: {
      emitSaveEvent () {
        // Prevent saving as empty string
        if (this.$refs.nameInput.value !== '') {
          this.$emit('save-click')
        }
      }
    }
  }
</script>

<style scoped>
  input {
    width: auto;
    text-align: center;
    border-style: inset;
    border-color: slategrey;
    border-radius: 3px;
  }

  input:focus {
    background: blanchedalmond;
  }
</style>

```

File - C:\Users\Odd\vue-digtwin\src\components\AppToolbar.vue

```
<template>
  <v-toolbar app>
    <navigation-menu> </navigation-menu>
    <v-layout row align-center>
      <page-menu> </page-menu>
      <v-spacer />
      <slot name="right-top-corner"/>
    </v-layout>
  </v-toolbar>
</template>

<script>
import PageMenu from './menus/PageMenu'
import NavigationMenu from './menus/NavigationMenu'

export default {
  name: 'AppToolbar',
  components: { NavigationMenu, PageMenu }
}
</script>

<style scoped>
</style>

```

```
<template>
  <v-layout column>
    <layout-grid
      :editable="false"
      :verticalCompact="true"
      :initLayout="gridLayout"
    ></layout-grid>
  </v-layout>
</template>
<script>
  import LayoutGrid from './layoutgrid/LayoutGrid'

  const gridLayout = [
    { 'x': 0, 'y': 0, 'w': 4, 'h': 1, 'i': '0', type: 'MarkerPlot', props: { title: 'CraneShort' } },
    { 'x': 0, 'y': 1, 'w': 4, 'h': 1, 'i': '1', type: 'Visualizer' }
  ]

  export default {
    name: 'StaticGrid',
    components: {
      LayoutGrid
    },
    data () {
      return {
        gridLayout: gridLayout
      }
    },
    methods: {
      async subscribe () {
        // eslint-disable-next-line no-undef
        await fetch('http://129.241.90.187:1337/' + 'datasources/')
      }
    }
  }
</script>

<style scoped>

</style>

```

```
<template>
  <v-layout row align-center>
    <v-checkbox v-model="activated" :color="color" style="max-width: 30px"></v-checkbox>
    <v-text-field class="text-field-input"
                  :disabled="!activated"
                  v-model="markerYCoord"
                  type="number"
                  label="Horizontal Line"
                  placeholder="Y-Coordinate"
                  @keyup.enter="notifyValueChanged(markerYCoord)"
    ></v-text-field>
    <v-menu :disabled="!activated">
      <v-btn :disabled="!activated" flat icon :color="color" slot="activator">
        <v-icon>
          color_lens
        </v-icon>
      </v-btn>
      <v-list>
        <v-list-tile
          v-for="(color, index) in availableColors"
          :key="index"
          @click="setColor(color)"
        >
          <v-list-tile-content>
            <v-list-tile-title>{{ color }}</v-list-tile-title>
          </v-list-tile-content>
        </v-list-tile>
      </v-list>
    </v-menu>
  </v-layout>
</template>

<script>
  export default {
    name: 'MarkerInput',
    props: {
      onValueChanged: {
        type: Function,
        required: true
      },
      onDeactivated: {
        type: Function,
        required: true
      }
    },
    data () {
      return {
        activated: false,
        color: 'blue',
        markerYCoord: 0.0,
        availableColors: ['red', 'blue', 'orange', 'yellow']
      }
    },
    watch: {
      activated (val) {
        if (val) {
          this.notifyValueChanged()
        } else {
          this.onDeactivated()
        }
      }
    },
    methods: {
      notifyValueChanged () {
        if (this.activated) {
          this.onValueChanged(this.markerYCoord, this.color)
        }
      },
      setColor (color) {
        this.color = color
        if (this.activated) {
          this.notifyValueChanged()
        }
      }
    }
  }
</script>

<style scoped>
.text-field-input{
  max-width: 100px;
}
</style>
```

```
<template>
  <v-layout column>
    <v-card class="sensorCard">
      <v-card-text class="sensorList" ref="listOfSensors">
        <v-data-table
          :loading="isLoadingSensors"
          hide-actions
          :headers="headers"
          :items="sensors"
          class="elevation-1"
        >
          <template #items="props">
            <td>
              <v-checkbox v-model="props.item.selected" />
            </td>
            <td>
              <v-edit-dialog :return-value.sync="props.item.name" lazy>
                {{ props.item.name }}
                <template #input>
                  <v-text-field
                    v-model="props.item.name"
                    label="Edit"
                    single-line
                    autofocus
                  ></v-text-field>
                </template>
              </v-edit-dialog>
            </td>
            <td>
              <v-select
                :items="allowTypes"
                v-model="props.item.type"
                hide-selected
                style="max-width: 50px"
                height="20px"
                flat
              >
              </v-select>
            </td>
            <td class="text-xs-left">
              <v-icon small class="mr-2" @click="deleteSensor(props.item)">
                delete
              </v-icon>
            </td>
          </template>
          <template #no-data>
            <v-alert
              v-show="!isLoadingSensors"
              :value="true"
              color="error"
              icon="warning"
            >
              No channels found for source
            </v-alert>
          </template>
        </v-data-table>
      </v-card-text>
      <v-card-actions class="buttonRow">
        <v-layout row align-center>
          <v-btn @click="addSensor({ name: '', type: '' })">Add Sensor</v-btn>
          <v-spacer />
          <v-select
            label="Time Sensor"
            v-model="timeSensor"
            :items="sensors"
            item-text="name"
            return-object
            :rules="[v => !!v || 'A time sensor has to be defined']"
          />
          <v-spacer />
          <v-btn
            :disabled="Object.keys(timeSensor).length === 0"
            @click="$emit('save-click', timeSensor)"
            >{{ isEditing ? 'Create' : 'Save' }}</v-btn
          >
          <v-btn
            :disabled="Object.keys(timeSensor).length === 0"
            @click="$emit('save-then-home', timeSensor)"
            >{{ isEditing ? 'Create' : 'Save' }} and go to home</v-btn
          >
        </v-layout>
      </v-card-actions>
    </v-card>
    <v-snackbar v-model="snack" :timeout="3000" :color="snackColor">
      {{ snackText }}
      <v-btn flat @click="snack = false">Close</v-btn>
    </v-snackbar>
  </v-layout>
</template>
<script>
import VDataTable from 'vuetify/lib/components/VDataTable/VDataTable'
import VTextField from 'vuetify/lib/components/VTextField/VTextField'
import VSelect from 'vuetify/lib/components/VSelect/VSelect'

export default {
  name: 'SensorTable',
  components: {
    VDataTable,
    VTextField,
    VSelect
  },
  model: {
    event: 'change',
    prop: 'value'
  },
  props: {
    value: Array,
    isLoadingSensors: Boolean,
    isEditing: Boolean
  },
  data() {
    return {
      headers: [
        { text: 'Should Output', value: 'selected', sortable: false },
        { text: 'Name', value: 'name', sortable: false },
        { text: 'Type', value: 'type', sortable: false }
      ],
      snack: false,
      snackColor: '',
      snackText: '',
      allowTypes: ['d', 'I', 'H'],
      showDialog: false,
      timeSensor: {}
    }
  },
  computed: {
    sensors: {
      get() {
        return this.value
      },
      set(newList) {
        this.$emit('change', newList)
      }
    }
  },
  methods: {
    addSensor(newSensor) {
      this.sensors.push(newSensor)
      this.$nextTick(() => this.listScrollToBottom())
    },
    listScrollToBottom() {
      let objDiv = this.$refs.listOfSensors
      objDiv.scrollTop = objDiv.scrollHeight
    },
    deleteSensor(item) {
      const index = this.sensors.indexOf(item)
      // eslint-disable-next-line
      confirm('Are you sure you want to delete this item?') &&
        this.sensors.splice(index, 1) &&
        this.displaySnack('red', 'Deleted ' + item.name)
    },
    displaySnack(color, text) {
      this.snack = true
      this.snackColor = color
      this.snackText = text
    }
  }
}
</script>
<style scoped>
.sensorList {
  overflow-y: auto;
  max-height: 70vh;
}
.sensorCard {
  margin-bottom: 5px;
}

.buttonRow {
  position: sticky;
  bottom: 0;
}
</style>

```

File - C:\Users\Odd\vue-digtwin\src\components\PopupMessage.vue

```
<template>
  <v-snackbar
    v-model="snackbar"
    :bottom="y === 'bottom'"
    :left="x === 'left'"
    :multi-line="mode === 'multi-line'"
    :right="x === 'right'"
    :timeout="timeout"
    :top="y === 'top'"
    :vertical="mode === 'vertical'"
  >
    {{ message }}
    <v-btn flat @click="snackbar = false">
      <v-icon>
        close
      </v-icon>
    </v-btn>
  </v-snackbar>
</template>

<script>
  import VSnackbar from 'vuetify/lib/components/VSnackbar/VSnackbar'

  export default {
    name: 'PopupMessage',
    components: { VSnackbar },
    data () {
      return {
        snackbar: false,
        y: 'top',
        x: null,
        mode: '',
        timeout: 6000,
        message: ''
      }
    },
    methods: {
      displayMessage (msg) {
        this.snackbar = true
        this.message = msg
      }
    }
  }
</script>

```

```
<template>
  <v-layout column>
    <layout-grid ref="LayoutGrid" :initLayout="initLayout" />
    <v-layout column class="componentSelectorControls" align-center>
      <drop-down-selector
        :options="items"
        :on-select="addNewItem"
        item-text="name"
      >
        <v-btn>
          <v-icon>library_add</v-icon>
        </v-btn>
      </drop-down-selector>
      <v-switch label="Controls" v-model="showLayoutMenu" />
    </v-layout>
    <v-layout row align-center v-show="showLayoutMenu" class="rightCornerControls">
      <layout-selector
        layout-id="currentLayoutId"
        ref="layoutSelector"
      />
      <v-btn @click="deleteLayout">Delete current layout</v-btn>
      <save-box v-model="currentLayoutId" @save-click="saveCurrentLayout" />
    </v-layout>
  </v-layout>
</template>
<script>
  import LayoutSelector from './selectors/LayoutSelector'
  import SaveBox from './SaveBox'
  import DropDownSelector from './selectors/DropDownSelector'
  import LayoutGrid from './layoutgrid/LayoutGrid'
  import LayoutSaver from '../store/custom/dashboardLayoutSaver'

export default {
    name: 'DashboardGrid',
    components: {
      LayoutGrid,
      DropDownSelector,
      SaveBox,
      LayoutSelector
    },
    props: {
      isCreatingNewLayout: {
        type: Boolean,
        default: false
      },
      layoutId: {
        type: String,
        required: true
      }
    },
    data () {
      return {
        initLayout: [],
        // holds the unsaved layout name
        tempId: undefined,
        showLayoutMenu: true,
        items: [
          { name: 'Plot',
            run: () => {
              this.$refs.LayoutGrid.addNewPlot()
            }
          },
          { name: 'MarkerPlot',
            run: () => {
              this.$refs.LayoutGrid.addNewPlot('MarkerPlot')
            }
          },
          { name: 'Statistics',
            run: () => {
              this.$refs.LayoutGrid.addNewItem('MarkerPlot')
            }
          },
          { name: 'Timeline',
            run: () => {
              this.$refs.LayoutGrid.addNewItem('Timeline')
            }
          }
        ]
      }
    },
    watch: {
      layoutId: {
        immediate: true,
        handler (newId) {
          // reset tempId on layout change
          this.tempId = undefined
          this.loadLayoutById(newId)
        }
      }
    },
    computed: {
      currentLayoutId: {
        get () {
          // return layoutId if tempId is undefined (i.e. user hasn't edited the layout id)
          return this.tempId !== undefined ? this.tempId : this.layoutId
        },
        set (newId) {
          this.tempId = newId
        }
      }
    },
    methods: {
      loadLayoutById (layoutId) {
        this.initLayout = LayoutSaver.getLayoutById(layoutId) || []
      },
      addNewItem (option) {
        option.run()
      },
      saveLayout () {
        LayoutSaver.saveLayout({
          id: this.tempId || this.currentLayoutId,
          newLayout: this.$refs.LayoutGrid.getCurrentLayout()
        })
        this.displayMessage(this.currentLayoutId + ' was saved')
        // Update url with new layout id
        this.$router.replace({ name: 'DashboardLayout', params: { layoutId: this.currentLayoutId } })
      },
      async saveCurrentLayout () {
        if (this.isCreatingNewLayout) {
          if (LayoutSaver.layoutExists(this.currentLayoutId)) {
            let msg = 'There already exists a layout with the name: ' +
              this.currentLayoutId + ', do you wish to overwrite it?'
            const shouldSave = await this.$confirm(msg, { title: 'Warning' })
            if (!shouldSave) {
              return
            }
          }
          this.saveLayout()
        } else {
          this.saveLayout()
        }
        this.$refs.layoutSelector.refreshLayoutIds()
      },
      displayMessage (msg) {
        this.$root.displayPopup(msg)
      },
      async deleteLayout () {
        let redirectLayoutId = LayoutSaver
          .deleteLayout(this.currentLayoutId)
        this.displayMessage(this.currentLayoutId + ' was deleted')
        // redirect to NewDashBoard if redirectLayoutId is undefined
        if (!redirectLayoutId) {
          this.$router.push({ name: 'NewDashboard' })
        } else {
          this.$router.push({ name: 'DashboardLayout', params: { layoutId: redirectLayoutId } })
        }
        this.$refs.layoutSelector.refreshLayoutIds()
      }
    }
  }
</script>

<style scoped>
  .componentSelectorControls {
    position: fixed;
    right: 5px;
    top: 6%;
    padding-right: 5px;
    background: rgba(150,150,150, 0.5);
    border-radius: 5px;
  }
  .rightCornerControls {
    position: fixed;
    right: 5px;
    bottom: 5px;
    opacity: 5%;
    padding-left: 5px;
    background: rgba(150,150,150, 0.5);
    border-radius: 10px;
  }
  .componentSelectorControls:hover ~ .rightCornerControls{
    background-color: limegreen;
  }
</style>

```

```
<template>
  <v-layout>
    <marker-input :onValueChanged="setNewMarker"
                  :onDeactivated="removeMarker">
    </marker-input>
  </v-layout>
</template>

<script>
  import MarkerInput from './MarkerInput'
  export default {
    name: 'MarkerControl',
    components: { MarkerInput },
    props: {
      onMarkerChange: {
        type: Function,
        required: true
      }
    },
    methods: {
      setNewMarker (markerYCoord, color) {
        this.onMarkerChange({ shapes: this.generateNewMarkerShape(markerYCoord, color) })
      },
      removeMarker () {
        this.onMarkerChange({ shapes: [] })
      },
      generateNewMarkerShape (yCoord, color) {
        return [{
          type: 'line',
          xref: 'paper',
          x0: 0,
          y0: yCoord,
          x1: 1,
          y1: yCoord,
          line: {
            color: color,
            width: 4,
            dash: 'dot'
          }
        }]
      }
    }
  }
</script>

<style scoped>

</style>
```

```
<template>
  <v-tooltip v-bind="$attrs">
    <slot slot="activator"></slot>
    <span>{{tooltip}}</span>
  </v-tooltip>
</template>

<script>
  export default {
    name: 'TooltipWrapper',
    props: {
      tooltip: {
        type: String,
        required: true
      }
    }
  }
</script>

<style scoped>

</style>
```

```
<template>
  <v-layout column>
    <v-layout row>
      <input-output-selection
        v-model="processor"
        :dataChannels="dataSourceChannels"
      />
      <div class="marginForStartParams" v-if="!isStarted">
        <span class="headline marginForStartParams">Start Parameters</span>
        <v-text-field
          v-for="input in Object.entries(startParams)"
          v-model="startParams[input[0]]"
          :key="input[0]"
          :label="input[0]"
        />
      </div>
    </v-layout>
    <v-layout row>
      <v-btn :disabled="!isStarted" @click="sendEditProcessorRequest"
        >Save</v-btn
      >
      <v-btn :disabled="isStarted" @click="startProcessor" color="success"
        >Start</v-btn
      >
      <v-btn
        :disabled="!isStarted"
        @click="sendStopProcessorRequest"
        color="info"
        >STOP</v-btn
      >
      <v-btn @click="sendDeleteProcessorRequest" color="error">DELETE</v-btn>
    </v-layout>
  </v-layout>
</template>

<script>
import InputOutputSelection from './forms/InputOutputSelection'
import { getJSONResponse } from '../utils/util'
import {
  createProcessorInputFormData,
  createProcessorOutputFormData
} from '../api/formDataCreator'
import {
  editProcessorIOs,
  getOutputNames,
  rootAPI
} from '../api/APIHelper'
import processorLoader from '../mixins/processorLoader'
import VTextField from 'vuetify/lib/components/VTextField/VTextField'

export default {
  name: 'SimulationEditor',
  components: { InputOutputSelection, VTextField },
  mixins: [ processorLoader ],
  methods: {
    sendEditProcessorRequest() {
      this.sendSetInputsRequest(this.processor.inputs)
      if (this.processor.selectedOutputs) {
        this.sendSetOutputsRequest(this.processor.selectedOutputs)
      }
    },
    sendStopProcessorRequest() {
      console.log("Trying to stop...")
      const request = new XMLHttpRequest()
      try {
      request.open('GET', rootAPI + '/processors/' + this.processorID + '/stop', true)
      request.onload = (response) => {
        this.$root.displayPopup('Process Stopped!')
        console.log(response)

      }
      request.send()

      } catch (error) {
        this.$root.displayPopup('Network Error')
      }
    },
    sendDeleteProcessorRequest() {
      const request = new XMLHttpRequest()
      try {
      request.open('GET', rootAPI + '/processors/' + this.processorID + '/delete', true)
      request.onload = (response) => {
        this.$root.displayPopup('Process Deleted!')
        console.log(response)

      }
      request.onerror = (error) => {
        this.$root.displayPopup(error)
      }
      request.send()

      } catch (error) {
        this.$root.displayPopup('Network Error')
      }
    },
    sendSetInputsRequest(inputs) {
      let formData = createProcessorInputFormData(inputs)
      if (formData) {
        this.sendRequest('inputs', formData)
      }
    },
    sendSetOutputsRequest(outputs) {
      let formData = createProcessorOutputFormData(outputs)
      this.sendRequest('outputs', formData)
    },
    async sendRequest(endPoint, formData) {
      console.log(endPoint)
      try {
        const response  = await editProcessorIOs(this.processorID, endPoint, formData)
        if (!response.ok) {
          this.$root.displayPopup('Couldn\'t set ' + endPoint)
          return
        }
        this.$root.displayPopup('Set ' + endPoint + ' successfully')
      } catch (error) {
        this.$root.displayPopup('Network Error: ' + error)
      }
    }
  }
}
</script>



<style scoped>
.marginForStartParams{
  margin-top: 5px;
  min-width: 150px;

}

</style>

```

File - C:\Users\Odd\vue-digtwin\src\components\ProgressHandler.vue

```
<template>
  <v-layout column>
    <v-alert v-if="value.showNoneFound" :value="true" color="error" icon="warning">
      {{ msg }}
    </v-alert>
    <v-progress-linear v-show="value.isLoading" color="primary" indeterminate/>
  </v-layout>
</template>

<script>
  export default {
    name: 'ProgressHandler',
    props: {
      value: Object,
      msg: String
    }
  }
</script>

<style scoped>

</style>

```

File - C:\Users\Odd\vue-digtwin\src\components\DragResizeContainer.vue

```
<template>
  <v-layout>
    <slot></slot>
    <v-layout column align-center class="controls">
      <tooltip-wrapper right tooltip="Drag item">
        <v-icon :disabled="dragDisabled" class="vue-draggable-handle">
          drag_handle
        </v-icon>
      </tooltip-wrapper>
      <control-button @click="lockUnlockDrag"
                      tooltip="Disable drag">
        {{ lockIcon }}
      </control-button>
      <slot name="extraControls"></slot>
    </v-layout>
  </v-layout>
</template>

<script>
  import TooltipWrapper from './TooltipWrapper'
  import ControlButton from './buttons/ControlButton'
  export default {
    name: 'DragResizeContainer',
    components: { ControlButton, TooltipWrapper },
    data () {
      return {
        dragDisabled: false
      }
    },
    computed: {
      lockIcon () {
        return this.dragDisabled ? 'lock' : 'lock_open'
      }
    },
    methods: {
      lockUnlockDrag () {
        this.dragDisabled = !this.dragDisabled
      }
    }
  }
</script>

<style scoped>

  .controls {
    background-color: aqua;
    border-radius: 5px;
    position: absolute;
    left: 1px;
    top: 5px;
  }

  .vue-draggable-handle:disabled {
    background: slategrey;
  }

  .vue-draggable-handle:enabled {
    background: white;
  }
</style>

```

File - C:\Users\Odd\vue-digtwin\src\components\DataConnectionMonitor.vue

```
<script>

import VSwitch from 'vuetify/lib/components/VSwitch/VSwitch'
import VProgressCircular from 'vuetify/lib/components/VProgressCircular/VProgressCircular'
import VLayout from 'vuetify/lib/components/VGrid/VLayout'

export default {
  name: 'DataConnectionMonitor',
  functional: true,
  model: {
    prop: 'isOpen',
    event: 'changed'
  },
  props: {
    isOpen: Boolean,
    isConnecting: Boolean
  },
  render (h, { props, listeners }) {
    return (
      <VLayout row class="wrapContentSwitch">
        <VSwitch class="wrapContentSwitch"
          color={props.isConnecting ? '' : 'green' }
          value={props.isOpen}
          onChange={listeners.changed}
          label={'Websocket ' + (props.isOpen ? 'open' : 'closed')}
          hide-details>
        </VSwitch>
        <VProgressCircular v-show={props.isConnecting} indeterminate />
      </VLayout>
    )
  }
}
</script>

<style scoped>
.wrapContentSwitch {
  max-width: 200px;
  width: 20vw;
}
</style>

```

```
<template>
  <v-layout column>
    <v-stepper :value="readyToCreate" vertical min-height="1000px">
      <v-stepper-header>
        <v-stepper-step :complete="readyToCreate > 1" step="1"
          >Select Blueprint and source</v-stepper-step
        >
        <v-divider></v-divider>
        <v-stepper-step :complete="readyToCreate > 2" step="2"
          >Select Inputs and Outputs</v-stepper-step
        >
        <v-divider></v-divider>

        <v-stepper-step :complete="readyToStart" step="3"
          >Add name and create Processor</v-stepper-step
        >
        <v-stepper-step step="4">Start Processor</v-stepper-step>
      </v-stepper-header>
      <div v-if="!readyToStart">
        <v-select
          @change="findInitParams()"
          v-model="selectedblueprint"
          label="Select Blueprint"
          :items="blueprints"
        ></v-select>
        <v-select
          v-if="selectedblueprint == 'fmu'"
          v-model="selectedfmu"
          label="Select fmu"
          :items="fmus"
        >
        </v-select>
        <v-select
          v-model="selectedTopic"
          label="Select Source"
          :items="Object.keys(sources)"
        ></v-select>
        <span v-if="readyToCreate > 1">Set input parameters</span>
        <div v-if="readyToCreate > 1">
          <v-text-field
            v-for="initKey in Object.keys(initParams)"
            v-model="initParams[initKey]"
            :hint="initDocs[initKey]"
            :key="initKey"
            :label="initKey"
          ></v-text-field>
        </div>
        <v-text-field
          v-model="processId"
          v-if="readyToCreate > 1"
          label="Name of Processor"
        ></v-text-field>
        <v-btn
          @click="createProcessor()"
          :disabled="readyToCreate < 2"
          color="success"
          >Create</v-btn
        >
      </div>
      <div v-if="readyToStart">
        <ProcessorStarter :processorID="processorID" />
      </div>
    </v-stepper>
  </v-layout>
</template>

<script>
```

```
import routeNavigator from '@mixins/routeNavigator'
import { getJSONResponse } from '../../utils/util'
import {
  createProcessor,
  editProcessorIOs, fetchBlueprint, fetchBlueprints, fetchFMUs, fetchTopics,
  getOutputNames,
  rootAPI
} from '../../api/APIHelper'

import {
  createProcessorFormData,
  createProcessorInputFormData,
  createProcessorOutputFormData
} from '../../api/formDataCreator'
import InputOutputSelection from './InputOutputSelection'
import ProcessorStarter from '../menus/ProcessorStarter'
import { getInitDocs } from '../../api/extractor'
const apiEndPoint = rootAPI + '/'
export default {
  name: 'SimulationPage',
  components: { ProcessorStarter },
  data() {
    return {
      processorID: '',
      processor: {},
      dataSourceChannels: [],
      blueprints: [],
      fmus: [],
      sources: {},
      initParams: {},
      initDocs: '',
      selectedblueprint: undefined,
      selectedfmu: undefined,
      selectedTopic: undefined,
      processId: undefined,
      createType: undefined,
      created: false
    }
  },
  created() {
    this.fetchContent()
  },
  computed: {
    readyToCreate() {
      if (!this.selectedblueprint || !this.selectedTopic) {
        return 1
      } else if (this.selectedblueprint === 'fmu') {
        return 2
      } else {
        return 3
      }
    },
    readyToStart() {
      return this.created
    }
  },
  methods: {
    async fetchContent () {
      this.fmus = await fetchFMUs()
      this.blueprints = await fetchBlueprints()
      await this.fetchDataSources()
    },
    async fetchDataSources() {
      const jsonResponse = await fetchTopics()

      let newSources = {}
      Object.entries(jsonResponse).forEach(source => {
        let nameOfSource = source[1].url.split('/').pop()
        newSources[nameOfSource] = source[0]
      })
      this.sources = newSources
    },
    async createProcessor() {
     const formData = createProcessorFormData(this.processId, this.selectedblueprint, this.initParams,
       this.sources[this.selectedTopic])
      try {
        const response = await createProcessor(formData)
        if (!response.ok) {
          this.$root.displayPopup('error: ' + response.statusText)
          return
        }
        this.$root.displayPopup('Process created!')
        // Set up for starting the processor
        this.created = true
        // Set the processorID for the the processorStarter to trigger loading
        this.processorID = this.processId
      } catch (error) {
        this.$root.displayPopup('Network Error: ' + error)
      }
    },
    async findInitParams() {
      let newParams = {}

      const jsonResponse = await fetchBlueprint(this.selectedblueprint)
      this.initDocs = getInitDocs(jsonResponse)
      jsonResponse.init_params.forEach(element => {
        newParams[element.name] = element.default
      })
      this.initParams = newParams
    }
  }
}
</script>

```

```
<template>
  <v-container grid-list-md>
    <v-layout wrap>
      <v-flex xs12 sm6>
        <v-layout column align-center>
          <span class="headline">Inputs</span>
          <v-layout column class="InputList ScrollableList">
            <input-selector
              v-for="(input, index) in processor.inputs"
              :key="index"
              v-model="processor.inputs[index]"
              :title="input.name"
              :options="dataChannels"
            />
          </v-layout>
        </v-layout>
      </v-flex>
      <v-flex xs12 sm6>
        <v-layout column align-center>
          <span class="headline">Outputs</span>
          <SelectList
            class="ScrollableList"
            :multi="true"
            v-model="processor.selectedOutputs"
            :items="processor.outputs || []"
            item-text="name"
          />
        </v-layout>
      </v-flex>
    </v-layout>
  </v-container>
</template>
<script>
import InputSelector from '../selectors/InputSelector'
import SelectChannelsList from '../lists/SelectChannelsList'
import SelectList from '@components/lists/SelectList'
export default {
  name: 'InputOutputSelection',
  components: { SelectChannelsList, InputSelector, SelectList },
  props: {
    value: Object,
    dataChannels: Array
  },
  computed: {
    processor: {
      get () {
        return this.value
      },
      set (newVal) {
        this.$emit('input', newVal)
      }
    }
  }
}
</script>

<style scoped>
.ScrollableList {
  max-height: 70vh;
  overflow-scrolling: auto;
  overflow-y: scroll;
  overflow-x: hidden;
}
.InputList {
  /* Making sure the scrollbar does not overlay content */
  padding-right: 5px;
  padding-left: 5px;
}
</style>

```

```
<template>
  <div>
    <v-list-tile v-if="multi" ripple @click="toggleSelectAll">
      <v-list-tile-action>
        <v-icon :color="selectedIndices.length > 0 ? 'blue darken-3' : ''">{{
          icon
        }}</v-icon>
      </v-list-tile-action>
      <v-list-tile-content>
        <v-list-tile-title>Select All</v-list-tile-title>
      </v-list-tile-content>
    </v-list-tile>
    <v-divider />
    <v-list>
      <v-list-tile
        v-for="(item, index) in items"
        :key="index"
        @click="setSelectedItems(index)"
      >
        <v-list-tile-action>
          <v-checkbox
            :value="selectedIndices.includes(index)"
            @input="setSelectedItems(index)"
          ></v-checkbox>
        </v-list-tile-action>
        <v-list-tile-content>
          <v-list-tile-title>{{
            itemText ? item[itemText] : item
          }}</v-list-tile-title>
        </v-list-tile-content>
      </v-list-tile>
    </v-list>
  </div>
</template>

<script>
import VCheckbox from 'vuetify/lib/components/VCheckbox/VCheckbox'

export default {
  name: 'SelectList',
  components: { VCheckbox },
  props: {
    value: [Array, Object, String],
    multi: {
      type: Boolean,
      default: true
    },
    items: Array,
    itemText: {
      type: String,
      default: undefined
    }
  },
  data: () => ({
    selectedIndices: []
  }),
  watch: {
    selectedIndices(newIndices) {
      const newSelections = this.items.filter((item, index) =>
        newIndices.includes(index)
      )
      // emit an object instead of an array if mode is single
      this.$emit('input', this.multi ? newSelections : newSelections[0])
    },
    items(newItems) {
      // Call nextTick to ensure the checkbox values displayed update
      this.$nextTick(() => this.setAlreadySelectedIndices(newItems))
    }
  },
  computed: {
    allSelected() {
      return this.selectedIndices.length === this.items.length
    },
    someSelected() {
      return this.selectedIndices.length > 0 && !this.allSelected
    },
    icon() {
      if (this.allSelected) return 'check_box'
      if (this.someSelected) return 'indeterminate_check_box'
      return 'check_box_outline_blank'
    }
  },
  methods: {
    setSelectedItems(index) {
      if (this.multi) {
        const indexOfSelected = this.selectedIndices.indexOf(index)
        const alreadySelected = indexOfSelected !== -1
        alreadySelected
          ? this.selectedIndices.splice(indexOfSelected, 1) // Item is already selected unselect
          : this.selectedIndices.push(index)
      } else {
        const alreadySelected = this.selectedIndices[0] === index
        const newSelectedIndex = alreadySelected ? -1 : index
        // using $set for reactivity, simply setting the item won't trigger an update of the template
        this.$set(this.selectedIndices, 0, newSelectedIndex)
      }
    },
    // Finds the indices of items which have selected = true
    // and adds them to selected
    setAlreadySelectedIndices(newItems) {
      this.selectedIndices = []
      for (let i = 0; i < newItems.length; i++) {
        const item = newItems[i]
        if (item.selected) {
          this.selectedIndices.push(i)
        }
      }
    },
    toggleSelectAll() {
      this.$nextTick(() => {
        if (this.allSelected) {
          this.selectedIndices = []
        } else {
          this.selectedIndices = [...this.items.slice().keys()]
        }
      })
    }
  }
}
</script>

```

```
<template>
  <v-layout column>
    <img v-show="channels.length === 0"
         style="width: 100%"
         alt="image"
         src="https://i.kym-cdn.com/entries/icons/original/000/023/967/obiwan.jpg">
    <SelectList v-model="selectedChannels" :items="channels" item-text="channelName"/>
  </v-layout>
</template>

<script>
  import SelectList from './SelectList'
  import {getOutputNames} from '../../api/APIHelper'

  export default {
    name: 'SelectChannelsList',
    components: { SelectList },
    props: {
      value: Array,
      channels: Array
    },
    computed: {
      selectedChannels: {
        get (){
          return this.value || []
        },
        set (newSelections){
          this.$emit('input', newSelections)
        }
      }
    }
  }
</script>

<style scoped>

</style>

```

File - C:\Users\Odd\vue-digtwin\src\components\menus\PageMenu.vue

```
<template>
  <v-layout row align-center>
      <minimizeMenu
        @menuClick="option => navigateTo(option.route)"
        :menuOptions="menu"
      >
        <v-icon>
          expand_more
        </v-icon>
      </minimizeMenu>
    <layout-selector v-if="atDashboard"></layout-selector>
    <v-spacer />
  </v-layout>
</template>
<script>
import routeNavigator from '@mixins/routeNavigator'
import minimizeMenu from './MinimizeMenu'
import LayoutSelector from '../selectors/LayoutSelector'
export default {
  name: 'PageMenu',
  components: { minimizeMenu, LayoutSelector },
  mixins: [ routeNavigator ],
  data () {
    return {
      dashboard: [{ name: 'New Layout', route: '/dashboard/newlayout' }]
    }
  },
  watch: {
    '$route.name': {
      immediate: true,
      handler (newRoute) {
        this.updatePageMenu(newRoute)
      }
    }
  },
  methods: {
    updatePageMenu (routeName) {
      this.atDashboard = routeName ? routeName.includes('Dashboard') : false
      if (this.atDashboard) {
        this.menu = this.dashboard
      }
    }
  }
}
</script>

```

```
<template functional>
  <VTooltip bottom>
    <template #activator="{ on }">
      <VIcon v-on="on">info</VIcon>
    </template>
    <span class="TooltipStyle">
      <slot/>
    </span>
  </VTooltip>
</template>

<style scoped>
  .TooltipStyle{
    max-width: 200px;
  }
</style>

```

```
<template>
  <v-layout row wrap align-center>
    <v-toolbar-items class="hidden-sm-and-down">
      <v-btn
        style="text-transform: inherit;"
        v-for="(option, index) in menuOptions"
        :key="index"
        flat
        @click="menuClick(option)"
      >
        {{ option.name }}
      </v-btn>
    </v-toolbar-items>
    <drop-down-selector
      class="hidden-md-and-up"
      :options="menuOptions"
      :on-select="menuClick"
      item-text="name"
    >
      <v-btn flat icon>
        <slot />
      </v-btn>
    </drop-down-selector>
  </v-layout>
</template>
<script>
import DropDownSelector from '../selectors/DropDownSelector'
export default {
  name: 'minimizeMenu',
  components: { DropDownSelector },
  props: {
    menuOptions: {
      type: Array,
      required: true
    }
  },
  methods: {
    menuClick (option) {
      this.$emit('menuClick', option)
    }
  }
}
</script>

```

File - C:\Users\Odd\vue-digtwin\src\components\menus\NavigationMenu.vue

```
<template>
  <v-menu offset-y>
    <template #activator="{ on }">
      <v-toolbar-side-icon v-on="on"></v-toolbar-side-icon>
    </template>
    <v-list>
      <v-list-tile
        v-for="route in menu"
        :key="route.name"
        @click="navigateTo(route.path)"
      >
        <v-list-tile-title style="text-transform: capitalize">{{ route.path.substring(1) || route.name }}</v-list-tile-title>
      </v-list-tile>
    </v-list>
  </v-menu>
</template>

<script>
  import routeNavigator from '@mixins/routeNavigator'
  export default {
    name: 'NavigationMenu',
    mixins: [routeNavigator]
  }
</script>

```

File - C:\Users\Odd\vue-digtwin\src\components\menus\ProcessorStarter.vue

```
<template>
  <div>
    <v-text-field
      v-for="input in Object.entries(startParams)"
      v-model="startParams[input[0]]"
      :key="input[0]"
      :label="input[0]"
    />
    <input-output-selection
      v-model="processor"
      :dataChannels="dataSourceChannels"
    />
    <v-btn @click="startProcessor" color="success">Start</v-btn>
    <v-progress-linear v-show="isLoading" color="primary" indeterminate />
  </div>
</template>
<script>
  import InputOutputSelection from '../forms/InputOutputSelection'
  import VTextField from 'vuetify/lib/components/VTextField/VTextField'
  import processorLoader from '../../mixins/processorLoader'
  import {createProcessorInputFormData} from '../../api/formDataCreator'
  import {startProcessor} from '../../api/APIHelper'
  import ProgressHandler from '../ProgressHandler'

  export default {
    name: 'ProcessorStarter',
    components: { ProgressHandler, InputOutputSelection, VTextField },
    mixins: [ processorLoader ]
  }
</script>

```

File - C:\Users\Odd\vue-digtwin\src\components\buttons\ToggleButton.vue

```
<template functional>
  <v-tooltip top>
    <template #activator="{ on }">
      <v-btn v-on="on" icon @click="listeners.input(!props.isActive)">
        <v-icon :color="props.isActive ? 'red' : ''">
          {{
            props.isActive ? 'radio_button_checked' : 'radio_button_unchecked'
          }}
        </v-icon>
      </v-btn>
    </template>
      <span>
        {{ props.tooltip }}
      </span>
  </v-tooltip>
</template>
<script>
export default {
  model: {
    event: 'input',
    prop: 'isActive'
  },
  props: {
    isActive: {},
    tooltip: String
  }
}
</script>

```

```
<template>
  <tooltip-wrapper right :tooltip="tooltip">
    <v-btn flat icon @click="$emit('click')">
      <v-icon>
        <slot>
        </slot>
      </v-icon>
    </v-btn>
  </tooltip-wrapper>
</template>

<script>
  import TooltipWrapper from '../TooltipWrapper'
  export default {
    name: 'ControlButton',
    components: { TooltipWrapper },
    inheritAttrs: false,
    props: {
      tooltip: String
    }
  }
</script>

<style scoped>

</style>

```

File - C:\Users\Odd\vue-digtwin\src\components\dialogs\DialogWrapper.vue

```
<template>
  <v-dialog v-model="showDialog" max-width="600px">
    <v-card>
      <v-card-title>
        <span class="headline">{{ title }}</span>
      </v-card-title>
      <v-card-text>
        <slot></slot>
      </v-card-text>
      <v-divider inset />
      <v-card-actions>
        <v-btn color="blue darken-1" flat @click="close">Close</v-btn>
        <v-spacer></v-spacer>
        <slot name="extra-buttons"></slot>
      </v-card-actions>
    </v-card>
  </v-dialog>
</template>

<script>
  export default {
    name: 'DialogWrapper',
    model: {
      event: 'toggleDialog',
      prop: 'value'
    },
    props: {
      value: Boolean,
      title: String
    },
    computed: {
      showDialog: {
        get () {
          return this.value
        },
        set (hideOrShow) {
          this.$emit('toggleDialog', hideOrShow)
        }
      }
    },
    methods: {
      close () {
        this.showDialog = false
      }
    }
  }
</script>

<style scoped>
</style>

```

```
<template>
  <div>
    <v-btn @click="open" style="text-transform: inherit"
      >Select DataSources</v-btn
    >
    <dialog-wrapper title="Select Data sources" v-model="showDialog">
      <SourceSelector v-model="selectedSources" :topics="topics" />
      <v-tabs color="cyan" dark slider-color="yellow" centered>
        <v-tab
          v-for="source in selectedSources"
          :key="source.id"
          :href="'#' + source.id"
        >
          <h2>{{ source.url.split('/')[2] }}</h2>
        </v-tab>
        <v-tabs-items>
          <v-tab-item
            v-for="source in selectedSources"
            :key="source.url"
            :value="source.id"
          >
            <SelectChannelsList class="SelectChannelsList"
              v-model="source.selectedChannels"
              :channels="source.channels"
            />
          </v-tab-item>
        </v-tabs-items>
      </v-tabs>
      <template #extra-buttons>
        <v-btn flat @click="subscribeToChannels">Done</v-btn>
      </template>
    </dialog-wrapper>
  </div>
</template>

<script>
import DialogWrapper from './DialogWrapper'
import DropDownSelector from '../selectors/DropDownSelector'
import { EventBus, EVENTS } from '../../js/EventBus'
import {deepCopy, getJSONResponse} from '../../utils/util'
import SelectList from '../lists/SelectList'
import VSelect from 'vuetify/lib/components/VSelect/VSelect'
import SelectChannelsList from '../lists/SelectChannelsList'
import SourceSelector from '../selectors/SourceSelector'
import {fetchTopics, rootAPI} from '../../api/APIHelper'

// Bundles each output matrix as one channel to be selected instead of individual
// channels for each element in the matrix
// Returns a list of scalaraOutputs and matrixOutputs, where matrixOutputs are a list of
// objects like this: { channelName: matrix_name + _matrix, outputChannels: channelForMatrix }

function bundleMatrixOutput (allOutputs, matrixOutputRefs) {
  const matrixOutputs = Object.entries(matrixOutputRefs).map(matrixOutput => {
    const matrixOutputIndices = matrixOutput[1]
    let matrixChannels = []
    // Fetch matrixchannels from scalaroutputs which holds all the output channels when entering this function
    matrixOutputIndices.forEach(index => {
      const matrixChannel = allOutputs[index]
      matrixChannels.push(matrixChannel)
    })
    return {
      channelName: matrixOutput[0] + '_matrix',
      outputChannels: matrixChannels
    }
  })
  // Filter out scalar outputs based on if the name does not contain '_mXY' where X and Y are integers
  const scalarOutputs = allOutputs.filter(channel => !/_m\d\d/.test(channel.channelName))
  return scalarOutputs.concat(matrixOutputs)
}

// checks for output_names ending in '_mXX', where X is a number and bundles
// them into a matrix output selecteable channel
function makeChannels (topicJSON) {
  if (topicJSON.output_names === undefined) {
    return []
  }
  const matrixOutputRefs = topicJSON.matrix_outputs
  let allOutputs = topicJSON.output_names.map(
    (output_name, index) => ({ id: index, channelName: output_name }))
  if (matrixOutputRefs === undefined || Object.keys(matrixOutputRefs).length === 0) {
    // There exists no matrixOutputs return all outputs as they are
    return allOutputs
  }
  return bundleMatrixOutput(allOutputs, matrixOutputRefs)
}

function unBundleMatrixChannels (channels) {
  // unbundling matrix channels
  const matrixChannels = channels
    .filter(channel => channel.channelName.includes('_matrix'))
    .flatMap(channel => channel.outputChannels)
  // putting scalar channels together with the matrix ones
  return channels.filter(channel => !channel.channelName.includes('_matrix')).concat(matrixChannels)
}

export default {
  name: 'SelectDataSourceChannels',
  components: {
    SourceSelector,
    SelectChannelsList,
    DialogWrapper
  },
  data() {
    return {
      topics: [],
      // list of sources, example: { id: '0000', url: '/datasources/testrig' }
      // Will have a selectedChannels property if any channels are selected for the source
      selectedSources: [],
      showDialog: false
    }
  },
  methods: {
    async open() {
      this.showDialog = true
      await this.loadSources()
    },
    async subscribeToChannels() {
      const subscribeSources = this.selectedSources
        .filter(source =>
          source.selectedChannels !== undefined && source.selectedChannels.length > 0)
        .map(source => ({
          id: source.id,
          name: source.url.split('/')[2],
          byteFormat: source.byteFormat,
          url: source.url,
          subscribedChannels: unBundleMatrixChannels(source.selectedChannels)
        }))
      await this.$store.dispatch(
        'channelModule/generateDataSources',
        subscribeSources
      )
      this.showDialog = false
      EventBus.$emit(EVENTS.subscribe, subscribeSources)
    },
    async loadSources() {
      const topicsJSON = await fetchTopics()
      if (!topicsJSON) return
      this.topics = Object.entries(topicsJSON).map(topic => ({
        id: topic[0],
        url: topic[1].url,
        byteFormat: topic[1].byte_format,
        channels: makeChannels(topic[1]) || []
      }))
    }
  }
}
</script>

<style scoped>
  .SelectChannelsList{
    max-height: 50vh;
    overflow-y: scroll;
    overflow-x: hidden;
  }
</style>

```

```
<template>
  <v-layout row align-center class="InputRow">
    <v-select
      v-model="selectedDataSource"
      :label="title"
      :items="options"
      @change="onSelect"
      clearable
    />
    <VTextField
      v-model="processorInput.measurement_proportion"
      class="scaleFactorInput"
      type="number"
      label="Scale Factor"
    />
    <v-icon :color="selectionColor">
      input
    </v-icon>
  </v-layout>
</template>

<script>
import DropDownSelector from './DropDownSelector'
import VSelect from 'vuetify/lib/components/VSelect/VSelect'
import TooltipWrapper from '../TooltipWrapper'
import VTextField from 'vuetify/lib/components/VTextField/VTextField'

export default  {
  components: { TooltipWrapper, DropDownSelector, VSelect, VTextField },
  props: {
    options: Array,
    title: String,
    value: Object
  },
  data: () => ({
    selectedDataSource: ''
  }),
  computed: {
    processorInput: {
      get () {
        return this.value
      },
      set (newSelection) {
        this.$emit('input', newSelection)
      }
    },
    selectionColor () {
      return this.selectedDataSource ? 'green' : 'red'
    }
  },
  watch: {
    // Check if a datachannel is selected at the input and set it if so, when options changes
    options: {
      immediate: true,
      handler (newOptions) {
        if (newOptions.length > 0) {
          this.selectedDataSource = newOptions[this.processorInput.measurement_ref]
        }
      }
    }
  },
  methods: {
    onSelect (item) {
      this.processorInput.measurement_ref = this.options.indexOf(item)
    }
  }
}
</script>

<style scoped>
.scaleFactorInput {
  margin-left: 5px;
  margin-right: 5px;
  max-width: 120px;
}

.InputRow{
  margin-left: 5px;
  margin-right: 5px;
}

</style>

```

```
<template>
  <v-combobox
    v-model.lazy="selectedItem"
    :items="dataSources"
    :hide-no-data="!search"
    item-text="id"
    :label="label"
    :search-input.sync="search"
    return-object
  >
    <template #no-data>
      <v-list-tile>
        <span class="subheading">Create</span>
        <v-chip :color="'blue lighten-3'" label small>
          {{ search }}
        </v-chip>
      </v-list-tile>
    </template>
  </v-combobox>
</template>

<script>
export default {
  name: 'CreateSelector',
  model: {
    event: 'change',
    prop: 'value'
  },
  props: {
    dataSources: Array,
    value: Object
  },
  data() {
    return {
      search: null
    }
  },
  computed: {
    selectedItem: {
      get() {
        return this.value
      },
      set(newVal) {
        if (!newVal)
          return // Disallow empty string as name
        if (typeof newVal === 'string')
          this.$emit('create-new', newVal)
        else {
          this.$emit('change', newVal)
        }
      }
    },
    label() {
      return this.value ? 'DataSource' : 'Select a DataSource'
    }
  }
}
</script>

<style scoped></style>

```

File - C:\Users\Odd\vue-digtwin\src\components\selectors\LayoutSelector.vue

```
<template>
  <v-select
    label="Selected Layout"
    :items="layoutIds"
    v-model="selectedLayoutId"
    hide-selected
    class="SelectorStyle"
  >
    <template #selection="{ item }">
      <div class="SelectedItem">
        {{ item }}
      </div>
    </template>
  </v-select>
</template>

<script>
import LayoutSaver from '../../store/custom/dashboardLayoutSaver'
import VSelect from 'vuetify/lib/components/VSelect/VSelect'

export default {
  name: 'LayoutSelector',
  components: { VSelect },
  data() {
    return {
      layoutIds: LayoutSaver.getLayoutIds()
    }
  },
  computed: {
    selectedLayoutId: {
      get() {
        return this.$route.params.layoutId
      },
      set(newId) {
        this.setLayout(newId)
      }
    }
  },
  methods: {
    setLayout(id) {
      this.$router.push({
        name: 'DashboardLayout',
        params: { layoutId: id }
      })
    },
    refreshLayoutIds() {
      this.layoutIds = LayoutSaver.getLayoutIds()
    }
  }
}
</script>

<style scoped>
.SelectorStyle {
  max-width: 100px;
}
.SelectedItem {
  width: 100%;
  justify-content: center;
  text-align: center;
}
</style>

```

File - C:\Users\Odd\vue-digtwin\src\components\selectors\SourceSelector.vue

```
<template>
  <v-select
    v-model="selectedSources"
    label="Select Data Sources"
    multiple
    chips
    deletable-chips
    :items="topics"
    item-text="url"
    return-object
  >
    <template #selection="{ item, parent, selected }">
      <v-chip
        v-if="item === Object(item)"
        :color="getSourceColor(item.url)"
        :selected="selected"
        label
        small
      >
            <span class="pr-2">
              {{ item.url.split('/')[2] }}
            </span>
        <v-icon small @click="parent.selectItem(item)">close</v-icon>
      </v-chip>
    </template>
  </v-select>
</template>
<script>
  import VSelect from 'vuetify/lib/components/VSelect/VSelect'

  export default {
    name: 'SourceSelector',
    components: { VSelect },
    props: {
      value: Array,
      topics: {}
    },
    computed: {
      selectedSources: {
        get () {
          return this.value
        },
        set (newSources) {
          this.$emit('input', newSources)
        }
      }
    },
    methods: {
      // Displays data sources as blue boxes and processors as red when selected
      getSourceColor(sourceUrl) {
        return (
          (sourceUrl.includes('/datasources/') ? 'blue' : 'red') + ' lighten-3'
        )
      }
    }
  }
</script>

```

```
<template>
  <v-autocomplete
    :allow-overflow="false"
    label="Selected Channels"
    chips
    v-model="channelsSelected"
    :items="channels"
    deletable-chips
    small-chips
    item-text="name"
    multiple
    return-object
    clearable
  >
    <template #selection="{ item, parent, selected, index }">
        <v-chip v-if="index < numberOfDisplayedItems"
                :color="`blue lighten-3`"
                :selected="selected"
                label
                small
        >
        <span class="pr-2">
          {{ item.name }}
        </span>
          <v-icon small @click="parent.selectItem(item)">close</v-icon>
        </v-chip>
      <tooltip-wrapper v-if="index === numberOfDisplayedItems" top :tooltip="otherSelected">
        <span >
        (+ {{ channelsSelected.length - numberOfDisplayedItems }} others)
        </span>
      </tooltip-wrapper>
    </template>
  </v-autocomplete>
</template>

<script>
  import VAutocomplete from 'vuetify/lib/components/VAutocomplete/VAutocomplete'
  import TooltipWrapper from '../TooltipWrapper'

  export default {
    name: 'ChannelSelector',
    components: { TooltipWrapper, VAutocomplete },
    props: {
      channels: {
        type: Array
      },
      // for the v-model
      value: {
        type: Array
      },
      numberOfDisplayedItems: {
        type: Number,
        default: 4
      }
    },
    computed: {
      channelsSelected: {
        get () {
          return this.value
        },
        set (val) {
          this.$emit('input', val)
        }
      },
      // a string of selected channels that do not show due to numberOfDisplayedItems, split by ','
      otherSelected () {
        return this.channelsSelected.slice(this.numberOfDisplayedItems).map(channel => channel.name).join(', ')
      }
    }
  }
</script>

<style scoped>

</style>

```

File - C:\Users\Odd\vue-digtwin\src\components\selectors\DropDownSelector.vue

```
<template>
  <v-menu offset-y>
    <slot slot="activator">
    </slot>
    <v-list>
      <v-list-tile
        v-for="(item, index) in options"
        :key="index"
        @click="onSelect(item)">
        <v-list-tile-title v-if="itemText">{{ item[itemText] }}</v-list-tile-title>
        <v-list-tile-title v-else>{{ item }}</v-list-tile-title>
      </v-list-tile>
    </v-list>
  </v-menu>
</template>

<script>
  import VMenu from 'vuetify/lib/components/VMenu/VMenu'

  export default {
    name: 'DropDownSelector',
    components: { VMenu },
    props: {
      options: {
        type: Array,
        required: true
      },
      onSelect: {
        type: Function,
        required: true
      },
      itemText: {
        type: String,
        default: undefined
      }
    }
  }
</script>

<style scoped>

</style>
```

```
<template>
  <grid-layout
    :layout.sync="gridLayout"
    :col-num="4"
    :row-height="510"
    :is-draggable="true"
    :is-resizable="true"
    :is-mirrored="false"
    :responsive="false"
    :autoSize="true"
    :margin="[0, 0]"
    :vertical-compact="verticalCompact"
    :use-css-transforms="true"
  >
    <grid-item
      v-for="item in gridLayout"
      :key="item.i"
      :x="item.x"
      :y="item.y"
      :w="item.w"
      :h="item.h"
      :i="item.i"
      drag-allow-from=".vue-draggable-handle"
      drag-ignore-from=".no-drag"
    >
      <v-layout fill-height>
        <layout-grid-item :extraControls="editable"
                          :compType="item.type"
                          :properties="item.props"
                          @add-grid-item="copyItem"
                          @remove-grid-item="removeGridItem">
        </layout-grid-item>
      </v-layout>
    </grid-item>
  </grid-layout>
</template>

<script>
  import * as VueGridLayout from 'vue-grid-layout'
  import { deepCopy } from '../../utils/util'
  const LayoutGridItem = () => import('./LayoutGridItem')
  const templateLayout = [
    { x: 0,
      y: 0,
      w: 4,
      h: 1,
      i: 0,
      type: 'PlotComponent',
      props: { title: '' }
    }]

  export default {
    name: 'LayoutGrid',
    components: {
      LayoutGridItem: LayoutGridItem,
      GridLayout: VueGridLayout.GridLayout,
      GridItem: VueGridLayout.GridItem
    },
    props: {
      verticalCompact: {
        type: Boolean,
        default: false
      },
      initLayout: {
        type: Array,
        default: undefined
      },
    editable: {
      type: Boolean,
      default: true
    }
  },
  data () {
    return {
      gridLayout: [],
      uniqueIndex: Number
    }
  },
  watch: {
    initLayout: {
      immediate: true,
      handler (newLayout) {
        this.loadLayout(newLayout)
      }
    }
  },
  methods: {
    loadLayout (layout) {
      this.gridLayout = layout || deepCopy(templateLayout)
      this.initializeUniqueIndex()
    },
    initializeUniqueIndex () {
      if (this.gridLayout.length > 0) {
        let largestIndex = this.gridLayout.reduce((max, layoutItem) => {
          return layoutItem.i > max.i ? layoutItem.i : max.i
        })
        // set uniqueIndex to the length of gridlayout if it's null or undefined
        this.uniqueIndex = ++largestIndex || this.gridLayout.length
      }
    },
    getCurrentLayout () {
      return this.gridLayout
    },
    removeGridItem (index) {
      this.gridLayout.splice(index, 1)
    },
    copyItem (clickedIndex) {
      let clickedItem = this.gridLayout[clickedIndex]
      let newItem = deepCopy(clickedItem)
      newItem.i = this.uniqueIndex++
      // Insert a new plot behind the clicked one
      this.gridLayout.splice(clickedIndex + 1, 0,
        newItem
      )
    },
    addNewPlot (type = 'PlotComponent') {
      let clickedItem = this.gridLayout.last
      let newItem = {}
      if (!clickedItem) {
        // No items currently in the dashboard, use template
        newItem = deepCopy(templateLayout[0])
      } else {
        // Create a deep copy
        newItem = deepCopy(clickedItem)
      }
      newItem.i = this.uniqueIndex++
      newItem.type = type
      // Insert a new plot behind the clicked one
      this.gridLayout.push(newItem)
    },
    addNewItem (type) {
      let newItem = deepCopy(templateLayout[0])
      newItem.i = this.uniqueIndex++
      newItem.type = type
```

```
        this.gridLayout.push(newItem)
      }
    }
  }
</script>

<style scoped>

</style>

```

File - C:\Users\Odd\vue-digtwin\src\components\layoutgrid\LayoutGridItem.vue

```
<template>
  <drag-resize-container>
    <component :is="itemComp"
             class="no-drag"
             v-bind="properties"
    >
    </component>
    <v-layout v-if="extraControls" column slot="extraControls">
      <control-button tooltip="Remove"
                      @click="$emit('remove-grid-item')">
        close
      </control-button>
      <control-button tooltip="Copy Item"
                      @click="$emit('add-grid-item')">
        add_circle_outline
      </control-button>
    </v-layout>
  </drag-resize-container>
</template>

<script>
  import DragResizeContainer from '../DragResizeContainer'
  import ControlButton from '../buttons/ControlButton'
  import { lazyLoadComponent } from '@utils/vueutils'

  export default {
    name: 'LayoutGridItem',
    components: {
      ControlButton,
      DragResizeContainer
    },
    props: {
      compType: {
        type: String,
        required: true
      },
      properties: {
        type: Object
      },
      extraControls: {
        type: Boolean,
        default: true
      }
    },
    computed: {
      itemComp () {
        const itemName = this.compType
        return lazyLoadComponent(import('./griditems/' + itemName + '.vue'))
      }
    }
  }
</script>

<style scoped>

</style>

```

File - C:\Users\Odd\vue-digtwin\src\components\layoutgrid\griditems\Timeline.vue

```
<template>
  <v-timeline class="Timeline">
    <v-timeline-item v-for="n in eventLog" :key="n" color="red lighten-2" large>
      <template v-slot:opposite>
        <span>28.03.2019</span>
      </template>
      <v-card class="elevation-2">
        <v-card-title primary-title class="headline"
          >Failure event</v-card-title
        >
        <v-card-text>{{ n }} </v-card-text>
      </v-card>
    </v-timeline-item>
  </v-timeline>
</template>

<script>
export default {
  data () {
    return {
      eventLog: ['200 mPA', '220 mPA']
    }
  }

}
</script>

<style scoped>
.Timeline{margin-left: 75px; min-width: 450px}

</style>

```

File - C:\Users\Odd\vue-digtwin\src\components\layoutgrid\griditems\MarkerPlot.vue

```
<template>
  <plot-component
    :channels="channels"
    :title="title"
  >
    <template #layoutControls="{ updateLayout }">
      <marker-control :onMarkerChange="updateLayout" style="MarkerControl"/>
    </template>
  </plot-component>
</template>

<script>
  import PlotComponent from './PlotComponent'
  import MarkerControl from '@components/MarkerControl'
  export default {
    name: 'MarkerPlot',
    props: {
      channels: Array,
      title: String
    },
    components: { MarkerControl, PlotComponent }
  }
</script>

<style scoped>
.MarkerControl{
  min-width: 10px;
}
</style>

```

```
<template>
  <v-layout fill-height v-resize="resizeLayout">
    <canvas ref="ceetronCanvas"></canvas>
    <div class="modelOptions">
      <v-select
        label="Selected Model"
        v-model="selectedFMUModel"
        hide-selected
        :items="models"
      />
      <v-layout row align-center>
        <ToggleButton
          v-model="updateCallbackIsActive"
          tooltip="Toggle Update"
        />
        <v-select
          label="FMU source"
          v-model="selectedFMUSource"
          :items="fmuSources"
          hide-selected
          item-text="name"
          return-object
        />
      </v-layout>
      <v-select
        outline
        :items="modelStyles"
        v-model="selectedModelStyle"
        hide-selected
        label="Model Style:"
      >
      </v-select>
    </div>
  </v-layout>
</template>

<script>
import { DigitalTwin } from '../../../js/DigitalTwin'
import resize from 'vue-resize-directive'
import { EVENTS, EventBus } from '../../../js/EventBus'
import ToggleButton from '../../buttons/ToggleButton'
import { deepCopy } from '../../../../utils/util'
import { mapState } from 'vuex'
let App = require('../../../js/usg')

export default {
  name: 'Visualizer',
  components: { ToggleButton },
  directives: {
    resize
  },
  data() {
    return {
      selectedFMUModel: '',
      selectedFMUSource: {},
      updateCallbackIsActive: false,
      Crane: null,
      myApp: null,
      selectedModelStyle: 'surface',
      modelStyles: [
        'surface',
        'surface_mesh',
        'outline_mesh',
        'lines',
        'points',
        'outline'
      ]
    }
  },
  computed: {
    ...mapState('digTwinModule', ['models']),
    fmuSources() {
      const channelsDict = this.$store.state.channelModule.sourceDict
      return Object.entries(channelsDict)
        .filter(source => source[1].name.includes('_fmu'))
        .map(source => ({ id: source[0], name: source[1].name }))
    },
    modelParts() {
      return this.Crane
        ? this.Crane.parts.map((part, index) => ({
            id: index,
            name: part.replace('.json', '')
          }))
        : []
    },
    fmuOutputChannels() {
      const channelsDict = this.$store.state.channelModule.sourceDict
      const selectedFMUSource = channelsDict[this.selectedFMUSource.id]
      if (selectedFMUSource !== undefined) {
        // Create a deep copy in order to not mutate the sourceDict in store
        const selectedFMUSourceCopy = deepCopy(selectedFMUSource)
        return selectedFMUSourceCopy.channels
          .filter(channel => /_m\d\d/.test(channel.name))
          .map(channel => {
            channel.name = channel.name.replace(/_m\d\d/)
            channel.partId = this.modelParts.filter(part =>
              channel.name.includes(part.name)
            )[0].id
            return channel
          })
      }
      return []
    },
    partIdsToUpdate() {
      return [...new Set(this.fmuOutputChannels.map(channel => channel.partId))]
    }
  },
  mounted() {
    // Should perhaps look for other solutions, but this one works for now
    window.setTimeout(this.initCeetronCanvas, 10)
  },
  methods: {
    resizeLayout() {
      this.myApp.resizeLayout()
    },
    initCeetronCanvas () {
      this.myApp = App.startApp(this.$refs.ceetronCanvas)
    },
    initialiseUSG() {
      // Initialise USG module
      this.initCeetronCanvas()
      this.Crane = new DigitalTwin(this.selectedFMUModel, this.myApp)
    },
    setModelStyle(style) {
      this.myApp.setDrawStyle(style)
    },
    updateDisplacement(newData) {
      const fmuChannels = this.fmuOutputChannels
      const sourceID = this.selectedFMUSource.id
      // failsafe
      if (sourceID === undefined) {
        return
      }
      console.log('updating')
      const newDataYBuffer = newData[sourceID].y_buffer
      const paddIndices = [2, 5, 8, 11]
      for (let i = 0; i < this.partIdsToUpdate.length; i++) {
        let newDisp = []
        const partId = this.partIdsToUpdate[i]
        for (let j = 0; j < 12; j++) {
          newDisp.push(newDataYBuffer[fmuChannels[i * 12 + j].id][0])
          if (paddIndices.includes(j)) {
            newDisp.push(j === 11 ? 1 : 0)
          }
        }
        this.myApp.updateDisplacement(newDisp, partId)
      }
    }
  },
  watch: {
    selectedModelStyle(val) {
      if (val !== '')
        this.setModelStyle(val)
    },
    selectedFMUModel() {
      this.initialiseUSG()
    },
    updateCallbackIsActive(isActive) {
      isActive
        ? EventBus.$on(EVENTS.newData, this.updateDisplacement)
        : EventBus.$off(EVENTS.newData, this.updateDisplacement)
    }
  }
}
</script>

<style scoped>
.modelOptions {
  position: absolute;
  right: 0.1%;
  top: 1%;
  max-width: 200px;
}
</style>

```

```
<template>
  <v-layout column v-resize="relayout">
    <vue-plotly
      ref="plotlyDiv"
      :data="plotData"
      :layout="layout"
      :options="options"
    ></vue-plotly>
    <v-layout
      row
      justify-center
      align-center
      ref="controlsRow"
      style="background: palegreen; max-height: 68px"
    >
      <v-spacer>
        <slot name="layoutControls" :updateLayout="this.updateLayout"></slot>
      </v-spacer>
      <v-spacer></v-spacer>
      <channel-selector v-model="selectedChannels" :channels="channels" />
      <v-spacer></v-spacer>
      <VTextField v-model="maxPoints" label="Plot Density"/>
      <ToggleButton v-model="plotCallbackIsActive"
                    tooltip="Toggle Plotting" />
      <tooltip-wrapper top tooltip="Save plot">
        <v-btn icon @click="savePlot">
          <v-icon color="indigo lighten-">
            save
          </v-icon>
        </v-btn>
      </tooltip-wrapper>
    </v-layout>
  </v-layout>
</template>

<script>
import VuePlotly from '@statnett/vue-plotly'
import * as PlotSaver from '@utils/plotSaver'
import ChannelHandler from '@/mixins/channelHandler'
import ChannelSelector from '@components/selectors/ChannelSelector'
import resize from 'vue-resize-directive'
import { EventBus, EVENTS} from '../../../js/EventBus'
import VTextField from 'vuetify/lib/components/VTextField/VTextField'
import ToggleButton from '../../buttons/ToggleButton'
import TooltipWrapper from '../../TooltipWrapper'

export default {
  mixins: [ChannelHandler],
  components: {
    TooltipWrapper,
    ToggleButton,
    ChannelSelector,
    VuePlotly,
    VTextField
  },
  directives: {
    resize
  },
  props: {
    title: String,
    plotWidth: {
      type: Number
    }
  },
  data () {
    return {
      plotData: [{
        type: 'scattergl-visible',
        x: [],
        y: [],
        mode: 'lines'
      }],
      padArray: Array(5000).fill(1),
      plotCallbackIsActive: false,
      layout: {
        title: this.title,
        xaxis: { title: 'x' },
        yaxis: { title: 'y' }
      },
      // NB: Setting responsive to true will cause the optionsrow to jump on selection
      options: {responsive: true},
      newData: {},
      indicesToUpdate: [],
      maxPoints: 10000
    }
  },
  mounted () {
    // Should perhaps look for other solutions, but this one works for now
    window.setTimeout(this.relayout, 500)
  },
  watch: {
    plotCallbackIsActive(isActive) {
      isActive ?
        EventBus.$on(EVENTS.newData, this.dataReceivedCallback) :
        EventBus.$off(EVENTS.newData, this.dataReceivedCallback)
    },
    selectedChannels (newChannels, oldChannels) {
      this.plotData = newChannels.map(it => ({
        name: it.name,
        type: 'scattergl-visible',
        x: [],
        y: [],
        mode: 'lines'
      }))
      this.indicesToUpdate = [...Array(this.selectedChannels.length).keys()]
      if (newChannels.length > 0 && oldChannels.length === 0) {
        this.plotCallbackIsActive = true
        return
      }
      if (newChannels.length === 0) {
        this.plotCallbackIsActive = false
      }
    }
  },
  created: function () {
    window.addEventListener('resize', () =>
      window.setTimeout(this.relayout, 10)
    )
  },
  beforeDestroy: function () {
    window.removeEventListener('resize', this.relayout)
  },
  methods: {
    // Set the plotly containers width to match controlsRow
    relayout () {
      let parentWidth = this.$refs.controlsRow.offsetWidth
      this.$refs.plotlyDiv.relayout({ width: parentWidth })
    },
    // call relayout manually when watcher on layout property does not trigger >:C
    updateLayout (newProps) {
      this.$refs.plotlyDiv.relayout(newProps)
    },
    savePlot () {
      PlotSaver.save(this.selectedChannels[0])
    },
    dataReceivedCallback (newData) {
      this.newData = newData
      requestAnimationFrame(this.updatePlot)
    },
    updatePlot () {
      let newXValues = []
      let newYValues = []
      const newData = this.newData
      for (let i = 0; i <this.selectedChannels.length; i++) {
        const sourceChannelID = this.selectedChannels[i].id
        const newChannelData = newData[sourceChannelID[0]]
        newYValues.push(newChannelData.y_buffer[sourceChannelID[1]])
        newXValues.push(newChannelData.x_buffer)
      }
      this.$refs.plotlyDiv.extendTraces(
        {
          y: newYValues,
          x: newXValues
        },
        this.indicesToUpdate,
        this.maxPoints)
    }
  }
}
</script>
<style scoped>
  .ChannelSelector{
    max-height: 100px;
  }
</style>

```