Master's thesis

2020

Erik Kjernlie, Anne Pernille Wulff Wold

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

Master's thesis

NTNU
Norwegian University of
Science and Technology

Erik Kjernlie
Anne Pernille Wulff Wold

Developing a Cloud-Based Monitoring System for Digital Twins

June 2020

NTNU
Norwegian University of
Science and Technology

Developing a Cloud-Based Monitoring System for Digital Twins

Erik Kjernlie
Anne Pernille Wulff Wold

Engineering and ICT
Submission date: June 2020
Supervisor: Bjørn Haugen
Co-supervisor: Terje Rølvåg

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

Preface

This Master's thesis concludes our M. Sc. Engineering and ICT degrees at the Norwegian University of Science and Technology in Trondheim. The project is conducted at the Department of Mechanical and Industrial Engineering. It is the continuation of our Specialization thesis, "Evaluating the Cloud-Based Monitoring System for Further Development," from December 2019.

We would like to thank our supervisors Bjørn Haugen and Terje Rølvåg, for support and guidance throughout the project and introducing us to relevant people that have influenced the project. SAP Norway and Runar Heggelien, in particular, for the enthusiasm and taking time to answer all of our questions. Jan Christian Meyer, for discussions, technical insights, and valuable feedback. Bjørn Lindi and the rest of MIME's Brønn, for providing us with technical support.

It is assumed that the reader possesses a general understanding of the fields of information and communications technology (ICT) and mechanical engineering.

Trondheim, 10.06.2020

Erik Kjernlie

Erik Kjernlie

Anne P.W. Wold

Anne Pernille Wulff Wold

..

NTNU

Faculty of Engineering

Department of Mechanical and Industrial Engineering

MASTER'S THESIS 2020

FOR

STUD.TECHN. ERIK KJERNLIE

AND

STUD.TECHN. ANNE PERNILLE WULFF WOLD

DEVELOPING A CLOUD-BASED MONITORING SYSTEM FOR DIGITAL TWINS

Utvikling av skybasert monitoreringssystem for digitale tvillinger

Several software companies are developing digital twin solutions for predictive maintenance and monitoring of structural integrity. These are based on very expensive proprietary formats and solutions not applicable to academia and SME companies. NTNU/MTP is therefore developing a cloud-based monitoring system (CBMS) for integrity monitoring of physical structures and mechanisms. The CBMS is currently in a prototype phase and we want to benchmark this system on the MTP's knuckle boom crane.

Tasks include:

1. Build a user-friendly client in React
2. Facilitate support for personal and persisting projects
3. Implement a generic configuration system for easy adaption to other digital twin applications
4. Implement functionality for monitoring of physical assets
5. Implement methods for post-processing and analytics

If time permits:

6. Write a scientific digital twin paper with the supervisors

Contact:

At the department (supervisor, co-supervisor): Bjørn Haugen, Terje Rølvåg and Eilif Pedersen

From SAP: Runar H. Refsnæs and Henrik Løfaldli

Abstract

A wide range of businesses and industries worldwide are increasingly adopting digital twins to achieve more intelligent and automated manufacturing processes. Their applications expose more insight into physical assets, revolutionize outdated work processes and save companies time and money. Industrial equipment is instrumented with sensor technology, which enables continuous monitoring of assets. Monitoring of industrial equipment uncovers potentially harmful operating conditions, and the insights gained provides a better basis for making decisions about the system. By incorporating emerging technology trends such as data analytics, cloud computing, and machine learning, one can simulate remaining useful lifetime and optimize operations of assets.

The development of digital twin platforms for monitoring and predictive maintenance is a complex process, as it requires extensive knowledge about information, communication and sensor technologies, and expertise within the application domain. Most of these platforms are based on expensive proprietary formats, and are not applicable to academia and SME companies. At NTNU, there is an ongoing project at the Department of Mechanical and Industrial Engineering that aims to develop a cloud-based monitoring system (CBMS) for digital twins. The project is developed in multiple iterations by students at the department.

The authors' specialization project in the fall of 2019 (Kjernlie and Wold 2019) made a thorough evaluation of the state of the CBMS. This thesis further develops the CBMS based on findings from the specialization project, including building a completely new front-end solution and substantial extensions to the back-end. A database and an authentication service enable the creation of persisting projects. The platform is equipped with features to facilitate monitoring of physical assets, such as curve plots, video streaming, and dynamic maps. Users are notified of alarming sensor values by event triggers and predictions based on machine learning models. Fast Fourier transforms (FFTs) and spectrogram analyses expose changes in the structural integrity. Filters can be applied to remove noise from the signal. This thesis demonstrates that all the requirements are fulfilled. Future work is proposed for the next iteration of the development process. This work contributes a functional CBMS, and takes the project one step closer to the desired full-featured CBMS for digital twins.

Sammendrag

Et bredt spekter av virksomheter og næringer over hele verden tar i stadig større grad digitale tvillinger i bruk for å oppnå mer intelligente og automatiserte produksjonsprosesser. Bruksområdene deres gir mer innsikt i fysiske eiendeler, revolusjonerer utdaterte arbeidsprosesser og sparer selskaper tid og penger. Industrielt utstyr er utstyrt med sensorteknologi, som muliggjør kontinuerlig overvåking av eiendeler. Overvåking av industrielt utstyr avdekker potensielt skadelige driftsforhold, og innsikten som innhentes gir et bedre grunnlag for å ta beslutninger om systemet. Ved å innlemme nye teknologitrender som dataanalyse, skytjenester og maskinlæring, kan man simulere gjenværende nyttig levetid og optimalisere driften av eiendeler.

Utviklingen av digitale tvillingplattformer for overvåking og prediktiv vedlikehold er en kompleks prosess, ettersom den krever omfattende kunnskap om informasjons-, kommunikasjons- og sensorteknologier, og ekspertise innen applikasjonsdomenet. De fleste av disse plattformene er basert på dyre proprietære formater, og gjelder ikke akademia og SMB-selskaper. Ved NTNU er det et pågående prosjekt ved institutt for maskinteknikk og produksjon som har som mål å utvikle et skybasert overvåkingssystem (CBMS) for digitale tvillinger. Prosjektet er utviklet i flere iterasjoner av studenter ved instituttet.

Forfatternes fordypningsprosjekt høsten 2019 (sst.) foretok en grundig evaluering av status for CBMS-prosjektet. Denne avhandlingen videreutvikler CBMS basert på funn fra fordypningsprosjektet, inkludert å bygge en helt ny front-end-løsning og betydelige utvidelser til back-end. En database og en autentiseringstjeneste muliggjør opprettelse av vedvarende prosjekter. Plattformen er utstyrt med funksjoner for å lette overvåking av fysiske eiendeler, for eksempel kurveplott, videostreaming og dynamiske kart. Brukere blir varslet om alarmerende sensorverdier av *event triggere* og prediksjoner basert på maskinlæringsmodeller. Fast Fourier transformer (FFTs) og spektrogramanalyser avslører endringer i strukturell integritet. Filtre kan brukes for å fjerne støy fra signalet. Denne oppgaven viser at alle kravene er oppfylt. Framtidig arbeid foreslås for neste iterasjon av utviklingsprosessen. Dette arbeidet bidrar med en funksjonell CBMS, og tar prosjektet et skritt nærmere ønsket fullverdige CBMS for digitale tvillinger.

Abbreviations

AaaS	Authentication as a Service
API	Application Programming Interface
CBMS	Cloud based monitoring system
CP	Concurrent Probing
CSS	Cascading Style Sheets
CSV	Comma Separated Values
CTA	Concurrent Think Aloud
DAQ	Data Acquistion System
FFT	Fast Fourier Transform
FMI	Functional Mock-up Interface
FMU	Functional Mock-up Unit
GDPR	General Data Protection Regulation
GUI	Graphical User Interface
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
JSON	JavaScript Object Notation
MTP	Department of Mechanical and Industrial Engineering
MTU	Maximum transmission unit
NDA	Non-Disclosure Agreement
noSQL	not only SQL
PLM	Product Lifecycle Management
PoC	Proof of Concept
RTA	Retrospective think aloud
TCP	Transmission control protocol
UDP	User Datagram Protocol
SaaS	Software as a Service

Contents

List of Figures

List of Tables

Code Lisings

Chapter 1

Introduction

This chapter presents the background of the thesis, the long-term ambitions, and the previous work related to the project. The scope, objectives, and structure of the thesis are described.

1.1 Background

The fourth industrial revolution, often referred to as Industry 4.0, leverages the latest industry trends to achieve more intelligent and automated manufacturing processes (Zhou, Taigang Liu, and Lifeng Zhou 2015). The digital transformation utilizes advanced information and communication technology to increase productivity and enhance operations and products (Rosen et al. 2015).

Digital twins are an essential application of Industry 4.0 and the Industrial Internet of Things (ibid.). The idea is to use the digital twin to monitor and interact with the physical twin. Digital twins can be used to optimize the performance of real assets, using emerging technologies such as machine learning, big data, and cloud computing. Manufacturing processes are becoming more digitized, which makes digital twins a critical component for the fourth industrial revolution.

Digital twins are often represented in interactive platforms to capture and display real-time data and visualizations (NSW 2019). Digital twin platforms monitor and analyze equipment's health based on the sensor data sent from physical assets. The asset changes its behavior based on results from analyzes and processed data. The development of a digital twin platform requires comprehensive knowledge about information, communication and sensor technologies, which is why most of the existing platforms are proprietary and not well suited for use in academia and small enterprises.

1.2 The Cloud-Based Monitoring System

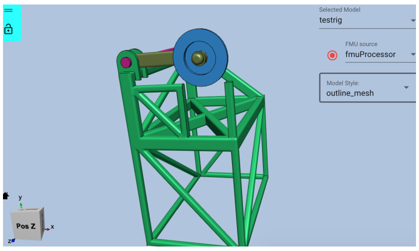
The department of Mechanical and Industrial Engineering at NTNU has an ongoing project that aims to develop an open-source Cloud-Based Monitoring System (CBMS) for educational purposes. The platform should reflect and monitor physical assets in real-time, predict potential failure modes, and notify users of the asset's state. It should be generic by using standardized interfaces to create and manage digital twins, and it should be able to handle multiple users and assets. The platform should visualize historical data and run simulations and analyses.

The process of developing and deploying an advanced system for monitoring of digital twins is time-consuming and labor-intensive. The system is developed iteratively in multiple master's theses, and this thesis is the second iteration of the development process. It continues the development of the platform and facilitates further development by other students. An essential part is to document how the system is implemented and the workflows during the implementation. The outcome includes a separate chapter dedicated to the *implementation*, and information about the development process in the *method* chapter. The *discussion* chapter contains suggestions for further development, and comprehensive guides for installing, deploying, and using the system are in the *appendix*.

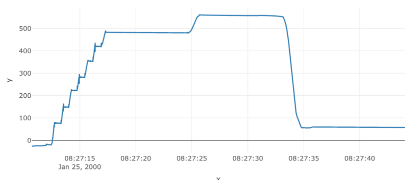
1.3 Previous Work

In the fall of 2018, a prototype of the platform was developed in a specialization project by students at the department (Jensen et al 2018). The following spring, three Master's theses started the development of the CBMS; a back-end development project that is related to the server-side of the platform (Jensen 2019), a front-end related to client-side (Børhaug and Sande 2019) and finally, a project that created a prototype of a configuration system for a more generic solution (Johansen 2019).

The back-end was developed by Jensen (2019) in Python. It is extendable, and provides support for Functional Mock-up Units (FMUs) along with filtering and fast Fourier transforms (FFTs). The back-end connects to the front-end via an *application programming interface* (API) made in collaboration with Børhaug and Sande (2019) to accommodate their needs for the front-end. The front-end supports real-time visualization of sensor data in a curve plot, and visualization of a 3D model of an asset is possible with a large delay. The implementation was customized for the torsion bar suspension rig. Johansen (2019) instrumented a physical asset and made a digital model in order to connect the physical asset to the system. Additionally, Johansen (ibid.) developed a prototype for a more generic configuration system for digital twins.



(a) 3D visualization of NTNU's torsion bar suspension rig rig



(b) Curve plot of the load applied to the torsion bar suspension rig for a given time series

Figure 1.1: Snapshots from the previous work on the front-end solution.

In the fall of 2019, a specialization project conducted at the department investigated the use of the Microsoft Azure platform as an alternative back-end to the CBMS (Sandtveit 2019). A prototype was developed and connected to a Raspberry Pi that transmitted sensor data. Azure is a commercial product, which means one has to consider the possible advantages against the cost. Azure is not integrated with this thesis due to the conclusion in Sandtveit (ibid.). In parallel with this project, a Master's thesis is investigating two-way communication between a CBMSand a physical asset using a Raspberry Pi.

Specialization Project

The authors' specialization project in the fall of 2019 (Kjernlie and Wold 2019) laid the groundwork for the scope of this thesis. The main objectives of the project were to

1. Carry out a thorough analysis of the state of the project
2. Consider if any of the modules were to be rebuilt
3. Make specification and requirements for further development of the CBMS

The analysis revealed that implementation of a database and an authentication service would enable users to create persistent projects, and that a system for configuring digital twins was required to create a more generic platform. The platform should accept standardized formats for virtual models and accept different data formats for the sensor data from physical assets. Furthermore, a list of tools to facilitate monitoring, post-processing, and analyzing physical assets was defined, including curve plots, filtering, event triggers, real-time predictions, and FFTs.

It was decided to keep the existing back-end due to its robustness and extendable structure. Implementation of the features listed in the previous paragraph requires extensions to the back-end, *e.g.*, communication with the database. It was decided to develop a new front-end with React from scratch since the application did not meet the stakeholders' requirements. Both the solutions lacked sufficient documentation, which is why providing comprehensive documentation was considered an essential element on the requirement list for this Master's thesis.

1.4 Scope of Thesis

The findings from the specialization project forms the foundation for the scope of this thesis. It includes continuing the development of the platform according to the specifications and requirements. Objectives include:

1. Build a user-friendly client in React[1]
2. Facilitate support for personal and persisting projects

[1]React is an open-source JavaScript library for building user interfaces.

3. Implement a generic configuration system for easy adaption to other digital twin applications
4. Implement functionality for monitoring of physical assets
5. Implement methods for post-processing and analytics

Build a user-friendly client in React The success of an application directly relates to how easy it is to use. A platform that is difficult to understand and navigate does not attract users, and it reduces the productivity of the application. A user-friendly application engages users and assists them in completing tasks efficiently and effectively. The users understand the purpose of the application and are able to use the implemented functionality.

Facilitate support for personal and persisting projects Enabling storage of sessions and projects saves the user a considerable amount of work in terms of setting up a project and configuring digital twins. A database facilitates the storage and management of data related to users and projects. A user is authorized to access their data after an authentication process to log into the system.

Implement a generic configuration system for easy adaption to other digital twin applications The platform should be available to anyone, and the end-user must be able to configure digital twins of their assets and data to be useful. The platform should leverage standardized formats of streaming data, models, and files to facilitate the configuration of digital twins that can be monitored and simulated in the platform.

Implement functionality for monitoring of physical assets Adequate tools should be implemented to monitor the state of assets in the platform. Monitoring tools can receive real-time data from assets and display them to the user, or they can process the data and let the user know if alarming values occur.

Implement methods for post-processing and analytics Post-processing and analytics increase insights about the health of a physical component or an asset and its structural integrity. Simulations can be used to calculate the remaining useful lifetime.

1.5 Structure of Thesis

The thesis consists of the following chapters:

The **Introduction** sets the context of the project, describes the CBMS project, lists the previous work, and defines the scope of this thesis.

Technical Background introduces definitions and concepts on which the thesis builds upon. Digital twins and digital twin platforms are defined, the terms *cloud* and *monitoring* are presented and explained, and aspects of the *system architecture* and requirements within the field are covered.

Method outlines the methods used during development. Development approaches and evaluation methods are described for the thesis objectives. Finally, the general development process and tools used during development are introduced and explained.

System Overview describes the system after development. Terminology used during development is explained. Requirements are listed and categorized, and the software architecture and the technology stack is described.

Implementation explains specific implementation carried out in the thesis.

Results shows the application's user interface and the results from the evaluation of the objectives.

Discussion evaluates implementation and results of each objective. Finally, other aspects of the CBMS are presented.

Further Work lists interesting directions for further research and other aspects that can be implemented in the future.

Finally, the **Conclusion** summarizes the thesis.

The **Appendices** include more detailed information about results, elaborate code listings, user and installation guides along with instructions on how to deploy the front-end.

Chapter 2

Technical Background

This chapter defines terms and concepts such as digital twins, digital twin platforms, cloud services, and architectural aspects of software development.

2.1 Digital Twins

Various industries are adopting digital twins, from the more traditional manufacturing and automotive businesses to construction, utilities, and healthcare. Sensors transmit data, which reveals the state of components and can be used to perform simulations to obtain more knowledge about the physical asset. This knowledge can be used to optimize the performance of assets, and the digital twins become a bridge between the real and digital worlds. This section defines the concept of a digital twin and proceeds to introduce a digital twin platform and its requirements.

2.1.1 Definition

A digital twin is a virtual copy of a process, product, or service, but there is no single, fully accepted technical definition. This thesis uses the same definition as the previous work by the authors (Kjernlie and Wold 2019), which is the first concept of a digital twin, presented by Micheal Grieves in 2002 (M. Grieves 2016). Grieves' model consists of the real and virtual space that communicate with each other. The real space consists of a physical asset that sends sensor data that reflects its state to the virtual space. Simulations are executed in the virtual space, and the results provide further insight into the state of the physical asset. The insight indicates which actions can be taken in the real space, *e.g.*, maintenance actions. Grieves did not invent the term digital twin; it was John Vickers of NASA who used it to describe Grieves' concept (M. Grieves 2016).

The premise for Grieves' concept was the exchange of information - that the real and virtual space adjusts each other, which is known as a digital twin today. Two other concepts are related to digital twins; *digital models* and *digital shadows*. The extent of autonomy in the interaction between the real and virtual space is what distinguishes them. A digital shadow receives the data and processes it but does not automatically send a response to the physical asset, as the response must be sent manually. A digital model sends information both ways manually. Figure 2.1 shows an illustration of the three concepts.

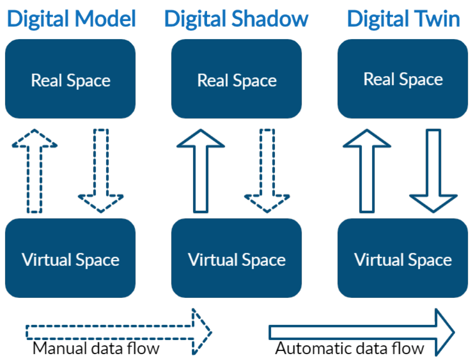


Figure 2.1: A digital model has no automated data exchange between the physical and digital object. A digital shadow sends data from the physical to the digital object and the digital twin sends data in both directions.

2.1.2 Digital Twin Platform

A digital twin platform must facilitate the creation of digital twins by fulfilling the requirements defined by Grieves (2014) Grieves (2014):

1. Physical products in real space
2. Virtual products in virtual space
3. The connections of data and information that ties the virtual and real products together

This creation consists of uploading *models* to the virtual space. The models must contain a representation of the physical asset that can perform simulations. The platform must provide functionality for exchanging information between the physical and virtual products to complete the digital twin. The physical twin must be instrumented with sensors that capture its state, and sensor data must be sent to the platform in real-time. After processing this data, the system can send a response with instructions back to the physical asset. The specific simulations and analyses in the platform is decided by the industry the digital twin is applied in.

2.2 Cloud Computing

In 1961, John McCarthy stated that (sometime in the future), one would pay for computational resources used as a utility made available to the public (Daylami 2015). In the end of the 20^{th} century, *grid computing* was defined as a large-scale resource sharing system providing on-demand computational resources (Foster, Kesselman, and Tuecke 2001). It was developed to solve the problem of facilitating

> direct access to computers, software, data, and other resources required by a range of collaborative problem-solving and resource-brokering strategies emerging in industry, science, and engineering (ibid.)

Cloud computing is essentially providing computing resources, applications, storage, or entire infrastructures as a pay-as-you-go solution. An individual or an enterprise does not need knowledge of the hardware infrastructure (Bhardwaj, L. Jain, and S. Jain 2015), but can subscribe to cloud providers' services.

The cloud has the characteristics of being

> on-demand self-services with broad network access, resources pooling, elasticity in terms of scaling, and pay-per-use (Bhardwaj, L. Jain, and S. Jain 2015).

Scaling refers to allocating or removing resources, and can be performed *vertically* or *horizontally*. Vertical scaling consists of adding resources to an existing server instance, while horizontal scaling adds another instance to the server.

2.2.1 Service Models

One usually divides cloud services into three categories; Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS), and Software-as-a-Service (SaaS). IaaS provides processing, storage, and networking services (Daylami 2015). The provider maintains the system, and the user installs and runs software on top of the infrastructure. PaaS extends IaaS, where developers can build and deploy applications without buying a single piece of hardware (Bhardwaj, L. Jain, and S. Jain 2015). SaaS offers access to or use of software hosted by the provider, integrating with the user's application or infrastructure through configuration and customization.

The services are hosted in a virtual environment, either *virtual machines* (VMs), *containers*, or *serverless* architecture. A VM is an instance that has the characteristics of a traditional computer, including an operating system and a certain amount of available resources. The user can install software and run applications. Containers are more lightweight than VMs as they do not have an operating system. They provide an environment to run applications, which only requires the application itself and files and information needed to run it, such as dependencies, environment variables, and configuration files. Since it virtualizes the operating system, the container is operating system-independent. The serverless architecture lets the user upload a *package* to a server that contains the source code of an application. When a function in the package is called, it is deployed in a container. The customer does not know anything about the environment in which the application runs (Baldini et al. 2017).

2.2.2 Deployment Models

The most common cloud models are *public* and *private* clouds. Public clouds are available to the general public and large organizations. These are typically offered as a pay-per-use fee and scale with the user's demand. Private clouds are operated for a single organization, either by the organization itself or with a third party service. Only granted members can use the private cloud, providing greater control over the infrastructure and computational resources (Goyal 2014). Control over the system is an advantage of a private cloud, facilitating data privacy and security management. A significant drawback of a private cloud is higher costs, as a public cloud enables the *economy of scale*.

Some of the largest public cloud providers are Amazon Web Services, Google Cloud Platform, and Microsoft Azure. These platforms offer services for the development, management, and deployment of applications, including solutions for authentication and database management, big data analytics, and IoT devices. Developers can select the services that fit their needs. The services are often straightforward to integrate, as public cloud providers want to attract as many users as possible. Simple integration makes it possible to develop faster, but it has a tradeoff in terms of costs and flexibility.

A large organization such as NTNU manages a private cloud within the organization, as it has a large technological community and comprehensive resources. NTNU users can create networks, routers, and virtual machines in the private cloud *stackit*. Stackit does not offer serverless services, but it is possible to scale on-demand by upgrading resources.

2.3 Monitoring and Maintenance

Monitoring is the regular observation and recording of a system's quality and progress over time. It is valuable to monitor industrial equipment because it uncovers the state of an asset's structural integrity, and increases the awareness of potentially harmful operating conditions. Insights gained from monitoring provides a better basis for making decisions about the system, *e.g.*, when to perform maintenance to prevent structural failure. Continuous measuring and *real-time* transmission of critical parameters such as strain, temperature, and applied load is facilitated by instrumenting assets with sensors. The requirements for real-time transmission depends on the application; a delayed signal in a car engine control system has an entirely different effect than a delayed signal from a wind turbine in the ocean. Filtering sensors removes unwanted components of the signal and improves the quality of the data. However, a defect sensor can lead to incorrect decisions, and one must carry out frequent tests of equipment.

Historically, maintenance policies have concerned doing maintenance and repairs at equipment failure (reactive maintenance), under suspicion of fault (corrective maintenance), or at systematic time intervals (preventive maintenance) (Barros 2019). These maintenance schemes are suitable for some applications, but often sub-optimal. Corrective maintenance might lead to extra costs due to broken equipment, and the faulty equipment can lead to unexpected halts in operations, which affects other parts of the system. Preventive maintenance is often performed in excess, as it usually over-schedules maintenance in fear of failure.

Condition-based and predictive maintenance strategies try to solve these problems by exploiting sensor data. Condition-based monitoring uses data from sensors or other conditions such as listening to a machine's sound to obtain indications about a system's health. These indications can imply a degradation of the system's performance, and maintainers should be notified. Furthermore, triggers can automatically inform the relevant persons when sensor values are outside specified limits.

Predictive maintenance relies on advanced statistical methods such as *machine learning* (ML) to predict when to perform maintenance on a part. Continuous monitoring is required to obtain sufficient information about specific components' health (Carvalho et al. 2019). Predictive maintenance looks at complex patterns across multiple sensors and makes predictions about the system's future behavior. ML algorithms are often preferred due to their high degree of generalization for classification and regression tasks without substantial domain knowledge. These properties make ML suitable for generic systems such as the CBMS. The key is that models can be trained to detect patterns without using any explicit instructions, making them applicable to problems that are difficult to solve using conventional algorithms. The data must be pre-processed before it trains a model. The knowledge discovery during the training phase is more complicated if the data contains noise, as the quality of the model is highly dependent on the data's quality (Kotsiantis, Kanellopoulos, and Pintelas 2007). Cleaning the data is, therefore, essential to increase the accuracy and efficiency of the model. Training an ML model with high accuracy can be difficult, but once a model is trained, it is fast to predict. Fast prediction is one of the significant advantages of using ML models for real-time applications. Adaption to data-driven ways of predicting faulty components and sensors minimizes maintenance costs and increases the remaining useful lifetime.

2.4 Software Architecture

Architectural decisions lay the groundwork for the development and have implications in terms of constraints. The following sections list concepts that should be evaluated in order to plan the development of an application.

The CBMS uses a client-server architecture. The client provides an interface that allows the user to interact with the server (Hosch 2015). It sends *requests* a dedicated central server that receives, processes, and returns a *response*. The requests can be for resources or services that the server provides, and the network communication goes through the server as the clients can not communicate with each other. Hence, it needs to run on hardware with high processing power and large storage space to manage communication and process requests. The server can provide authentication service, database resources, and run applications. The communication between the client and the server is facilitated through a network or the internet.

2.4.1 Front-end

The front-end is the client in a client-server architecture. It provides the interface that the end-user can use to interact with the server and displays the requests' response. Furthermore, the front-end decides what happens when the user interacts with elements in the interface, such as buttons or input fields. Since the front-end is what the user sees and uses, it is also responsible for the user experience (Granevang 2019b). The user experience is essential for how valuable the application is to the user, which is why they are often designed by interaction and graphic designers.

The front-end is usually developed with JavaScript, HTML, and CSS. A *framework* can be used for more efficient development. The framework provides a foundation for building programs by offering specific functionality in reusable code, such as sorting algorithms that can be used in an application.

Front-end solutions can be web applications that run in browsers or native apps that can be downloaded on a device. The compatibility of the application decides how broad it can reach; if it is developed as an iOS app, it is only available to end-users with Apple devices. Web applications are available to everyone with an internet connection. They are compatible with different JavaScript, CSS, and HTML versions, and thus a web application might not be compatible with all web browsers. One should investigate if any of the elements used in the front-end rule out compatibility with a web browser during development.

2.4.2 Back-end

The back-end is the counterpart of the front-end and the server in the client-server architecture. The end-user does not interact directly with the back-end. The back-end's tasks include heavier calculations, interaction with a database, and communication with the front-end. The back-end is often split into different layers that handle various tasks (Granevang 2019a). If the application processes large amounts of data, the back-end must interact with a database. The back-end is also responsible for the security of the application by regulating access for different user groups (ibid.). As explained in Section 2.2, the back-end can be hosted locally or in a cloud solution, depending on the requirements for the back-end.

2.4.3 Authentication

Authentication is required in all platforms where one can register a user and log into a system. It is the process of verifying an identity and relies on something you *know*, *have* or *are* (Bishop 2005). When a purchase is made using a credit card, authentication is completed using a code that the card owner knows. Something you have can be a device that generates a code, *e.g.* the code chip used by Norwegian banks. Signatures, fingerprints, and retinal scans can authenticate users because they are a part of who you are. Authentication grants access and enables storing information that can be accessed later in a new session.

It is common to implement an authentication system in an application, and many companies offer Authentication-as-a-Service (AaaS). AaaS enables easy implementation of authentication and user management services in applications. Using a third party to authenticate users is preferable because the probability that it is more robust than a self-written system is high.

2.4.4 Storage

Applications need to store data in different ways, ranging from the name of a person to an entire file. Size, format, availability, and duration of storage decide how a database should be implemented.

Web Storage

Web storage makes it possible to save information persistently directly in the browser with JavaScript. It is a feature of the HTML5 specification, and the largest web browsers support it. Data can be stored in either *session storage* or *local storage.* Session storage only saves data for the current session, whereas local storage keeps the data stored after the browser is closed. The data is saved as strings in key-value pairs. If an integer or object is stored, it is automatically converted to strings. Web storage is easy to use and *misuse.* It is not a substitute for a database as it has a limit of 5MB storage space, and there is no access protection. The information is only stored in the browser, and any JavaScript code can access it, which is why sensitive information should never be stored in web storage.

Database

The purpose of a database is to store information based on a specific data model over time. It is possible to read, update, delete, or add new data using a *database management system* (DBMS). There exists a variety of databases that offer different solutions for different needs. The database implementation must be tailored to fit the data model of the application in the best possible way. It is possible to implement multiple databases in an application. This is advantageous for complex applications with a variety of data formats and performance requirements. A drawback of using more than one database is more complicated implementation, but one can gain substantial performance improvements. A real-time database processes transactions for time-sensitive information in real-time. Databases can be hosted locally, or in private or public clouds, with the advantages and drawbacks explained in section 2.2.

Relational databases use schemes to define how the data should be stored in different tables that are related to each other. Within one table, there are strict rules that determine which data can and must be within each row of the table. Structured Query Language (SQL) is used for data management. Relational database management systems provide atomicity, consistency, isolation, and durability (ACID) guarantees. They can be vertically scaled, but might not offer ACID characteristics if they are horizontally scaled. It is easy to restrict access to data for different user groups. Relational databases are optimized for structured data. The format of the data should be defined before the implementation. Examples of relational databases include MySql, MS-SQL and Oracle.

Non-relational (or NoSQL) databases support unstructured data by providing flexible schemas that do not have to be defined before implementation. They provide more scalability, and horizontal scaling is more straightforward than in relational databases. However, NoSQL databases do not support ACID transactions in general. Well-known NoSQL databases include MongoDB, Hbase, and Redis.

2.4.5 Network Protocols

Network protocols define rules and standards for communication over a network and facilitate communication between two devices (Shimonski, Cross, and Hunter 2005). Both the sender and receiver need to set up the same network protocol to be able to communicate. There are many different protocols, and the one being used depends on the communication requirements. Some connections require high throughput, and others rely on assurance that every message is sent and received.

Transmission Control Protocol (TCP) splits messages into small packets, which can be lost or arrive in an incorrect order. If a device sends more than one message at a time, it is important to identify which packets belong to which message. A three-way *handshake* initializes the connection between two devices. Then, messages start flowing back and forth. Each time a computer receives a packet, a confirmation message is sent back to the sender to confirm that it is received. If the sender does not receive a packet confirmation, that packet is resent. This means that the connection is reliable: all messages are successfully transmitted. However, reliable connections impact bandwidth as all messages require a confirmation (Zimmermann, Eddy, and Eggert 2016).

User Datagram Protocol (UDP) is similar to TCP, but no handshake is required to initiate the communication. One specifies an IP address and start sending messages. The messages are split into packets in the same manner as TCP, but confirmation messages are not sent back to the sender. Hence, UDP communication is not reliable and must tolerate a certain loss, but can reach a higher throughput than TCP (*User Datagram Protocol* 1980).

Hyper Text Transfer Protocol (HTTP) is designed to transfer information between networked devices, and transfers HTML documents over the web (Belshe et al. 2015). A client sends a request and receives a response from the server. The response contains a *code*, which indicates if the request is successful (200), if there is a client-side (400) or server-side (500) error. If the request is successful, additional information is sent with the response. HTTP is built upon the TCP protocol.

Hyper Text Transfer Protocol Secure (HTTPS) is an extension of the HTTP protocol, which is used for secure communication between the client and the server. HTTPS uses an encryption protocol to encrypt messages to protect data during transmissions.

WebSocket is also built on top of TCP, and facilitates two-way communication between a client and a server. Real-time communication is achieved by allowing the server to send data to the client without being requested by the client first, and keeping the connection open, reducing the overhead of HTTP (Pimentel and Nickerson 2012).

2.4.6 Functional Mock-Up Interface and Unit

A *functional mock-up interface* (FMI) is an open standard interface that allows the exchange of dynamical simulation models between different tools (Blockwitz et al. 2012). The FMI standard provides import and export between various tools while keeping the same model. According to Blockwitz et al. (ibid.), it has become heavily adopted in the industry since it was released in 2010. A model following the FMI standard is called a *functional mock-up unit* (FMU). The FMU contains a simulation model which is acquiescent with FMI specification. The FMU extension is a zip file containing an XML file defining input and output for the model. It also consists of binary files and DLL[1] files that contain the equations used in the model.

There are two types of FMUs, namely co-simulation and model exchange of dynamic models. Both kinds of FMUs can be used for both co-simulation and model exchange and can perform simulations of the system over time. The difference lies in the execution of the models. Dynamical models consist of differential equations and have inputs that provide values to these equations. Outputs are specified by the result of the differential equations and their input. In a co-simulation FMU, the solver, determining how to solve the differential equations, lies in the FMU. The importing tool only sets the input and tells the FMU to do a time step. In a model exchange FMU, however, the importing tool have the solver, and the FMU provides the equations. The solver in the importing tool computes the state and decide when to do a time step.

[1]A DLL file is a library that contains code and data for executing a particular task in Windows.

2.5 Usability

Usability is concerned with how easy it is for a user to accomplish a desired task. Making a product usable is one way of improving a user's perception about a system's quality (Bass, Clements, and Kazman 2015). However, the task of making a usable product is not an easy task, or as Albert Einstein said; "any darn fool can make something complex; it takes a genius to make something simple" (ibid.). Usability derives from the term user friendly (Alonso-Ríos et al. 2009), which was critized of having undesirably vague and subjective connotations (Bevan 1995b). The term usability is therefore used to replace the term user-friendly to overcome its limitations (Bevan, Kirakowski, and Maissel 1991).

2.5.1 Achieving Usability

Developing a user-friendly application involves making it as easy as possible to learn the features of the system, make the user use them efficiently, minimize the impact of errors, adapt the system to the needs of the user and increase the satisfaction and confidence of the user while he or she is using the system (Bass, Clements, and Kazman 2015).

Learning the Features of the System. The system should help users unfamiliar with the system use new and existing features. Learning new software requires time and effort, and the system should help the user with the process.

Using a System Efficiently. The system should help users to be as efficient as possible in the process of achieving their goals. The system should be intuitive and avoid time-consuming operations to make a seamless experience. Users should be able to manage the system's resources without having to think about the underlying logic and functionality.

Minimizing the Impact of Errors. Errors should be displayed in a simple and interpretable format to minimize the impact of the error.

Adapting the System to the User Needs. The system should adapt to users to make it easier to use. The system should automate tasks to make it easier for users to achieve their goals.

Increasing the Confidence and Satisfaction. The system should give feedback to the user when a correct or incorrect action occurs to raise the user's confidence and satisfaction. The design of the system should reflect the purpose of the service to deliver a better user experience. Every web appliation has its personality, even if the owners of the applications are not aware of it (Wathan and Schoger 2018). A web application raises different impressions, whether it is professional, playful, formal, or technological. The choice of colors, fonts, and hierarchy of elements affects these impressions and should be taken into consideration when creating a design.

2.5.2 Conducting and Measuring Usability Tests

Usability testing is conducted to see how usable or intuitive an application is by observing how real users use a service by encountering problems and experience confusion. It can be used in many ways during the lifecycle of a project, both during the beginning of the project, during development and after the project is finished. Usability testing is a useful tool to see if an application achieves its goals, but it cannot completely mimic real-life usage.

There are many ways of conducting a usability test. A common usability technique is *concurrent think aloud* (CTA) (Haak, Jong, and Schellens 2003), which is used to understand what users think while interacting with an application. The user is narrating their thoughts throughout the whole session. This technique is helpful to understand what kind of problems the user encounters while using the service, and provides instant feedback and emotional responses of the user.

Another technique is *retrospective think aloud* (RTA) (ibid.), where the users retrace their steps after they have completed the session. This technique is used if it is preferable to not interfere with the user, for example if it is interesting to see how much time the user spends on a certain action. The overall length of the session is increased, and it can be difficult for the user to remember all the steps. *Retrospective probing* (Birns et al. 2002) is similar to RTA and is based on an interview after the session about the tester's thoughts and actions.

A technique that makes it possible to hear what the user thinks and lets the person conducting the test interact with the user is called *concurrent probing* (CP) (Aiyegbusi 2019). The user narrates while testing as in CTA, but the researcher can interrupt with questions if the user does or says something interesting. CP brings the users' thoughts to the surface while working through a task. However, it interferes with their thoughts, making them execute actions later that might be different if they are not interfered with.

Data can be collected both during and after usability tests with questionnaires to measure the usability and quality of a web application. The questionnaire can be standard or homegrown tailored to the specific situation. A standardized questionnaire provides a more reliable and valid measure compared to homegrown questionnaires (Sauro 2015). The drop in reliability is most likely due to poor questionnaire design (Hornbæk and Law 2007). A standardized questionnaire can also be used to measure improvements in the future. A drawback of standardized questionnaires is that the questions are typically wide and open and it can be difficult to isolate particular issues. These issues can be detected by using one of the techniques described above to observe how the user uses the system and encounters problems.

Chapter 3

Method

The first five sections of this chapter describe the methods used to implement and assess the objectives of the thesis. Section 3.6 describes the workflow and tools used during development.

3.1 User-Friendly Client

A usable web application has to be developed based on usability principles. Usability tests are executed during and after development to measure if the implementation fulfills a set of requirements.

3.1.1 Architectural Design Checklist

A design checklist provides guidance to achieve the qualities of a user-friendly web application described in Section 2.5. The checklist used in this thesis is defined in Bass, Clements, and Kazman (2015), and is a categorization of seven design decisions developers can focus on to develop a usable application. *Allocation of responsibilities* describes how the system allocates responsibilities to assist the user, and the *coordination model* explains how the system's elements coordinate for a better user experience. *Data model* describes the data abstractions involved in making the web application perceivable for the user, and *mapping among architectural elements* describes how architectural elements are visible to the user. *Resource management* says how the user manages available resources, and the *binding time* says something about when the user can perform specific actions.

Lastly, *choice of technology* explains how technologies are chosen to create the best user experience. Table 3.1 explains the checklist item more thoroughly.

3.1.2 Usability Testing

Usability tests are useful to reveal flaws in the platform's user experience design during and after the development process. For the CBMS, the CTA technique is used during the development phase. It is simple to conduct and provides fast results as the user expresses his or her thoughts during the session. The feedback from CTA tests are applied directly in the implementation. A mix of CP and RTA is used to test the CBMS at the end of the project to get a more detailed impression of the usability of the platform. The users are told to narrate their thoughts and interact with the person conducting the test while doing the following actions:

- Register an account and log in
- Create a project, upload a model and configure a data source
- Create a dashboard and visualize real-time data
- Experiment with the functionality available in the platform
- Log out

At the end of the session, the users fill out a questionnaire. A modified version of the Standardized User Experience Percentile Rank Questionnaire (Sauro 2015) is used to validate the platform's usability. The questionnaire is a comprehensive measure of the quality of the user experience (ibid.). The original questions described in Appendix D are adjusted to adapt to the use case of the CBMS. The questionnaire presents statements related to the usability of the platform, and are answered on a scale from 1 to 5 according to the level of agreement with the statement. The first statement evaluates the relevant background knowledge related to the purpose of the platform, and the rest are related to the usability of the platform. The questionnaire contains the following statements:

- I am familiar with the field of digital twins.
- The platform is easy to use.
- It is easy to navigate within the platform.

Category	**Description**
Allocation of responsibilities	Ensure system responsibilities are located to assist the user in learning how to use the system, efficiently achieve the task at hand, adapt and configure the system and recover from user and system errors.
Coordination model	Ensure system elements coordinates to make sure the user learns the systems, achieves its goals, adapts and configures the system, recover from errors, and gains increased confidence and satisfaction.
Data model	Determine abstractions of data and operations to make it easier for the user to learn the system, achieve the tasks at hand, adapt and configure the system, recover from errors and increase satisfaction and confidence.
Mapping among Architectural Elements	Determine the mapping among architectural elements visible to the end-user, for example, the extent to which the user is aware of local and remote services to make sure the user learns the system, achieve tasks, adapt and configure the system, recover from errors and increase satisfaction and confidence.
Resource Management	Determine how the user can adapt and configure the system's use of resources. Ensure that the level of resources does not adversely affect the users' ability to learn how to use the system or decrease confidence and satisfaction.
Binding Time	Determine which and when decisions should be under user control. Binding time is the latest time during the process the user has to decide to make a decision. These decisions should not affect the user's ability to learn the system, use it efficiently, minimize errors, adapt and configure the system, and increase confidence and satisfaction.
Choice of technology	Ensure technologies help to achieve the system's usability without adversely affecting users' ability to learn the system, use it efficiently, minimize errors, and adapt and configure the system and increase confidence and satisfaction.

Table 3.1: The design checklist for usability defined by Bass, Clements, and Kazman (2015).

- The platform is fast and responsive.
- It is easy to get an overview of available features.
- I will likely return to the platform in the future.
- I find the platform attractive.
- The platform has a clean and simple presentation

A minimum of 25 participants with knowledge of digital twins should test the platform to obtain a sufficient dataset to draw conclusions regarding the platform's usability. The responses are scored by averaging the results from the questions related to the usability of the platform. A survey executed in Sauro (2015) with 2,513 responses across 70 websites with the standard SUPR-Q questionnaire shows an average score of 3.93. This is not directly comparable to the questionnaire used to measure the usability of the CBMS as it is customized for its purpose. It is still used as a basis to determine the score for considering the usability objective to be fulfilled. The average usability of the web application is set as a benchmark for the user-friendliness of the CBMS. The platform is therefore considered to be user-friendly if the average score from the questionnaire is higher than 3.93. The question related to the user's knowledge about the field of digital twins is not associated with the platform's usability, so this question is not included in these calculations.

3.2 Personal and Persisting Projects

The objective of facilitating support for persistent projects is approached by implementing an authentication service and a database that contains data about users and projects. Evaluation of the objective achievement is carried out in a test where two users simultaneously execute the following tasks:

1. Register an account and log in
2. Create a new project
3. Connect to a data source
4. Upload a model
5. Create one or more dashboards
6. Create one or more tiles

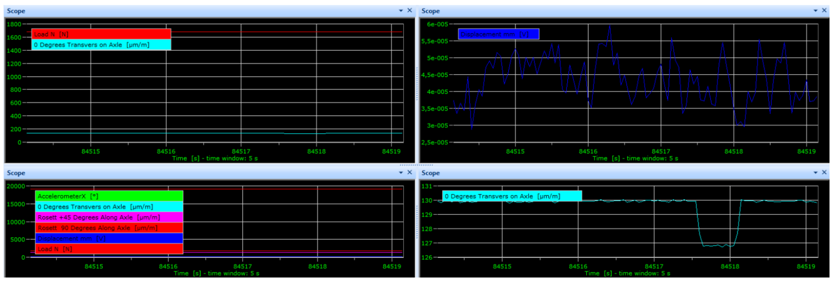
When these steps are completed, the testers log out. The next day, they log on again on *different* devices. If they see the same content they created the previous day, the test is considered successful, and the objective is completed.

3.3 Generic Configuration System

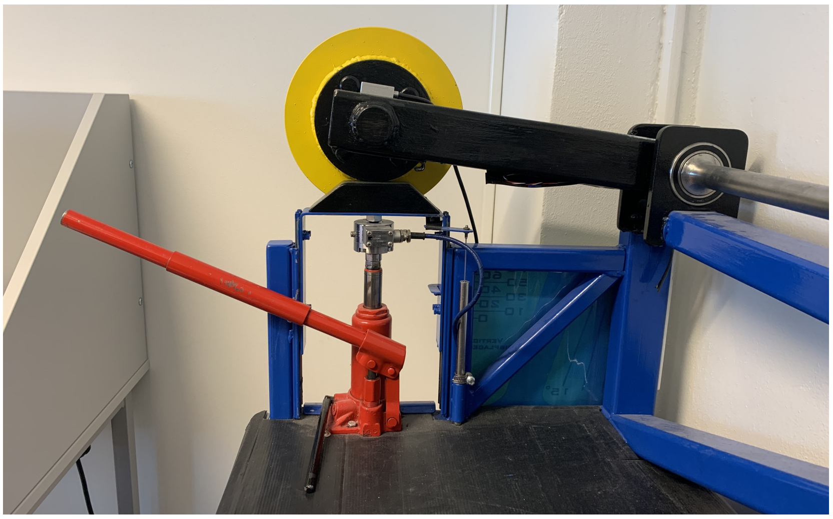
During development, two physical assets are used to implement functionality for digital twin configuration; a torsion bar suspension rig and a smartphone.

NTNU has a torsion bar suspension rig used for experimental purposes. The rig is instrumented with sensors to capture its state. The sensors include a load cell, accelerometer, strain gauge, and rosettes. Sensors send data to a data acquisition system (DAQ) before it is sent to the server, as seen in Figure 3.1a. The DAQ software is installed on a computer connected to the rig and configures and transmits the data. Figure 3.1b shows a picture of the rig. The red handle is used to jack up the torsion bar, and the black handle attached to the jack lowers it.

SensorLog is an application developed by Bernd Thomas, published on Apple Store. It displays real-time sensor data from the hardware of an Apple device, *e.g.,* longitude and latitude coordinates, accelerometer, and a gyroscope, as seen in Figure 3.2a. The app provides three main options for the use of the data. It can be displayed on the device, logged to file or streamed. Figure 3.2b shows the configuration settings of the application. Streaming frequency ranges from 1 to


Scope
Load N [N]
0 Degrees Transvers on Axle [μm/m]
Displacement mm [V]
AccelerometerX [°]
Rosett +45 Degrees Along Axle [μm/m]
Rosett 90 Degrees Along Axle [μm/m]
Time [s] - time window: 5 s

(a) Real-time plots from the DAQ software used to measure the physical characteristics of the torsion bar suspesion rig

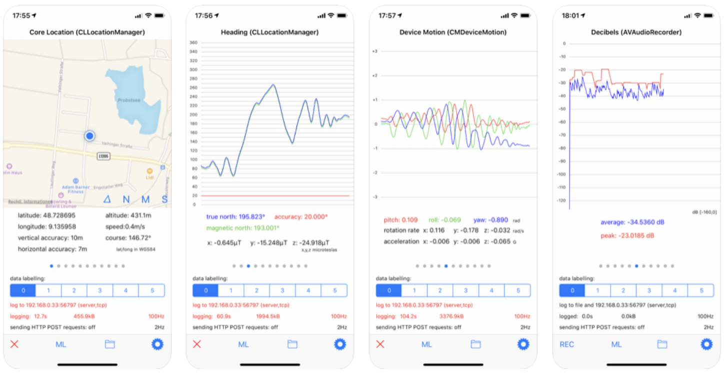


(b) NTNU's torsion bar suspension rig used during development of the CBMS

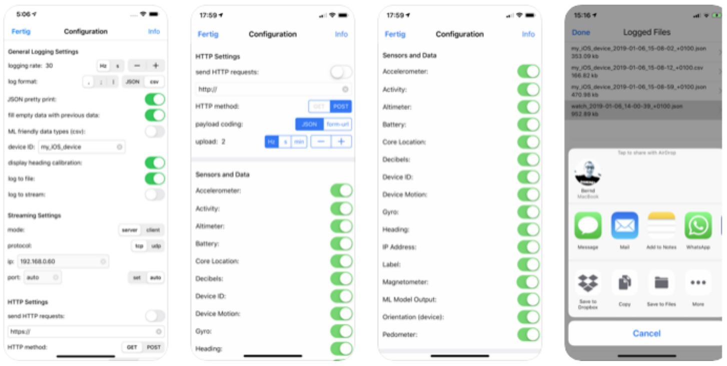
Figure 3.1: Screenshot of the DAQ software and a picture of the torsion bar suspension rig.

100 hertz. The data can be logged and streamed in a comma-separated values (CSV) or JavaScript object notation (JSON) format. The data is transmitted with the TCP or UDP protocol.

A new physical asset is connected to the system after the development phase to evaluate how flexible and generic the system is. The model is uploaded, and a data source is configured. The new asset is monitored, and analyses and post-processing are executed. The results indicate how successful the implementation is. The objective is considered fulfilled if the monitoring and analytics features produce results, *i.e.* the real-time data is displayed, and one can query recent data to generate analyses.


17:55
Core Location (CLLocationManager)
latitude: 48.728695
altitude: 431.1m
longitude: 9.135958
speed:0.4m/s
vertical accuracy: 10m
course: 146.72°
horizontal accuracy: 7m
lat/long in WGS84
17:56
Heading (CLLocationManager)
true north: 195.823°
accuracy: 20.000°
magnetic north: 193.001°
x: -0.645µT
y: -15.248µT
z: -24.918µT
x,y,z microteslas
17:57
Device Motion (CMDeviceMotion)
pitch: 0.109
roll: -0.069
yaw: -0.890 rad
rotation rate x: 0.116 y: -0.178 z: -0.032 rad/s
acceleration x: -0.006 y: -0.006 z: -0.065 G
18:01
Decibels (AVAudioRecorder)
dB (-160,0)
average: -34.5360 dB
peak: -23.0185 dB
data labelling:
0
1
2
3
4
5
log to 192.168.0.33:56797 (server,tcp)
logging: 12.7s 455.9kB 100Hz
logging: 60.9s 1994.5kB 100Hz
logging: 104.2s 3376.9kB 100Hz
log to file and 192.168.0.33:56797 (server,tcp)
logged: 0.0s 0.0kB 100Hz
sending HTTP POST requests: off 2Hz
REC
ML

(a) Visualization of sensor data from a smart phone in the SensorLog application


5:06

Fertig Configuration Info

General Logging Settings

logging rate: 30 [✓ Hz] s

log format: , ; | JSON [✓ csv]

JSON pretty print: on

fill empty data with previous data: on

ML friendly data types (csv): off

device ID: my_iOS_device

display heading calibration: on

log to file: on

log to stream: off

Streaming Settings

mode: [✓ server] client

protocol: [✓ tcp] udp

ip: 192.168.0.60

port: auto set [✓ auto]

HTTP Settings

send HTTP requests: off

https://

HTTP method: GET POST

17:59

Fertig Configuration Info

HTTP Settings

send HTTP requests: off

http://

HTTP method: GET [✓ POST]

payload coding: [✓ JSON] form-url

upload: 2 [✓ Hz] s min

Sensors and Data

Accelerometer: on

Activity: on

Altimeter: on

Battery: on

Core Location: on

Decibels: on

Device ID: on

Device Motion: on

Gyro: on

Heading: on

17:59

Fertig Configuration Info

Sensors and Data

Accelerometer: on

Activity: on

Altimeter: on

Battery: on

Core Location: on

Decibels: on

Device ID: on

Device Motion: on

Gyro: on

Heading: on

IP Address: on

Label: on

Magnetometer: on

ML Model Output: on

Orientation (device): on

Pedometer: on

15:16

Done Logged Files

my_iOS_device_2019-01-06_15-08-02_+0100.json
353.09 kb

my_iOS_device_2019-01-06_15-08-12_+0100.csv
166.82 kb

my_iOS_device_2019-01-06_15-08-59_+0100.json
470.98 kb

watch_2019-01-06_14-00-39_+0100.json
952.89 kb

(b) Options for configuring a smart phone to collect and send data in the SensorLog application

Figure 3.2: The SensorLog Application.

3.4 Monitoring

This objective's goal is to implement monitoring features that provide insights and knowledge about the current state of the physical asset. The elements must present the data reasonably and intuitively. The level of success is evaluated by measuring the average delay from the data is sent from the physical asset until it is received and displayed in the front-end. It is crucial to synchronize the clocks in the physical assets, the back-end, and the front-end before the test is executed to obtain accurate results. The objective is considered fulfilled if the results of the assessment comply with the requirements outlined in Section 4.2 related to the monitoring objective.

The CBMS needs to be running at all times to monitor physical assets and carry out tasks continuously. These properties can be evaluated by measuring the availability and the performance of the system. The availability is measured by how often the system responds to a request, while the performance is measured by the time it takes to respond to a request. Checkly is a relability platform used to measure availability and performance. Checkly automatically checks that the back-end is up and running by making an HTTP request to one of the API endpoints every 5 minutes. Checkly also makes a browser check every 10 minutes to verify that the client is running. The browser check starts up a Chrome browser and loads the web page. When the page finishes the loading, the check is complete. The requirement for the availability and performance of the platform is outlined in Section 4.2.

3.5 Post-Processing and Analytics

Evaluation of the implemented analytics functionality achieved by comparing the outcome to trusted, external data. SAP provides the data required to test and validate the features. The validation files are received after implementation to prevent customized implementation, and Table 3.2 describes the validation set.

The data files are in a CSV format, where one column contains the variables separated by commas. The raw data and spectrogram files are provided in two formats, `raw_data_iso.csv` has a UNIX timestamp in microseconds and `raw_data_epoch.csv` has a *date*[1] format.

The sample spacing and sample frequency need to be generated to generate the transformations. The total number of data points is found, and the duration

[1]The date format is on the form yyyy-mm-ddThh:mm:ss.msZ

of the sample is calculated from the timestamps. Table 3.3 lists the required variables.

The platform generates an FFT using the `raw_data_epoch.csv` file and the sample spacing from Table 3.3. Data from the generated FFT is downloaded. The result is evaluated by visual inspection and comparison of the values. The values in the `fft.csv` file generated by SAP are plotted using the historical plot functionality in the platform for visual inspection. The downloaded values from the platform are compared to the values in the `fft.csv` files. A spectrogram is generated from the `raw_data_epoch.csv` file and the sample spacing value from Table 3.3.

Filename	**Content**
fft.csv	FFT values from the signal: frequencies of the signal along with intensity
raw_data_epoch.csv	Raw file that consists of two columns: timestamps formatted as UNIX timestamps in microseconds and corresponding sensor values of an accelerometer in a compressor
raw_data_iso.csv	Raw data file that consists of two columns: timestamps formatted as Dates and corresponding sensor values of an accelerometer in a compressor
spectrogram_epoch.csv	CSV file containing three columns: frequencies of the signal, timestamps on a UNIX format in microseconds and acceleration values.
spectrogram_iso.csv	CSV file containing three columns: frequencies of the signal, timestamps formatted as dates and acceleration values
spectrogram.png	PNG file showing the spectrogram with values from the spectrogram data files.

Table 3.2: Files provided by SAP. The raw data files are used to generate FFT and spectrogram, `fft.csv` and the spectrogram files contain FFT and spectrogram values generated by SAP and `spectrogram.png` is SAP's spectrogram.

Variable	**Value**	**Calculation**
N	2560	Number of samples
Duration	9.9961	Difference between first and last data point
Frequency	256.1001	N / duration
Sample spacing	0.0039	duration / N

Table 3.3: Calculation variables for FFTs and spectrograms. *N* and *duration* are the number of samples and the duration of the time series in seconds respectively. Sample spacing is the inverse of frequency, the time between to samples. Frequency is in hertz and sample spacing in seconds

3.6 Development Process

The development is carried out in *sprints*, following an *agile* approach. *Kanban* boards organize and keep track of the progress of the sprints. *Figma* and *Adobe Photoshop* enables layout design of the application *Git* and *Github* facilitate version control and collaboration.

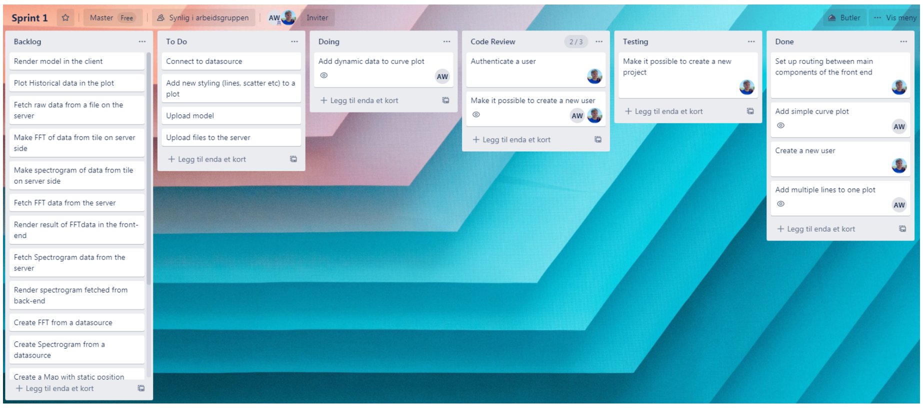
3.6.1 Agile Approach

The project development follows an agile methodology by developing and refining tasks in iterations called sprints. In an agile development approach, one starts with the fundamental features and adds complexity incrementally. The most substantial advantages of an agile approach are less time spent on detailed planning, which frees up resources in a project with time constraints, and continuous review of the progress. The agile approach relies heavily on communication between the members of the team. As the group only comprises two members working closely together at all times, it is considered a small risk.

Kanban

During the development phase, Kanban boards in Trello[2] are used to track the progress. Kanban boards consist of columns representing possible states of a task, in this case, "to do," "doing," "code review," "testing," and "done." Before initiating each sprint, a selection of tasks from the *backlog* is put in the "to-do" column. The backlog initially consists of the high level functional requirements in Table 4.1 split into small tasks. However, more tasks are added after sprint-reviews and when new needs arise. Sprint items that are not completed during one sprint are transferred to the next or removed if they are no longer relevant.

[2]Trello is a free, web-based Kanban project management application.


Sprint 1 | Master Free | Synlig i arbeidsgruppen | AW | Inviter | Butler | Vis meny

Backlog
- Render model in the client
- Plot Historical data in the plot
- Fetch raw data from a file on the server
- Make FFT of data from tile on server side
- Make spectrogram of data from tile on server side
- Fetch FFT data from the server
- Render result of FFTdata in the front-end
- Fetch Spectrogram data from the server
- Render spectrogram fetched from back-end
- Create FFT from a datasource
- Create Spectrogram from a datasource
- Create a Map with static position
- + Legg til enda et kort

To Do
- Connect to datasource
- Add new styling (lines, scatter etc) to a plot
- Upload model
- Upload files to the server
- + Legg til enda et kort

Doing
- Add dynamic data to curve plot (AW)
- + Legg til enda et kort

Code Review 2 / 3
- Authenticate a user
- Make it possible to create a new user (AW)
- + Legg til enda et kort

Testing
- Make it possible to create a new project
- + Legg til enda et kort

Done
- Set up routing between main components of the front end
- Add simple curve plot (AW)
- Create a new user
- Add multiple lines to one plot (AW)
- + Legg til enda et kort

Figure 3.3: Snapshot of a Kanban board in Trello from the first sprint.

During the sprint, the developers take items from the "to do" list and move them to the "doing" column while the development is in progress. When the task is completed, it is moved to the "code review," and finally the item is tested. Then, if there are no bugs, it moves on to the "done" column. Different developers can contribute to the process of implementing the same task, e.g., one developer solves the task and another tests and approves. Tasks are assigned to developers, as seen in Figure 3.3.

Design

The design process is based on the principles and tactics from Wathan and Schoger (2018). The design and implementation are developed in cycles. Must-have features are designed first, then implemented. New features are designed and implemented in a new iteration after the first iteration is complete. This process is repeated until there are no more features to implement. A cyclic design process is used to save time and focus on the most critical elements first.

The layout of the front-end solution is outlined using Figma and Adobe Photoshop. Figma offers a free, collaborative design interface where it is possible to design interactive components that can be connected through user actions such as clicking a button. Figma is used to create prototypes quickly, as shown in Figure 3.4. Adobe Photoshop is used to create and edit images used in the web application, such as the logo for the CBMS.

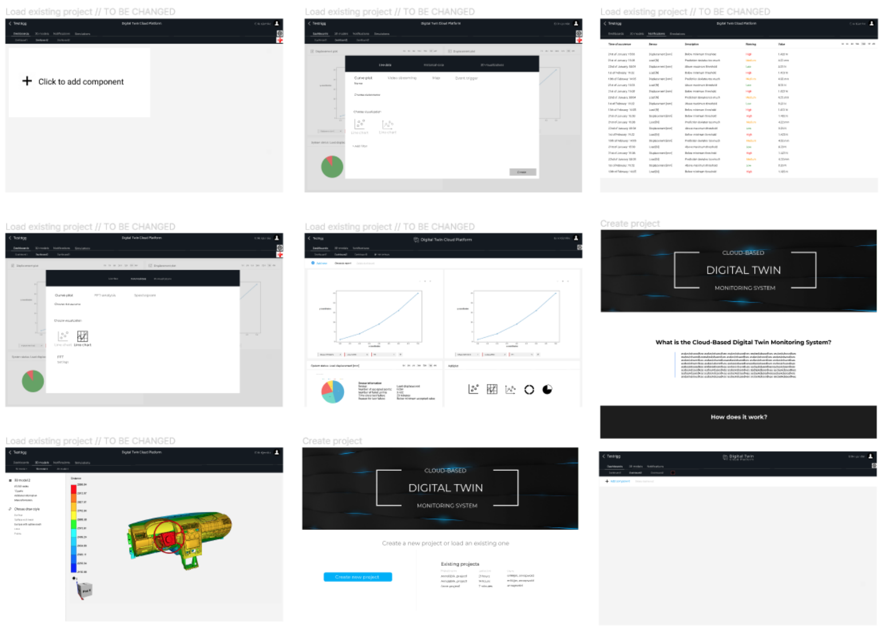

Load existing project // TO BE CHANGED
Click to add component
Create project
CLOUD-BASED
DIGITAL TWIN
MONITORING SYSTEM
What is the Cloud-Based Digital Twin Monitoring System?
How does it work?
Create a new project or load an existing one
Create new project
Existing projects

Figure 3.4: Design sketches from Figma.

3.6.2 Code Control and Collaboration

Code is written in *Visual Studio Code*, a source code editor created by Microsoft. Changes in the code are shared using Git. Git is a version control system for keeping track of changes, additions, and deletions of files, making it possible to revert the project or a part of the project to a previous state. Centralized synchronization of source code is essential in development projects. GitHub, a code collaboration platform, is used for this purpose. Source code for the project is uploaded to GitHub in a *repository*, and users can clone the repository to their local computer to implement changes. When a new task is initiated, a new *branch* is created. Upon completion, the branch with the updates is pushed to GitHub, and a *pull request* is created. A pull request is a request to merge the updated changes with another branch, usually the master, the main branch of the repository. A team member can look through the changes made and approve or leave comments before the branch is merged.

It is important to use identical code formatting to minimize the number of merge conflicts. Two Visual Studio extension packages are used for this purpose: *Prettier* and *vsc-organize-imports*. Prettier helps formatting code by adjusting the code to comply with defined rules when files are saved, e.g. length of lines, use of parentheses, commas, etc. vsc-organize-imports automatically sorts the imports in a file in alphabetical order for consistency.

3.6.3 File Sharing and Communication

Google Drive is used to organize the project and the administrative files. Summaries from status meetings, meetings with supervisors, and other external resources are located in different folders to makes it possible to go back and review old notes and information at a later stage. It simplifies administration and improves control of progress. Credentials, passwords and NDA-restricted files that cannot be uploaded to GitHub are stored in the drive. When communication cannot be conducted face to face, Microsoft Teams is used to conduct meetings.

Chapter 4

System Overview

The previous work described in Section 1.3 is the foundation for the system's requirements, architectural decisions and technology stack, which are described in this chapter. The first section defines terms used during development of the system.

4.1 Terminology

This section introduces the terms and definitions used in the development phase. The majority of the terms do not appear in the client as they can be confusing for the end-user. The terms are defined for the reader to understand the system better, and Figure 4.1 shows an illustration of the system with the terms integrated.

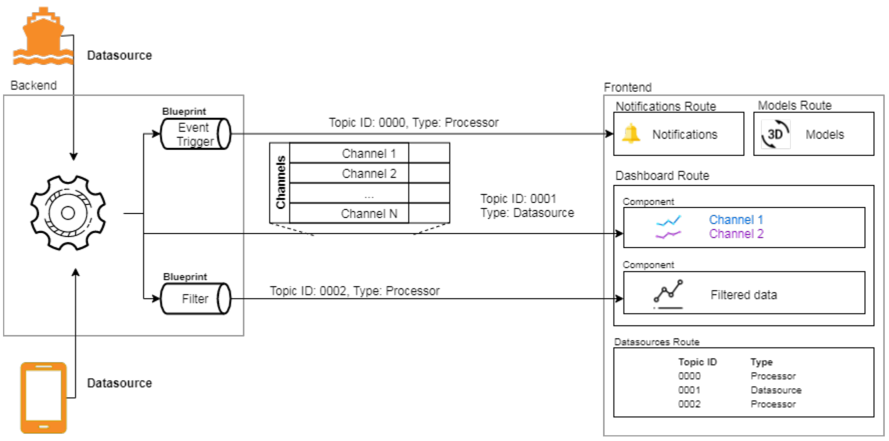
Datasources are the real-time streams of sensor data created by the users in a project. The values of the datasources originate from physical assets and are routed through the back-end before they arrive in the front-end. The back-end receives the data while the physical assets send data, but the user must define a datasource to connect the client to the sensor data.

Processors transform streams of sensor data, such as filtering. The user sees these as datasources to abstract logic, but there is a technical difference as the data is no longer the same as the raw data sent from the physical asset.

Blueprints are defined in the back-end and relate to processors. Jensen (2019) created a blueprint system to make a generic way to add processors. When the functionality in the back-end is extended by adding new functionality, a new processor that follows the blueprint system is created. This includes initiation parameters and functions such as `init`, `start` and `stop` for initiating, starting and stopping processors, respectively.

Topic The data streams are transmitted from the server to the client via Apache Kafka. To keep control over the traffic, Kafka makes a *topic* for each datasource. When the data arrives in the front-end, it is managed using topics. As with processors, the term topic is not visible; the user only interacts with datasources.

Channels of a topic or datasource are the sensors that the topic or datasource contains. The channels contain a name of the sensor and an id. Whenever the user creates a *tile* in a dashboard, the data comes from one or more channels.



Topic ID	Type
0000	Processor
0001	Datasource
0002	Processor

Figure 4.1: Physical assets are *datasources* sending sensor data to the back-end. The data is forwarded to the front-end in a *topic* as raw data or processed in a *processor*. A topic consists of channels containing sensor values. The user navigates between *routes* in the front-end and the data is visualized with *components*.

Dashboard is the main component of a project, and is initially a blank canvas. The users configure the dashboards themselves by creating **tiles** with monitoring and analysis functionality.

Notifications appear when a sensor value exceeds a selected threshold. The notification contains information about the event, *e.g.*, date, duration, and description, and the user can view a plot of the sensor value at the time of the event.

Models are the virtual models of the physical asset. Models are visualized in 3D and can be used to run simulations.

Components are the building blocks of React applications. Components receive input parameters and return a React element describing how a section of the user interface appears.

Routes keep the user interface in synchronization with the URL in the browser. A route's component is rendered when the route matches the URL.

4.2 Requirements

The system should be a cloud-based open-source platform for digital twin applications. The user should be able to configure projects by uploading models and connecting to streams of sensor data. Both the models and data streams should be visualized in the client of the platform. It is desirable to execute certain post-processing such as FFT or spectrogram. The client should be available in a web browser to be device-independent. The technology stack should be chosen so that it is easy to continue the development at a later stage, and thorough documentation should be generated.

The requirements are divided into two categories: the functional and non-functional requirements. The functional requirements outline what the system should do. In contrast, non-functional requirements include aspects of the system that do not have an explicit connection to the functionality, rather the quality of the product. Tables 4.1, 4.2 and 4.3 list the functional and non-functional requirements. These are numbered and given a rating in terms of priority and difficulty ranging from low to high. The requirements are related to the objectives of the thesis.

ID	**Priority**	**Difficulty**	**Obj.**	**Requirement**
FR01a	High	Medium	3	The user should be able to connect to streams of sensor data by providing an IP address.
FR01b	High	Medium	3	The platform should accept data streams in JSON and CSV formats.
FR02	High	Medium	3	The user should be able to generate a 3D model by uploading an FMU file.
FR03	High	Medium	4	The system should visualize real-time data streams from physical assets.
FR04	High	High	4	The system should visualize and update 3D models in real-time.
FR05	High	Medium	5	The user should be able to visualize historical data.

Table 4.1: List of functional requirements of the platform.

ID	Priority	Difficulty	Obj.	Requirement
FR06	High	High	5	The user should be able to post-process data, e.g. by computing FFTs.
FR07	High	Low	5	The user should be able to upload files in CSV or XLSX format.
FR08	High	Low	1	The user should get an error message if something goes wrong and be told how they can fix the problem.
FR09	Medium	High	5	The system should provide predictive maintenance functionality to avoid equipment failure.
FR10	Medium	Medium	3	The user should be able to apply filters to data sources.
FR11	High	Low	2	The user should be able to register a unique account using their e-mail.
FR12	High	Low	2	The user should be able to configure projects by adding models and connect to datasources. It should be possible to make personal dashboards that are saved for later sessions.
FR13	Low	Low	2	It should be possible to share projects among multiple users by sending invitations.

Table 4.2: List of functional requirements of the platform (continued).

ID	Priority	Difficulty	Obj.	Requirement
NFR01	High	Low	1	The platform should be available in Google Chrome, Microsoft Edge, Mozilla Firefox and Safari
NFR02	High	Low	1	The front-end should be developed using React
NFR03	Medium	High	1	The platform should be intuitive and easy to interact with.
NFR04	Medium	Low	1	Detailed documentation should be generated for the system in the form of README files, instruction videos and user guides.
NFR05	High	High	4	The real-time data should have a latency of at most two seconds.
NFR06	Low	High	4	The platform should have an uptime of at least 99.95%.

Table 4.3: List of non-functional requirements of the platform.

4.3 Architecture

This section describes the components of the complete system, its internal and external connectors, and the technology stack used for development. The architecture is a product of the requirements, the specialization project, and the state of the project before development.

Throughout this thesis, the terms *front-end* and *client* refer to the React application that constitutes the interaction with the user. *Back-end* and *server* both refer to the Python application running on the server. All use of *database* refers to the *Firestore* instance implemented in this thesis. *Firebase* refers to Google's mobile platform, providing both Firestore and the Authentication service used in the project. The CBMS is referred to as *the platform* and *the system.*

4.3.1 System Architecture

The platform uses a server-client architecture, and Figure 4.2 shows the top-level structure. The front-end uses an authentication service, listens to notifications from the database, and communicates with the server. The server also communicates with the database, receives data from physical assets, and reads and writes files in the local file system.

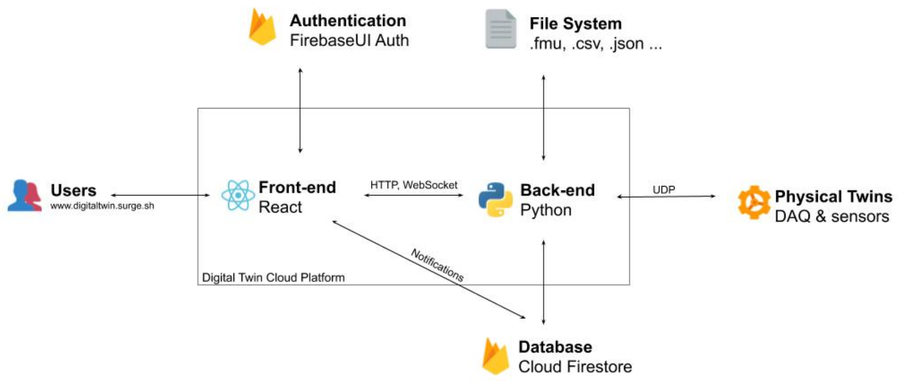


Figure 4.2: The system consists of a front-end developed in React, a Python back-end, an authentication service, a database and connections to a local file system and physical assets.

The front-end is a single-page application divided into components, or routes, as Figure 4.3 illustrates. The landing page route renders when the application first loads. The user can log in or register, or go directly to the projects Route if already authenticated. From the projects route, one can navigate to an existing project or configure a new project. In the project route, there are four subroutes: dashboards, notifications, models, and data sources. The admin user can navigate to the admin route to see administrative settings for the platform. Only the developers of the CBMS can make users admins.

The back-end has the structure illustrated in Figure 4.4. Similarly to the front-end, it is divided into separate modules based on the functionality they provide. Each model has two layers: the *views* and *models* layer. The views layer manages client communication by receiving requests and sending responses. The logic and calculations that are executed between these actions are taken care of by the models layer.

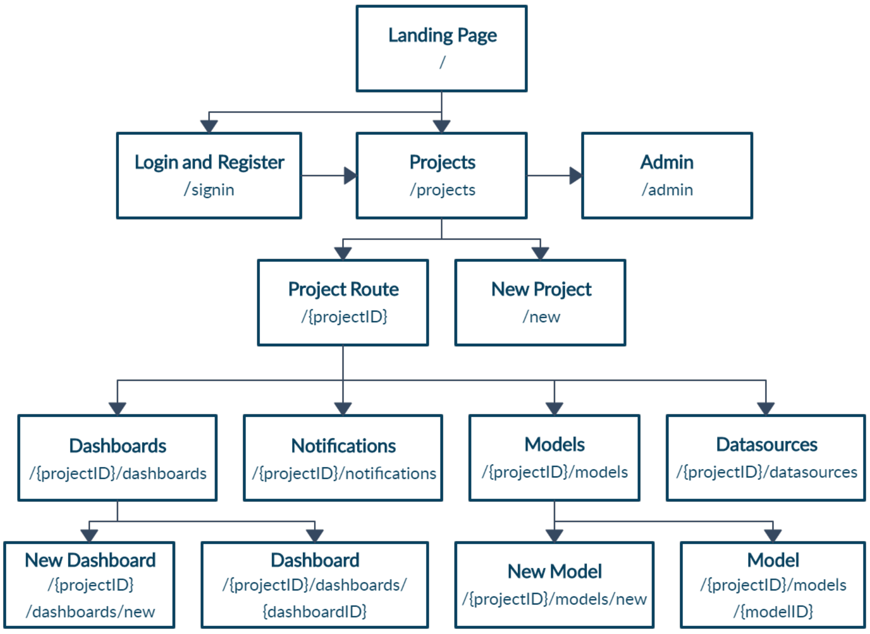


Figure 4.3: Routing diagram of the front-end. The text below the name of each route represents the URL in the browser, *e.g.*, `/signin` represents `www.cbms.surge.sh/signin`.

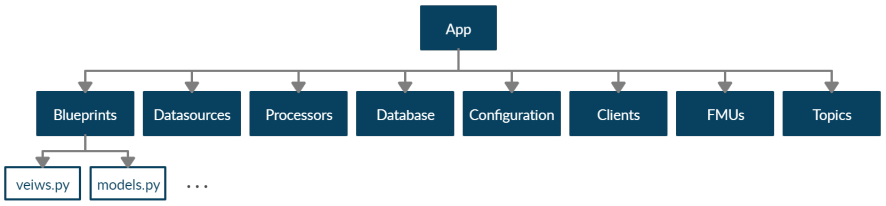


Figure 4.4: The back-end is divided into modules, such as *Database* or *Topics*. Each module contains a file for communication with the front-end, views.py, and some of the modules has a file for performing calculations, models.py.

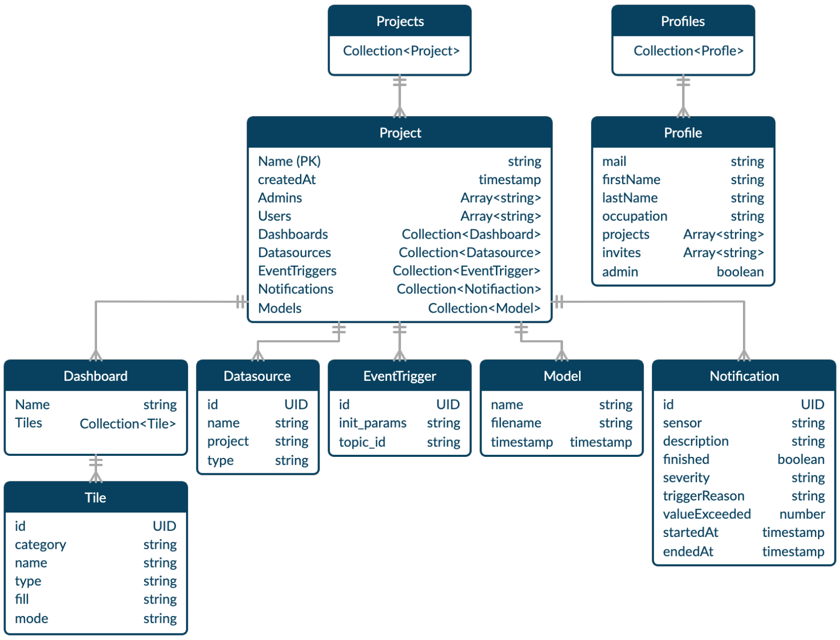


Figure 4.5: The database structure in Firestore consists of collections and documents. A project contains information about dashboards, datasources, event triggers, models, notifications and the users of the project. A user can have many projects and a project can be shared with other users.

The database is a real-time noSQL database that follows a concept of collections and documents. Each collection contains one or more document, and documents can contain sub-collections. The platform has two root collections, *profiles* and *projects*. The profiles collection consists of profile documents, which contain the user information. The projects collection has a more complex composition, which Figure 4.5 illustrates.

The file system structure follows the structure of the back-end, where the folders correspond to a particular module. Files are written to and read from, and used in the various components of the system. The blueprint folder contains the blueprint instances as sub-folders. When a processor is initiated, the `init`-file in the correct blueprint folder is run, and a processor is created. This processor is then saved as a file in the processor folder. Appendix A shows an illustration of the file system.

4.3.2 Communication

Different network protocols are chosen according to the need of the interaction between components. Data streams require high throughput, while communication with the database does not require real-time response.

The client and server communicate through WebSocket connections and HTTP requests as shown in Figure **??**. The sensor data is sent through the WebSocket connection, as high throughput is essential for the streams. Kafka distributes the data. All interactions that do not require real-time transmission use HTTP requests. This comprises querying data from the server, uploading files, fetching transformations, and handling database requests. The majority of the requests to Firebase from the client go through the server. There are two types of requests directly between Firebase and the front-end: the authentication service provided by Firebase and the real-time listeners in Firestore. When users are authenticated, the client sends a request directly to Firebase. The event-listeners in the front-end receive updates directly from Firestore in a WebSocket connection. The back-end listens for UDP connections from physical assets and forwards data to clients. Kafka simplifies the communication by sending messages directly from different processes.

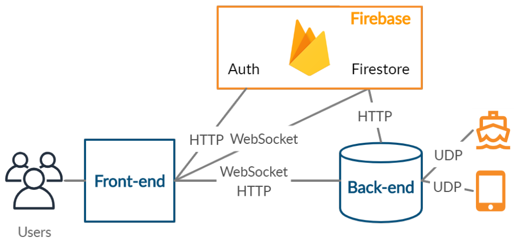


Figure 4.6: Overview of the communication protocols between the elements in the CBMS.

4.3.3 Cloud-based

All the components are hosted in the cloud. The client is hosted by Surge, which is a free resource provided through node package manager (npm). Surge publishes web applications for free and the process of deploying to Surge is presented in Appendix G. Google's Firebase is used for both authentication and database. The back-end is hosted with stackit as a VM with the specifications shown in Table 4.4. The server is globally available.

Specification	**Value**
Random Access Memory (RAM)	16GB
Virtual Central Processing Units (VCPUs)	4
Disk	60GB

Table 4.4: Specifications of server instance in terms of processing and storage capacity

4.4 Technology stack

This section lists and describes the selected technologies used during the development of the platform.

4.4.1 Front-end

During the research conducted in Kjernlie and Wold (2019), it was decided that React and TypeScript should be used to develop the new front-end solution due to its flexibility, performance, and prior knowledge within the team.

React is a JavaScript library, but it can be used with TypeScript to implement type safety. TypeScript is a superset of JavaScript, which compiles to JavaScript. TypeScript makes it easier to read and debug by catching type mistakes. Types reduce the need for code documentation and build more robust software. JavaScript files are still accepted for external Javascript imports into the project, *e.g.* for implementing 3D visualizations with Ceetron.

4.4.2 Back-end

The back-end solution was created in Jensen (2019), and it is decided to keep and further develop this solution in Kjernlie and Wold (2019). It is implemented in Python using a modular style that easily enables extension of functionality. The back-end is described in "Building an Extensible Prototype for a Cloud-Based Digital Twin Platform" by Jensen (2019).

4.4.3 Communication

JavaScript provides functionality for both HTTP and WebSocket communication; hence no further technologies are needed to facilitate communication in the front-end. However, Python does not have built-in feature for external communication, but several libraries provide it. Aiohttp was selected in Jensen (ibid.) to support HTTP and WebSocket communication, and Kafka through aiokafka. Apache Kafka is used as a message distributor from the processes in the back-end to the front-end.

4.4.4 Authentication

Many companies provide AaaS, but they offer different features, functionality, and pricing plans. Originally, Feide was selected to be the Authentication provider (Kjernlie and Wold 2019). However, the project would not be accessible to anyone other than students and professors in Norway. *Auth0* and *Firebase Authentication* are considered, and the latter is selected due to the integration with the database and lower costs. It provides a simple and well-documented API for features such as login, logout, and password resetting. Additional features such as phone authentication can be implemented, but are not a part of the free solution.

4.4.5 Storage

The platform handles three sorts of data: user-generated data, data associated with the physical state of the asset in the form of incoming real-time sensor data, and files uploaded by users. Due to different formats and use cases, three different storage units are used. The two latter units are implemented in previous work (Jensen 2019).

Two different databases are considered for user-generated data handling in the platform: a local MySQL database and Google's cloud database Firestore. MySQL is a free, open-source database, and it can be run on a local server or in the cloud. It is an SQL relational DBMS, and the data is stored in tables with row and columns. MySQL is one of the world's most popular databases[1] with a large community, known for being simple to install and using few resources in terms of computer memory and CPU.

Firestore is a cloud-based, real-time document store structured in collections and documents instead of tables. Each collection contains documents, which can be compared to a row in an SQL table. Firestore supports non-relational data, making it more flexible than traditional relational databases as documents in the same collection do not need to contain the same fields and sub-collections. This makes it easy to change the format of the data during development. Google's Firestore is selected due to flexibility, fast prototyping, and cloud-related advantages. Firestore has a free tier, including up to 50.000 and 20.000 read and write operations a day.

Files and models are stored in the file system on the server. Sensor data from physical assets is saved in *Kafka logs* folders on the server. Apache Kafka is mainly an open-source event-streaming system, but can also be used to store data. The data is stored as streams of events persisted to disk, checksummed, and replicated for fault tolerance. It does not work as a regular database with a query language to store and retrieve data. However, it can operate similarly to a database and provide ACID guarantees. Kafka does not replace an ordinary database, but this is neither its purpose.

[1]https://db-engines.com/en/ranking, accessed: 06.06.2020

4.4.6 Visualization of Virtual Models

Code from *Ceetron* is used to visualize the physical assets. Integration with Ceetron prohibits the CBMS from being completely open-source as the scripts are NDA-restricted and cannot be shared publicly. Code snippets related to the visualization and processing of virtual models are therefore not included in this thesis. The implementation builds on work in Børhaug and Sande (2019).

Chapter 5

Implementation

This chapter describes the implementation carried out in this thesis. The majority of the development is focused on the front-end since it is built from scratch, but the back-end is extended to support the implemented functionality. The chapter is divided into five sections, one for each thesis objective.

The development is executed in a top-down manner. The first step is to initiate and define the structure and data flow in the front-end and its communication with external parts of the system. Then, functionality is implemented in both the front-end and the back-end, including a generic configuration of profiles, projects, datasources and models, monitoring, and analysis features. Finally, after the critical functionality is completed, more details and additions are added following the agile approach described in Section 3.6. The external libraries used for implementation in this thesis are listed and briefly described in Appendix B.1.

5.1 User-friendly Client

Defining the application structure is the first task of creating a web application. The application structure includes determining the main modules of the front-end, establish how to navigate and pass data between them, and select how to communicate with external parts of the platform.

A React app consists of *components* that have a set of properties called *state*. State is an object containing the properties of a component, and when the properties change, the component is re-rendered. Components are nested in a hierarchy, and one can pass properties, called *props*, from the *parent* to the *child* component. The child component is a part of the parent component's scope, and props is the way to pass data down to the child. *Routing* facilitate navigation between the top-layer components. Data is shared globally between components in *stores* or in a uni-directional flow between components using props. The structure and data flow is illustrated in Figure 5.1.

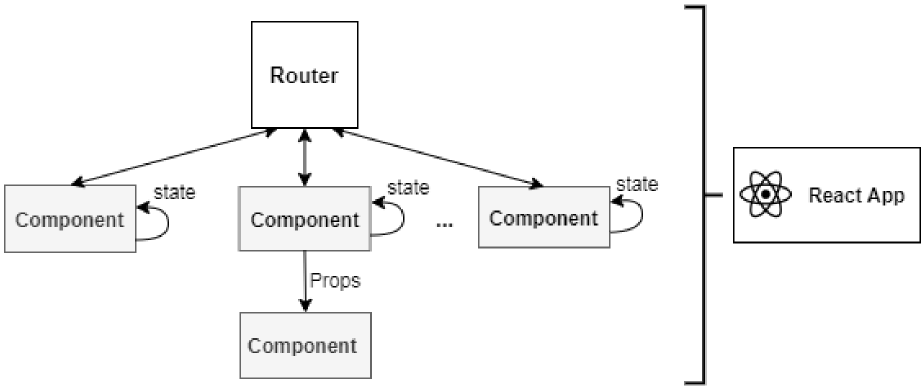


Figure 5.1: React's structure consists of nested components that pass props among them and keep a consistent internal state. Routing facilitates navigation between the routes of the application

5.1.1 Creating a React App from Template

The first step of the development phase is to create a React app. The application is initialized by using React's standard template[1] with TypeScript. TypeScript requires that each variable must be declared. *Interfaces* defines the variables of an object and their respective types. Different objects are defined as interfaces in the `types` folder and are divided into four files depending on use. Listing 5.1 shows the interface for the type `Profile`.

[1]https://create-react-app.dev/docs/getting-started/

```
export interface Profile {
  projects: string[];
  occupation: string;
  firstName: string;
  lastName: string;
  invites: string[];
}
```

Code lising 5.1: Interface that defines the type Profile, which consists of the variables projects, occupation, firstName, lastName and invites

5.1.2 Components

The React application is built with *functional* components. Functional components are JavaScript functions that take some parameters as inputs and return a React element as output. Functional components are more readable, reusable, require less code, and are easier to test compared to *class* components, which is another common standard in React. Classes provide *lifecycle* methods, which often generate larger amounts of code than necessary. Hooks reduce the number of concepts exposed to developers, which makes it easier to learn. Writing everything as functions simplifies reusing the code, instead of switching between functions and classes. Listing 5.2 shows a functional component called `MapComponent`. It takes in coordinates and returns another component, `Map`, which renders a map with a marker on the given position. `Marker` is the child component of `Map`, which is the parent component.

```
const MapComponent = ((long: string, lat: string) => {

  return (
    <Map>
      <Marker
        position={[long, lat]}
        icon={myIcon}
      >
    </Map>
  );
});
export default MapTile;
```

Code lising 5.2: A simple component that takes in longitude and latitude and renders a map using a `Map component`

Generic components such as buttons and input fields are created to ensure a consistent and coherent layout. The component is defined with attributes in a TypeScript file as in Listing 5.3. `Button` is here used as an example, but the process is identical for other components. Different buttons can be used by specifying the *className* attribute, which changes the design of the button.

```
import * as S from "./Button.style.ts";
export default function Button(props: Props) {
  /* ... code ... */
  return (
    <S.Button
      className={"Accept"}
      onClick={onClick}
      /* more button attributes */
    >
      {loading ? <ClipLoader size={15} color={"blue"}
      loading={loading} /> : children
      }
    </S.Button>
  );
}
```

Code lising 5.3: Custom button component. The `loading` property is set to `true` if the button should display a loading icon. `children` refers to the content inside the Button component, for example the text "Please click me".

Styling of components is implemented with the *styled-components* library. The design of the button is defined in a specific file called `Button.style.ts`. This file includes custom classes, as shown in Listing 5.4, where green and red buttons are used to continue or cancel an action, respectively. The design of the button is imported in the Button component in Listing 5.3, where it can be accessed by using `S.Button`. If one wants to use the design of the accept button, the *className* property is set to "Accept."

```
import styled from "styled-components";

export const Button = styled.button`
/* Default styling and other classes*/
 &.Accept {
    background-color: green;
    /* more attributes such as "font-size", "color" etc. */
 }
 &.Decline {
    background-color: red;
    /* more attributes */
 }
  /* More button classes */
`;
```

Code lising 5.4: Excerpt of styling file for the Button component showing the `Accept` and `Decline` classes for the `Button` component.

5.1.3 Routing

Routing manages the flow between the main components of an application that can be navigated to through an url. *React-router-dom* facilitates routing between the components by declaring where to direct the user when different routes are specified in the address bar, *e. g. www.cbms.surge.sh/signin* renders the `SignInSignUp` component as shown in Listing 5.5. It is implemented using the `Route`, `Router` and `Switch` components of react-router-dom.

```
<Router>
  <Switch>
      <Route exact path="/" render={() => <LandingPage />} />
      <Route exact path="/signin" render={() => <SignInSignUp />}
          />
      <Route exact path="/new" render={(props) => <NewProject />}
          />
      /* More routes */
  </Switch>
</Router>
```

Code lising 5.5: Router component renders `LandingPage`, `SignInSignUp` or `NewProject` depending on the url. *exact* specifies that the url must match exactly the path in the route.

Internally, react navigates between routes using the `useHistory` Hook, also provided by react-router-dom. `useHistory` has a variable called `history`, which can be used to manage routes, *e.g.* by sending the user to the new project page by calling `history.push("/new")`.

5.1.4 Data flow

React *Hooks* provides a simple way to manage the state within a functional component. Hooks are functions that make it possible to *hook into* a component for managing local state and lifecycle methods. State management is achieved by using the `useState` hook. The `useState` hook contains one variable for accessing the current state value and one for updating it. The hook only has one argument; the initial state. Listing 5.6 shows an example when the `useState` hook is used to control the e-mail value of an input-field when a user registers. E-mail is initially set to an empty string, and is updated with the `setEmail` method when a user types something in the input field.

```
const [email, setEmail] = useState("");
```

Code lising 5.6: Initiating an `email` variable as an empty string using useState

The `useState` cannot share data with other components. A complex React app requires a way to handle *global* state across components. The global state is managed using the Zustand library, a solution based on Hooks. *Stores* are defined in the `store` folder in the repository, which is divided into different categories, *e. g.* `profileStore` as shown in Figure 5.2.

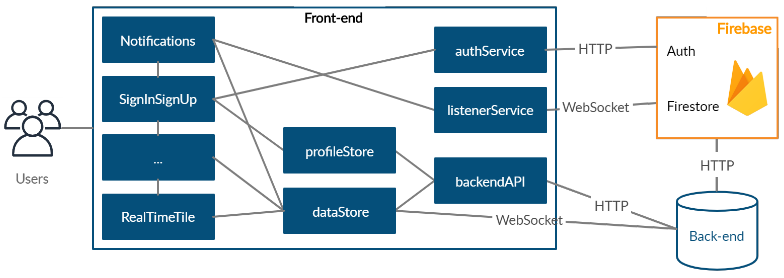


Figure 5.2: The dataflow inside the React application.

The stores contain variables and functions to manipulate the state of the variables. Listing 5.7 shows the `profileStore` used to store information about a user. A component that wants to access data from a store imports the store and declares the specific content it needs, as shown in Listing B.1 in Appendix B.2. When the state of the data is updated in the store, it is also updated in the components that use it.

```
import create from "zustand";

const [useProfileStore] = create((set, get) => ({
  profile: {} as Profile,
  fetchingProfile: false,
  /* More variables, such as profileError etc. */
  createProfile: async (profile: Profile) => {
   /* Code for creating profile */
  },
  /* More methods: getProfile etc. */
}));
```

Code lising 5.7: `profileStore` manages the `profile` variable and provides the `createProfile` function to create a new profile

Methods for fetching resources from the back-end are located in the `backendAPI` folder. Methods are split into different files depending on the API endpoint they call. For instance, the methods used for creating, deleting, and retrieving information about a user are in the `profile.ts` file, which contains the `profileStore`. Data can also be retrieved through *services*, which communicate directly with Firebase, as shown in Figure 5.2. Stores import functions from `backendAPI`, and share the data with components in the application.

5.1.5 External Communication

The front-end receives data from the back-end with HTTP requests and WebSockets.

HTTP requests

Resources are fetched from the back-end using HTTP requests. JavaScript provides a built-in asynchronous method called `fetch` to make a request and fetch a resource. The method takes one mandatory input argument, which is the path to the API endpoint. It returns a promise that resolves to the response, and the data can then be parsed directly in JavaScript as JSON, as shown in Listing 5.8.

```
export async function getJSONResponse(link: string) {
  const response = await fetch(link);
  jsonResponse = await response.json();
  return jsonResponse;
}
```

Code lising 5.8: Simplified getJSONResponse function that parses a HTTP response to JSON format

WebSockets

The process of receiving, parsing, and displaying data is similar to the approach used in Børhaug and Sande (2019). Information about the datasources is stored in a globally accessible dictionary called `datasources`. The dictionary contains the name, available channels, and the byte format for each datasource. A WebSocket connection opens when a user subscribes to one or more channels of a datasource, and is kept open as long as the user is in the same project. Data from processors is also sent through the WebSocket connection.

The process of handling the incoming data is shown in Listing B.3 in Appendix B.2. Data streams from the back-end are received through WebSocket connections in the `dataStore` in the front-end. WebSockets have an `onMessage` property, which is an *event handler* called when a message is received from the back-end. WebSockets transfer raw sequences of bytes, which is why the data is parsed into strings when received in the front-end. The first four bytes of a data stream are decoded with JavaScript's built-in *TextDecoder* to get the topic id of the data stream, which is used to retrieve the format of the data stream from the `datasources` object. The rest of the data stream is parsed with *structjs*; a library used to parse binary data to JavaScript values by iteratively unpacking data according to a given format. The timestamp for each data point is stored in a buffer called `timestamp_buffer`, whereas sensor values are stored in the `value_buffer`. The data from a single datasource is stored in one buffer called `datasourceBuffer`, whereas the buffers from all the datasources are stored in the `datasourcesBuffer`. The data is stored in a buffer to avoid updates every time new data arrives. The most recent data is sent to the `dataStore` every 100 milliseconds, and the buffer is reset, as shown in Listing 5.9.

```
setInterval(() => {
      set({
        newData: cloneDeep(get().datasourcesBuffer),
      });
      get().resetBuffers();
}, 100)
```

Code lising 5.9: `setInterval` pushes data to the global store every 100 milliseconds. `set` updates the store, whereas `get` retrieves the sourceBuffers object. `cloneDeep` is used to create a new, independent object.

Real-time components listen to the `dataStore` for updates, as shown in Listing 5.10. `newData` is updated every time parsing of new data is completed in the `dataStore`. When newData changes, the component calls `requestAnimationFrame` with `updatePlot` as a callback function. React's built in Hook `useMemo` only re-runs the function if the data has changed. `requestAnimationFrame` updates the plot before re-painting the object passed to the function to ensure a smoother update animation of the plot.

```
useMemo(() => {
    requestAnimationFrame(() => updatePlot());
}, [newData]);
```

Code lising 5.10: The useMemo hook listens to changes in the newData object

5.2 Personal and Persisting Projects

Personal and persisting projects are obtained by implementing a database and an authentication service.

5.2.1 Authentication

Firebase's authentication service is implemented as a service layer in the front-end, and a simplified version can be seen in Listing 5.11. When someone fills in the required information and wants to register or login, either the `createUserWithEmailAndPassword` or the `signInWithEmailAndPassword` method is called on the *authentication* object. The service layer receives the input arguments, passes them to the authentication provider, and returns a response with the result of the action.

```
export const createUserWithEmailAndPassword = async (
  email: string,
  password: string
) => {
  return authentication.createUserWithEmailAndPassword(email,
      password);
};
export const signInWithEmailAndPassword = async (
  email: string,
  password: string
) => {
  return authentication.signInWithEmailAndPassword(email, password)
      ;
};
```

Code lising 5.11: Methods that communicate with the authentication service and their respective input parameters.

5.2.2 Storage

Firestore communication is implemented in the front-end and the back-end. The database is implemented in the front-end solution to receive real-time notifications from actions, such as event triggers and project invites. Listing 5.12 illustrates a simplified method for listening to event triggers for a project, and shows that listening to changes in a collection is implemented with the `onSnapshot` method. Every time a document in the `notifications` collection is changed, the front-end receives the updated information. The `orderBy` method is used to order the notifications by date.

```
listenToNotifications: async (projectId: string) => {
    firestore.collection("projects").doc(projectId).collection("
        notifications").orderBy("startedAt", "asc").onSnapshot(
        snapshot => {
        snapshot.docChanges().forEach(change => {
          updateNotification(change);
        });
    });
}
```

Code lising 5.12: The `listenToNotifications` function is an event-listener to the `notifications` collections with the `onSnapshot` method.

Firestore is implemented in the `src/database` folder in the back-end using the same structure as the rest of the back-end components. The API endpoints for the client are placed in `views.py`, and the code for interactions with Firestore is placed in `models.py`. The latter is the only of the two files that contain code related to Firestore. When the front-end requests information about a user, it sends a request to the back-end, which is received by the method in Listing 5.13. Listing 5.14 shows the process for retrieving requested information from Firestore. Listing 5.13 and 5.14 shows that the API for retrieving data from Firestore is almost identical in Python and React.

```
@routes.get('/profile/{id}', name='get_profile')
async def get_profile(request: web.Request):
    email = request.match_info['id']
    profile = await database.get_user_profile(email)
    if profile is None:
        raise web.HTTPNotFound()
    return web.json_response(profile, dumps=dumps)
```

Code lising 5.13: The back-end receives a request from the client to retrieve a user profile and responds with the information fetched from Firestore.

```python
async def get_user_profile(email):
    doc_ref = db.collection('profiles').document(email)
    try:
        doc = doc_ref.get()
        return doc.to_dict()
    except:
        return None
```

Code lising 5.14: The back-end fetches the *document* for the user profile of the given *email*. This information is stored in the *profiles* collection in Firestore.

5.3 Generic Configuration System

The following sections summarize the implementation of components and logic that expand the configurability of the platform.

5.3.1 Creating a project

The user can configure a new project on the projects page. All the information the user needs to create a new project is the name of the project. The user can have a maximum of ten projects, as Firestore can only query information for a maximum of 10 documents in the same operation. The `createProject` call in Listing 5.15 sends a request to the server, which makes a new project in the database. It only contains information about the user and the selected project name. The server receives the request and forwards it to Firestore.

```typescript
export async function createProject(
  email: string,
  projectName: string
) {
  let formData = new FormData();
  formData.append("email", email);
  formData.append("date", new Date().toString());
  formData.append("projectName", projectName);
  return fetch(rootAPI + "/projects/new", {
    method: "POST",
    body: formData,
  });
}
```

Code lising 5.15: `createProject` sends a request to the back-end to create a new project. `rootAPI` it the IP-address of the back-end.

The process of creating a project is visualized in Figure 5.3. The request is received in the back-end in its API endpoint. The back-end creates a new project in the database if it does not already exist. The process of retrieving a request and sending a response is shown in Listing B.10 in Appendix B.2. Later on, users can add new information such as datasources, models, dashboards, event triggers, and tiles by sending similar requests to the back-end from the files in the `backendAPI` folder. The requests contain information about the different components that can be linked to the project, *i.e.*, the name of a datasource, or model. The file containing the information about the actual datasource or model is stored in the back-end, as only the reference to the file is stored in Firestore.

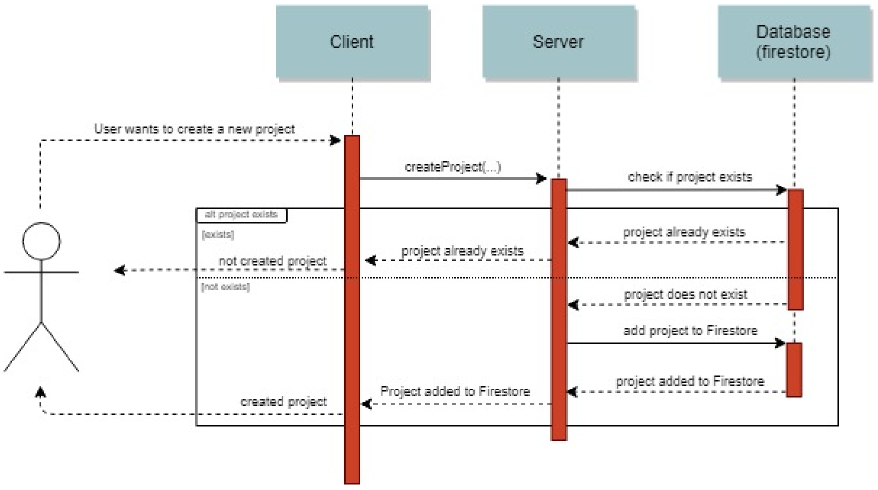


Figure 5.3: The user triggers a request in the client. Firestore checks if the project already exists. If so, no project is created. Otherwise, the user receives a confirmation that the project is created.

5.3.2 Datasources

The process of handling datagrams from UDP connected devices in CSV format is implemented in Jensen (2019). The back-end solution now accepts incoming datasources with JSON or CSV format, which can be configured by the user during the setup process. If the incoming data is in JSON format, it is handled differently than in Jensen (ibid.). An abstracted process of buffering JSON data is shown in Listing 5.16, and a complete version is shown in Listing B.8

in Appendix B.2. The size of messages sent in JSON format are usually larger than CSV as they include each sensor's name. This increases the probability of data fragmentation for the JSON data. The data is therefore stored in a buffer until the complete JSON object arrives. This is verified by checking if the JSON object has the same amount of left and right curly brackets, as the curly brackets defines the JSON object. The data is then packed into a binary array according to the datasource's byte format and distributed to clients when all the fragments have arrived.

```
if address in self.buffers:
    # If device is added to buffers: append more data here
else:
    self.buffers[address] = "" + raw_data.decode()
if len(self.buffers[address]) > 0 and
    self.buffers[address].count("{") == self.buffers[address].count
        ("}"):
    try:
        # code that converts JSON data into an array of values
        data = struct.pack(
            source.byte_format, incoming_data[source.time_index],
            *[incoming_data[ref] for ref in source.output_refs])
        # The data is then distributed to listening clients
```

Code lising 5.16: Excerpt of a function that buffers JSON data. The number of curly brackets is counted and sent when the number of left and right curly brackets is equal.

A function is implemented in the back-end to assist users if they do not know the IP-address of the device that sends JSON data. This is achieved by checking if a datasource is configured when new data arrives. If it is not yet configured, the IP-address and its sensor names are stored in a dictionary in the back-end, as seen in Listing B.9 in Appendix B.2. If a user wants to see the available datasources, a request is sent to the endpoint in Listing 5.17, which returns the dictionary with IP-addresses.

```
@routes.get('/datasources/available', name='
    get_available_datasources')
async def get_available_datasources(request: web.Request):
    datasources = request.app['datasources'].get_available_sources
        ()
    return web.json_response(datasources, dumps=dumps)
```

Code lising 5.17: The API endpoint in the back-end to view available datasource

The dictionary for available sensors contains the name of the sensors for each datasource. The sensor names are retrieved from the server by making an HTTP request from the client to the endpoint described above. The sensors are rendered as drag and drop items using the *react-beautiful-dnd* library. Sensors are rendered as `Draggable` components inside a `Droppable` component as described in Listing 5.18. The user can drag the desired sensors into a separate `Droppable` area for the selected sensors. The name of the channels are not included if a datasource is sending CSV data. The user, therefore, has to fill in the sensors' names in the correct order to get a correct output.

```
<Droppable>
  /* more code for handling the droppable component here */
  {sensors.map((sensor: string, index: i) => (
        <Draggable key={sensor} draggableId={sensor} index={i}>
          /* Displaying the sensor here*/
        </Draggable>
      ))}
  )}
</Droppable>
```

Code lising 5.18: Drag and drop functionality is implemented using `Droppable` and `Draggable` components.

5.3.3 Models and Generation of 3D Files

Models are configured in the NewProject route or in the models tab in the project route. The model is first uploaded in FMU format to the server using the `FileUpload` component, and then processed to generate JSON files used for 3D Visualization by Ceetron's Cloud Components. Processing the FMU is implemented as a parallel process with the *multiprocessing* package to do the process as a separate task. This process is shown in Figure 5.4. A message appears when the model is fully processed so that the user can view the 3D model. The code for processing the FMU and visualizing the model with Ceetron is not shown, as it is only available if an NDA is signed. The 3D visualization can be shown under the models tab, but it is also possible to add a model as a separate tile in a dashboard.

5.3.4 Tiles

Tiles are configured in the front-end in the `AddNew` component. The information provided by the user in terms of input data and configuration variables is stored in the database when a new tile is created. When the dashboard renders, the tile information is extracted and used to render the components. Listing 5.19 shows the POST request made in the client when a user creates a new tile. The implementation of the different tiles is described in Section 5.4 and 5.5.

```
export async function createTile(project: string, dashboard: string
    , tile: TileFormat) {
  return fetch(rootAPI + "/projects/" + project + "/dashboards/" +
      dashboard + "/tiles/new",
    {
      method: "POST",
      headers: {
        Accept: "application/json",
        "Content-Type": "application/json"
      },
      body: JSON.stringify(tile)
    }
  );
}
```

Code lising 5.19: Creating new tile in the client. `TileFormat` is the format of a tile, specified with an interface. `rootAPI` is the IP address of the back-end. The data is sent in JSON format with a POST request.

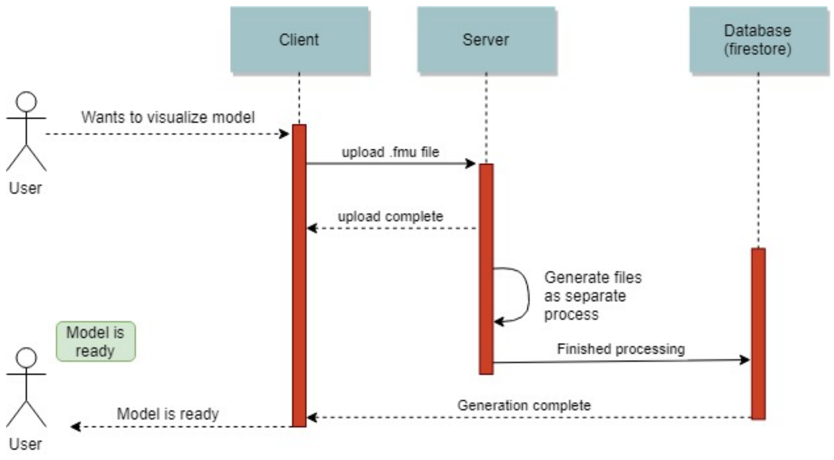


Figure 5.4: Process of uploading a model

5.4 Monitoring

Monitoring includes functionality for visualizing data in curve plots, video streams, and maps. It also provides functionality for event triggers and real-time predictions.

5.4.1 Curve Plot

The curve plot component displays the selected data as a function of time in a graph, either as a line or scatter plot. The plots are implemented with *React Plotly*, a JavaScript library used to draw charts. The tile renders using incoming real-time data from a datasource in a dynamic plot or queries historical data or data from a file to render a static plot. If the filtering of a datasource is desirable, this can be configured during creation. A Butterworth processor is created, configured, and started in the back-end using calls from the `backendAPI/processor.ts` file in the front-end. Processors are implemented in Jensen (2019). When the tile renders, a plot is built using Plotly the same way as for curve plots, but the real-time data comes from the processor instead of the raw data from the datasource. An abstracted example of a plot component is shown in Listing 5.20, where `<Plot />` is a component from the Plotly library.

```
 <S.PlotComponent>
      <S.Header>
      /* Code for header component */
      </S.Header>
      <S.Plot>
        <Plot data= data{} ... />
      </S.Plot>
</S.PlotComponent>
```

Code lising 5.20: Abstraction of curve plot. The `data` property contains the configuration parameters and the data that is rendered in the plot.

5.4.2 Video Streaming

Video streaming is implemented by displaying live YouTube videos with their free live streaming service. A Google account is required to use YouTube's live-streaming feature. The user navigates to YouTube's live dashboard in the web browser and press the "go live" button, and the attached camera can immediately start streaming. The user is given a unique YouTube video ID, which is used to display the video in the front-end solution. The live-stream is displayed in an *iframe* tag in the client. An iframe tag is typically used to display external objects within a web page, and it supports displaying videos.

5.4.3 Map

The *leaflet* and *react-leaflet* libraries are leveraged to implement the map feature. All the components in Listing 5.21 are react-leaflet components and constitute all the necessary code to render a map. Additionally, logic for updating the map is implemented and set to the `userLocation` attribute. The location is updated every five seconds for dynamic maps using JavaScript's `setInterval` method, but the frequency can be adjusted.

```
<Map center={[userLocation[0], userLocation[1]]}>
    <TileLayer
        url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png" />
    <Marker position={[userLocation[0], userLocation[1]]}
        <Popup>
          Coordinates: {userLocation[0]},{userLocation[1]}
        </Popup>
    </Marker>
</Map>
```

Code lising 5.21: `Map`, `TileLayer`, `Marker` and `Popup` are components imported from react-leaflet. OpenStreetMap is used to render the map based on the coordinates of the `url`.

5.4.4 Event Triggers

Event triggers functionality is implemented as a processor in the back-end. The processor receives real-time data and checks whether the value is higher than the maximum value or lower than the minimum value. If it is, a notification is created and stored in the database, as seen in Listing 5.22. The event is also stored in a dictionary to keep control of when the sensor returns to the normal state. If the time since the last notification is under 2 seconds, the back-end does not create a new notification to prevent multiple notifications in a short amount of time. While the incoming values are still below the minimum value (or above maximum), the trigger remains active. When the value returns to the normal state, it is updated to finished in the database and removed from the dictionary, as shown in Listing B.11 in Appendix B.2. The client listens to notifications in the database, receiving a notification in real-time if the sensor is not behaving according to the given limits.

```
if (value < minVal and time_since_last_notification > 2):
        database.create_notification(self.project_id, {
            u'sensor': sensor,
            u'startedAt': datetime.datetime.now(pytz.timezone('
                Europe/Oslo')),
            u'triggerReason': "Sensor value below minimum value",
            u'finished': False,
            u"severity": severity,
            u"description": description,
        })
```

Code lising 5.22: Simplified implementation for registering a notification when the value is below defined minimal value. The notification is created and updated to `finished` equal to True when the sensor is back to its normal state.

It is experimented by sending out SMS notifications to the users of a project if an event trigger is triggered. This is achieved by using a Norwegian supplier of SMS services called *Sveve*. SMS notifications can be implemented in the front-end by sending an SMS with Sveve's API when a notification for an event is received in the front-end. The code for sending an SMS is shown in Figure 5.23.

```
const sendSMS = (phoneNumber: string, message: string) => {
    fetch(
      "https://sveve.no/SMS/SendMessage?user=username&passwd=
          password&to=" +
        phoneNumber +
        "&msg=" +
        message +
        "&from=CBMS"
    );
  };
```

Code lising 5.23: Experimenting with SMS notifications to warn users about potential threats. `username` and `password` are credentials needed to use Sveve's API.

5.4.5 Real-Time Predictions

A simple predictive maintenance prototype was developed in Horn and Kjernlie 2019 to predict sensor values to tell if something is wrong with a sensor or a component. The prototype was implemented specifically for the torsion bar suspension rig in React with JavaScript and class components. The previously developed prototype is modified and implemented as a more generic system in this thesis. The machine learning logic is still implemented with *TensorFlowJS*; an open-source library that makes it possible to define, train and run machine learning models directly in the browser.

A more generic machine learning system is achieved by applying two significant changes to the prototype. The first change is a new configuration process, where the user can upload a dataset and connect the corresponding datasource directly to the model. The dataset used for training must be from the same datasource, as it relies on the sensors in the dataset. The second modification is that the model and the weights of the trained machine learning model are stored in the back-end instead of the user's browser. This makes it possible to access a machine learning model from different computers. It is achieved by creating API endpoints in the back-end, following TensorFlowJS's API requirements. When a user wants to save or load a machine model, it first requests a JSON file named `model.json`, a file which contains the topology and reference to the weights of the model. Then it requests a binary file named `model.weights.bin`, which carries the weights of the model. The names of these files are always the same, and one must implement the API endpoints in the described way to save and load models. The code for the API endpoints are shown in Listing 5.24.

```
@routes.post('/machinelearning/{project_id}/{tile}/model')
    # Save trained machine learning model and weigths to project
@routes.get('/machinelearning/{project_id}/{tile}/model.json')
    # Load and return the trained machine learning model
@routes.get('/machinelearning/{project_id}/{tile}/model.weights.bin
    ')
    # Load and return the trained weights
```

Code lising 5.24: API endpoints for storing and loading trained machine learning models and weights

Except for these changes, most of the machine learning logic remains the same. After a user uploads a dataset, the user is prompted to choose the sensor he or she wants to predict and which sensors should be used for the prediction. To choose the input sensors for the prediction, the user should always inspect the dataset before selecting the input parameters for the prediction. The data is pre-processed the same way as it is in Horn and Kjernlie (ibid.). The model is trained with the input parameters in Table B.3 in Appendix B.2. The trained model and its corresponding weights are then saved to the back-end. Trained models are used to predict sensor values in real-time in the client. Every time a new data point arrives from the selected datasource set in the configuration process, the model predicts the desired output based on the selected input parameters. As the model is pre-trained, it is not necessary to do any heavy calculations for predicting the sensor output in the front-end. The predicted output is then displayed with the actual output in a plot.

5.5 Post-Processing and Analytics

The platform implements functionality for creating FFTs, spectrograms and statistics. It is also possible to view and download historical data, as well as generating reports of the current tiles in the dashboard or by uploading a dataset.

5.5.1 Fast Fourier Transform

FFTs are generated in the back-end and plotted in the client using Plotly as a graph with frequencies on the X-axis and intensity on the Y-axis. FFTs are generated in the back-end from historical data from a data file or a datasource. The *SciPy* library is used to create an FFT, and *numpy* is used to extract the frequencies of the signal. Listing 5.25 shows a function that takes `data` and `sample spacing` as inputs and makes an FFT and corresponding frequencies which it returns. The data required for this function comes from historical datasource data or a data file depending on the FFT configuration decided by the user.

```python
def get_fft(data, sample_spacing):
    fft_values = scipy.fftpack.fft(data)
    frequencies = np.fft.fftfreq(data.shape[-1])
    intensities = 2.0 / data.shape[-1] * np.abs(fft_values[:data.
        shape[-1] // 2])
    return [[val for val in x], [float(val) for val in fft_values]]
```

Code lising 5.25: The function `get_fft` creates and returns an FFT of `data` with sample spacing `sample_spacing` using the numpy and scipy libraries.

5.5.2 Spectrogram

After configuration in the front-end, spectrograms are generated in the back-end using historical data from a datasource or contents of a data file as in the FFT. Listing 5.26 shows a function that creates spectrogram data, either from historical data from a datasource or a data file.

```python
def make_spectrogram(frequencies, duration):
    sampling_frequency = len(frequencies) / duration
    fig, ax = plot.subplots(1)
    power_spectrum, frequencies_found, time, image_axis = ax.
        specgram([f for f in frequencies], Fs=sampling_frequency)
    return [
        [[x for x in row] for row in power_spectrum],
        [freq for freq in frequencies_found],
        [t for t in time]
    ]
```

Code lising 5.26: The function `make_spectrogram` takes in data `frequencies` and duration `duration`, and creates a spectrogram using the *matplotlib* library.

The spectrogram data is generated in the back-end. The front-end receives the data from the back-end and plots the spectrogram using Plotly's *heatmap*, as seen in Listing 5.27. The z values are the color scale values, while the x and y are the axis values of the spectrogram.

```javascript
const plotData = [
      {
        x: jsonResponse[2],
        y: jsonResponse[1],
        z: jsonResponse[0],
        type: "heatmap",
        colorscale: "YlGnBu"
      }
    ];
```

Code lising 5.27: Plotting a spectrogram in the front-end using Plotly's "heatmap" type

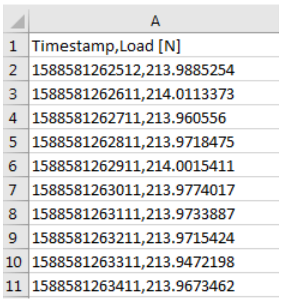
5.5.3 Statistics

Statistics are visualized in the front-end either as a table with statistical information or a histogram that shows the data distribution. The histogram is created using Plotly's histogram, shown in Listing B.5 in Appendix B.2. Listing B.6, also in Appendix B.2, illustrates the generation of the statistical information shown in a table. The information is generated using the JavaScript library *stats-lite*, but can also be generated without an external library. The code in both listings generates histograms and tables from a data file, but the same logic is generated from a data source's historical data. The data is provided from a call to the back-end.

5.5.4 Historical Data

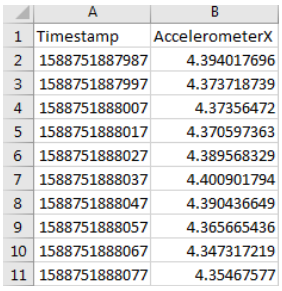
If one does not use real-time data, historical data from the datasources or data files can be used. If data from datasources is selected, the back-end retrieves a time series stored by Kafka. The result is the requested time series, which is used to generate the selected feature in the front or back-end. Alternatively, a data set can be uploaded, as seen in Listing B.4 in Appendix B.2. The function uploads the file and sends it to the back-end in a post request. Listing B.7 in Appendix B.2 shows the function that receives the request and saves it to the file system. When the tiles using the file is rendered in the front-end, the file is opened, and the data is used to generate the desired feature.

The data file must contain a UNIX timestamp in milliseconds as the first column and sensor data in the following column(s). The columns should contain a header in the first row with the name of the sensor or "Timestamp" for the first column. Figure 5.5 shows examples of correct data format of CSV and XLSX files respectively.



	A
1	Timestamp,Load [N]
2	1588581262512,213.9885254
3	1588581262611,214.0113373
4	1588581262711,213.960556
5	1588581262811,213.9718475
6	1588581262911,214.0015411
7	1588581263011,213.9774017
8	1588581263111,213.9733887
9	1588581263211,213.9715424
10	1588581263311,213.9472198
11	1588581263411,213.9673462

(a) Format of a CSV file



	A	B
1	Timestamp	AccelerometerX
2	1588751887987	4.394017696
3	1588751887997	4.373718739
4	1588751888007	4.37356472
5	1588751888017	4.370597363
6	1588751888027	4.389568329
7	1588751888037	4.400901794
8	1588751888047	4.390436649
9	1588751888057	4.365665436
10	1588751888067	4.347317219
11	1588751888077	4.35467577

(b) Format of an excel file

Figure 5.5: Expected format of CSV files and XLSX respectively

5.5.5 Downloading Data

If a tile's menu includes the header "Download," the user can download the content of the tile as a CSV or an XLSX file. The format of the downloaded data is the same as the required format for file uploads described in the previous section. This functionality is implemented using the *xlsx* and *file-saver* libraries. First, the selected data is converted to XLSX or CSV format using XLSX, and then the converted data is downloaded with file-saver, as seen in Listing 5.28.

```
const exportToCSV = (csvData: any, fileName: string, type: any) =>
    {
    const ws = XLSX.utils.json_to_sheet(csvData);
    const wb = { Sheets: { data: ws }, SheetNames: ["data"] };
    const excelBuffer = XLSX.write(wb, {
      bookType: type,
      type: "array"
    });
    const data = new Blob([excelBuffer], { type: type });
    FileSaver.saveAs(data, fileName);
  };
```

Code lising 5.28: Excerpt of the code that converts `csvData` to an XLSX or CSV workbook and downloads the converted data

5.5.6 Report Generator

Reports are generated as PDFs directly in the front-end through three steps; convert the DOM into an SVG, convert the SVG into a PNG, then convert the PNG into a PDF. This is necessary to save *screenshots* of the right elements. *HTML2Canvas* is a library that takes screenshots of elements directly on the user's browser into an SVG format. The screenshot may not be 100% accurate to what the user sees, as it is not an actual screenshot, but it creates a screenshot based on all the available information in the code. The SVG is then converted into a PNG with JavaScript. Converting the PNG to a PDF is achieved with an external library called *jsPDF*. Listing 5.29 shows the described process.

```
const pdf = new jsPDF();
/* ... code for adding a front page ... */
for tile in tiles {
     // if tile is a valid plot, do the following:
    const element = document.querySelector(tile);
    html2canvas(element).then((canvas) => {
        const elementToImage = canvas.toDtaURL("image/png");
        /* calculate image height and width */
        pdf.addImage(elementToImage, "PNG", xPosition, yPosition,
            imageWidth, imageHeight);
    })
}
pdf.save("report_name.pdf")
```

Code lising 5.29: Creating a report from a dashboard using the *html2canvas* and *jspdf* libraries

5.5.7 Inspect dataset

The user can upload a dataset in CSV format with the `FileUploader` component. The back-end receives the CSV file and uses *Pandas Profiling* to generate HTML profiling reports from the CSV file. The library creates a detailed interactive HTML report and returns it to the front-end. The report is visualized using an `iframe` tag in the front-end. Inspecting a dataset can be used for selecting the best input values for real-time predictions.

Chapter 6

Results

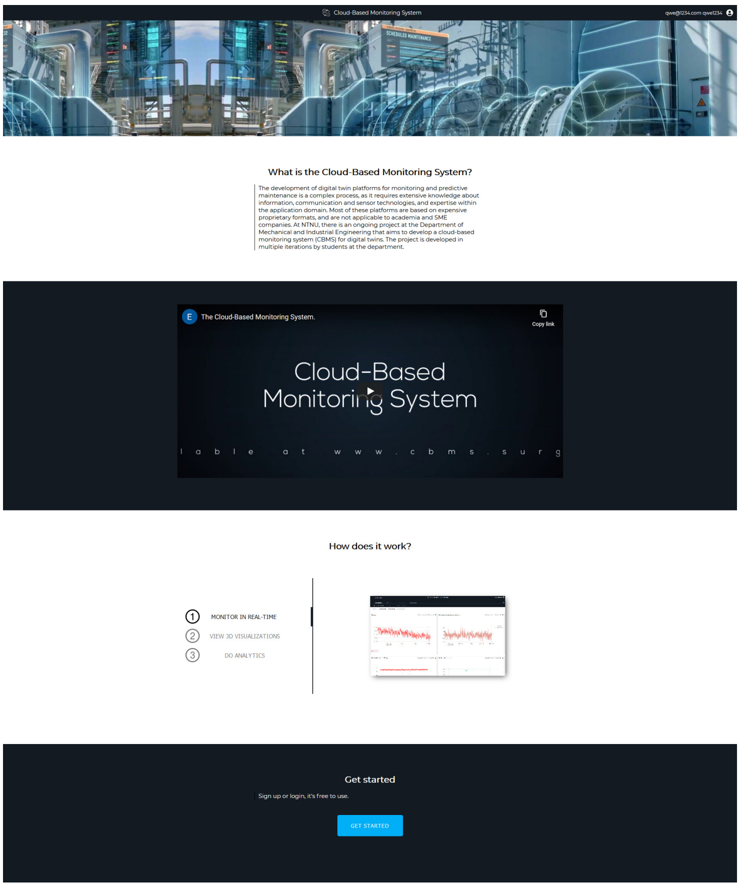
This section covers the results of the thesis, starting with the graphical user interface (GUI) of the front-end developed during the spring of 2020 and statistics from usability testing. Latency and availability statistics follow before the evaluation of analytics functionality is presented.

6.1 User Interface

The client consists of several routes and their respective sub-routes. The following sections present these routes, their respective GUIs, and how they can be used.

6.1.1 Landing Page

When the application loads, one is directed to the landing page as shown in Figure 6.1. This page sets the context for the application and the background, along with features the application provides. A video and an image slider show use cases for the system. The button at the bottom of the landing page leads to the sign-in page if the user is not logged in, or else the user is navigated to the user's home page. The process for signing in and signing up is shown in Figure 6.2.


Cloud-Based Monitoring System
qwe@1234.com qwe1234
The Cloud-Based Monitoring System.
Copy link
Cloud-Based Monitoring System
l a b l e a t w w w . c b m s . s u r g
How does it work?
1 MONITOR IN REAL-TIME
2 VIEW 3D VISUALIZATIONS
3 DO ANALYTICS
GET STARTED

What is the Cloud-Based Monitoring System?

The development of digital twin platforms for monitoring and predictive maintenance is a complex process, as it requires extensive knowledge about information, communication and sensor technologies, and expertise within the application domain. Most of these platforms are based on expensive proprietary formats, and are not applicable to academia and SME companies. At NTNU, there is an ongoing project at the Department of Mechanical and Industrial Engineering that aims to develop a cloud-based monitoring system (CBMS) for digital twins. The project is developed in multiple iterations by students at the department.

Get started

Sign up or login, it's free to use.

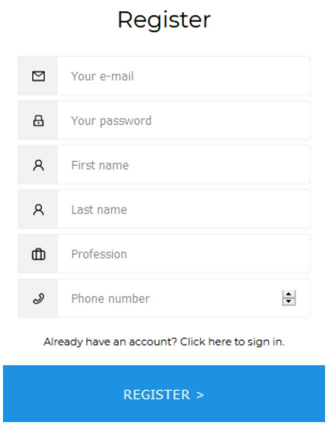
Figure 6.1: GUI of the landing page of the platform

6.1.2 Projects Page

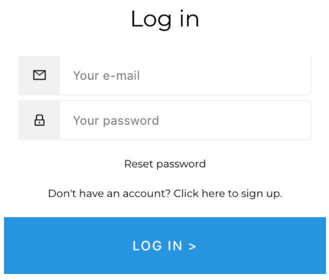
The projects page is the home page if the user is logged in. This page lists the projects the user has access to, along with information about the projects. The user can create a new project or navigate to an existing one by clicking on it.

6.1.3 New Project Page

The user can configure a project in the new project route. After selecting a name for the project, datasources and models can be configured. However, model and datasource configuration is also possible from an existing project. The initial stage of configuring a project is shown in Figure 6.4. If the user does not know the IP address of a datasource sending JSON data to the platform, it can be viewed as shown in Figure 6.5. Figure 6.6 and 6.7 show that the user can connect to new or existing datasources or upload different types of models and.

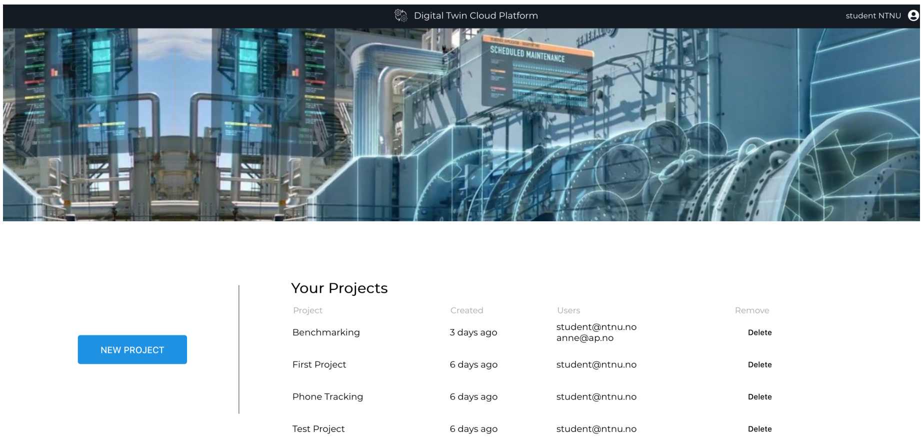


(a) Sign up



(b) Sign in

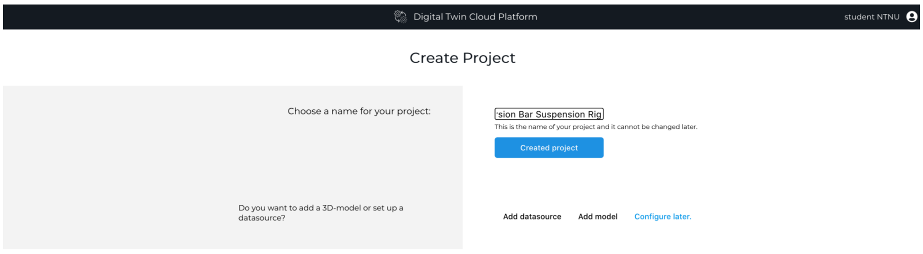
Figure 6.2: How to (a) create an account and (b) log into the system with an existing account



Your Projects

Project	Created	Users	Remove
Benchmarking	3 days ago	student@ntnu.no anne@ap.no	Delete
First Project	6 days ago	student@ntnu.no	Delete
Phone Tracking	6 days ago	student@ntnu.no	Delete
Test Project	6 days ago	student@ntnu.no	Delete

Figure 6.3: The projects route lists and displays information about existing projects.



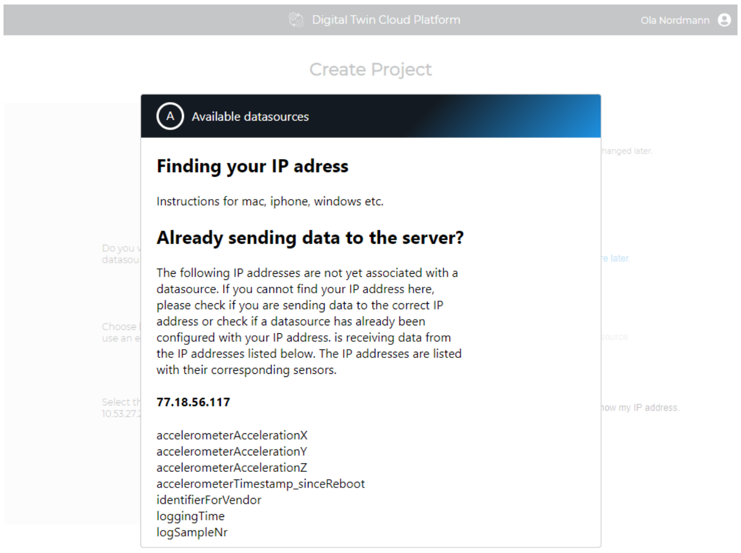
Create Project

Choose a name for your project: 'sion Bar Suspension Rig

This is the name of your project and it cannot be changed later.

Do you want to add a 3D-model or set up a datasource?

Figure 6.4: Initial stage of configuring a new project.



Finding your IP adress

Instructions for mac, iphone, windows etc.

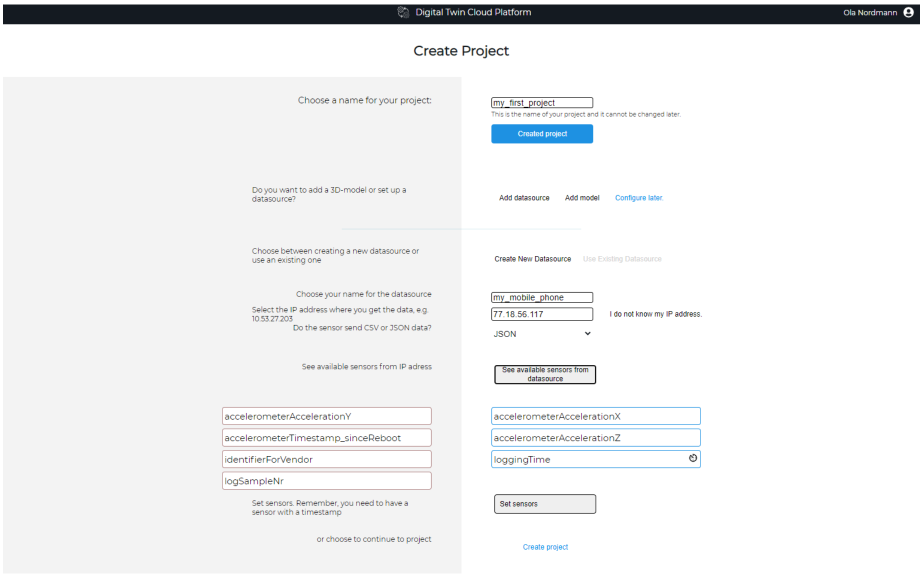
Already sending data to the server?

The following IP addresses are not yet associated with a datasource. If you cannot find your IP address here, please check if you are sending data to the correct IP address or check if a datasource has already been configured with your IP address. is receiving data from the IP addresses listed below. The IP addresses are listed with their corresponding sensors.

77.18.56.117

accelerometerAccelerationX
accelerometerAccelerationY
accelerometerAccelerationZ
accelerometerTimestamp_sinceReboot
identifierForVendor
loggingTime
logSampleNr

Figure 6.5: Viewing available datasources. A datasource with `77.18.56.117` as its IP address sends sensor data with seven available sensors.


Digital Twin Cloud Platform

Ola Nordmann

Create Project

Choose a name for your project: my_first_project

This is the name of your project and it cannot be changed later.

Created project

Do you want to add a 3D-model or set up a datasource? Add datasource Add model Configure later.

Choose between creating a new datasource or use an existing one Create New Datasource Use Existing Datasource

Choose your name for the datasource my_mobile_phone

Select the IP address where you get the data, e.g. 10.53.27.203 77.18.56.117 I do not know my IP address.

Do the sensor send CSV or JSON data? JSON

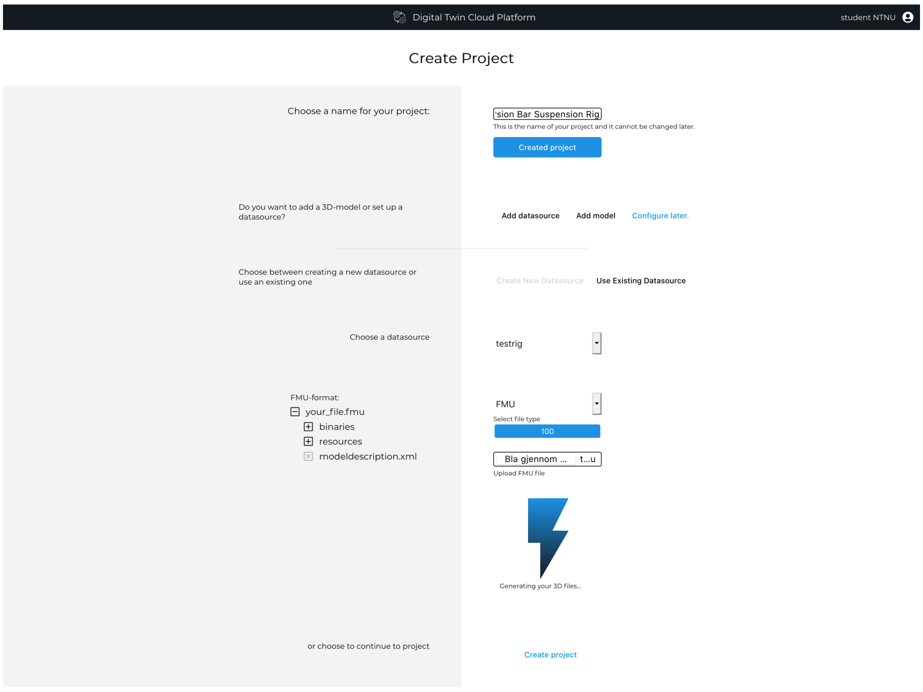
See available sensors from IP adress See available sensors from datasource

accelerometerAccelerationY	accelerometerAccelerationX
accelerometerTimestamp_sinceReboot	accelerometerAccelerationZ
identifierForVendor	loggingTime
logSampleNr	

Set sensors. Remember, you need to have a sensor with a timestamp Set sensors

or choose to continue to project Create project

Figure 6.6: Creating a new datasource during project configuration. The user has selected three sensors from the seven available sensors, where *loggingTime* is used as a timestmap value. The sensors with a red border are the remaining available sensors and the sensors with a blue border at the sensors the user wants to use.


Digital Twin Cloud Platform

student NTNU

Create Project

Choose a name for your project: sion Bar Suspension Rig

This is the name of your project and it cannot be changed later.

Created project

Do you want to add a 3D-model or set up a datasource?

Add datasource Add model Configure later.

Choose between creating a new datasource or use an existing one

Create New Datasource Use Existing Datasource

Choose a datasource: testrig

FMU-format:
your_file.fmu
binaries
resources
modeldescription.xml

FMU

Select file type

100

Bla gjennom ... t...u

Upload FMU file

Generating your 3D files...

or choose to continue to project

Create project

Figure 6.7: Uploading a model for a new project. The user uploads a FMU or FMM file and the files for 3D visualizations are automatically generated.

6.1.4 Project Page

The project page is the main page of the platform. This is where the digital twin is visualized, and the monitoring and analyses are executed.

Navigation bar

Figure 6.8 shows the navigation bar. The name of the project and information about the user is in the top left and right corners, respectively. More user information is displayed by clicking on the user icon, showed in Figure 6.9a. In the lower row of the navigation bar, the subsections of the projects are listed. One can navigate between these four views; dashboards, notifications, models, and configuration. One can invite other users to a project or initiate a group chat by clicking on the chat symbol to the far right. The chat view is shown in Figure 6.9b.

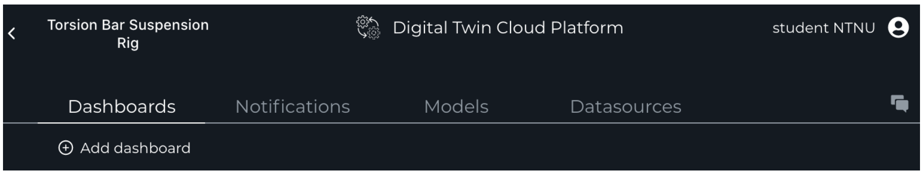
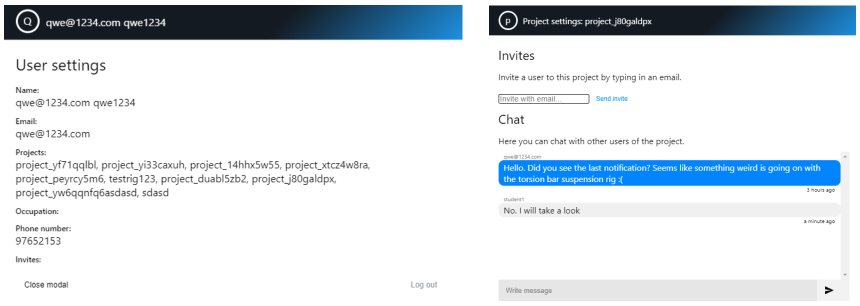


Figure 6.8: The navigation bar in the project route displays the tabs dashboards, notifications, models and datasources. The chat symbol to the right allows the user to invite another user to collaborate in the project or chat with other members of the project.


qwe@1234.com qwe1234

User settings

Name:
qwe@1234.com qwe1234

Email:
qwe@1234.com

Projects:
project_yf71qqlbl, project_yi33caxuh, project_14hhx5w55, project_xtcz4w8ra, project_peyrcy5m6, testrig123, project_duabl5zb2, project_j80galdpx, project_yw6qqnfq6asdasd, sdasd

Occupation:

Phone number:
97652153

Invites:

Close modal Log out

Project settings: project_j80galdpx

Invites

Invite a user to this project by typing in an email.

Invite with email... Send invite

Chat

Here you can chat with other users of the project.

qwe@1234.com
Hello. Did you see the last notification? Seems like something weird is going on with the torsion bar suspension rig :(
3 hours ago

student1
No. I will take a look
a minute ago

Write message

Figure 6.9: User settings displays the available information about a user (a). Project settings make it possible to invite other users to a project for collaboration (b).

Dashboard

The dashboard consists of a large canvas, empty at first. The user designs it according to his or her needs by creating *tiles* with different content. It is possible to add multiple dashboards if one wishes to physically separate the tiles. Figure 6.10 shows a sample dashboard called "CBMS" with two tiles.

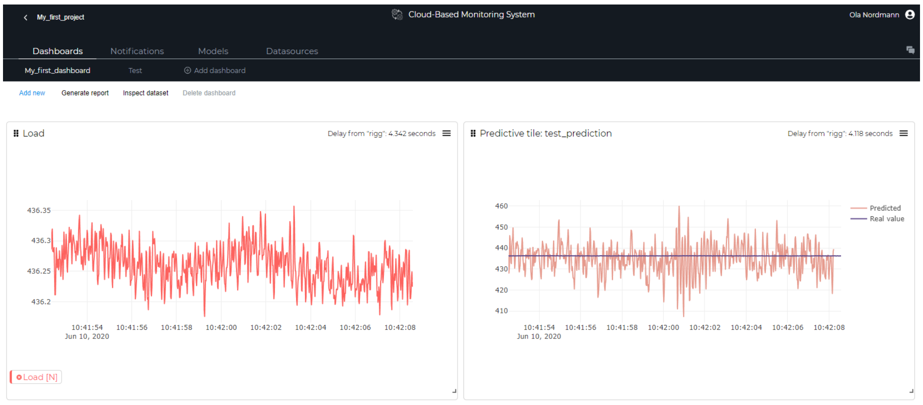
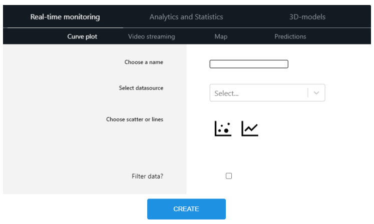


Figure 6.10: A dashboard with two tiles. Below the navigation bar, one can choose to add a new tile, generate a report, inspect a dataset or delete the current dashboard.

The monitoring tools are presented in Figure 6.11, configuration of and displaying curve plots, maps and sensor value predictions. User guides that describe in detail how to generate tiles are located in Appendix E.



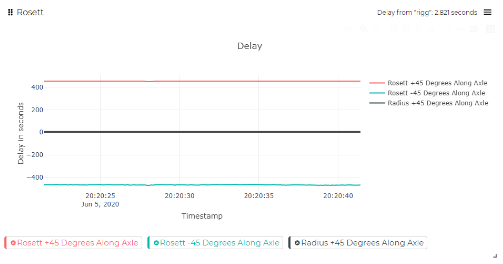
Choose a name

Select datasource: Select...

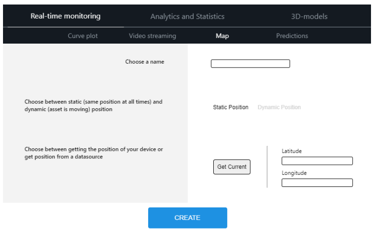
Choose scatter or lines

Filter data?

(a) Configuring a curve plot



(b) Curve plot



Choose a name

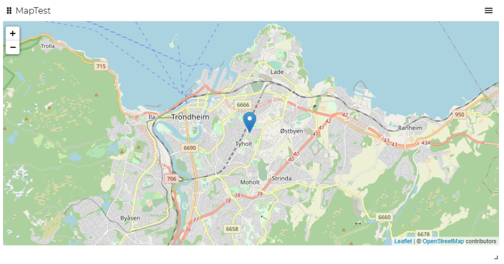
Choose between static (same position at all times) and dynamic (asset is moving) position

Choose between getting the position of your device or get position from a datasource

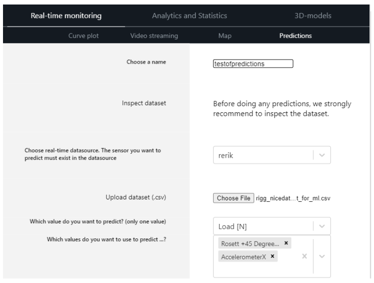
Latitude

Longitude

(c) Configuring a map



(d) Map



Choose a name: testofpredictions

Inspect dataset: Before doing any predictions, we strongly recommend to inspect the dataset.

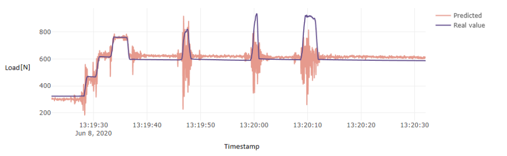
Choose real-time datasource. The sensor you want to predict must exist in the datasource: rerik

Upload dataset (.csv): rigg_nicedat...t_for_ml.csv

Which value do you want to predict? (only one value): Load [N]

Which values do you want to use to predict ...?: Rosett +45 Degree... AcceleromenterX

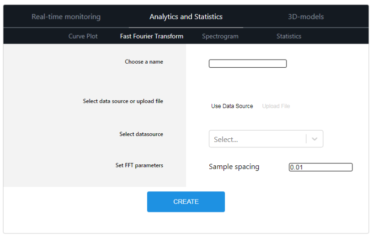
(e) Configuring real-time predictions



(f) Predictions plot

Figure 6.11: (a) Configuring and (b) displaying curve plots, (c) configuring and (d) displaying maps and (e) configuring and (f) displaying prediction plot.

The analytics tools are presented in Figure 6.12 for configuring and displaying FFTs, configuring and displaying spectrograms, and displaying statistics as a data distribution and a statistical summary.



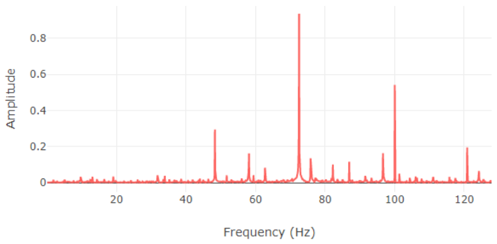
Choose a name

Select data source or upload file: Use Data Source / Upload File

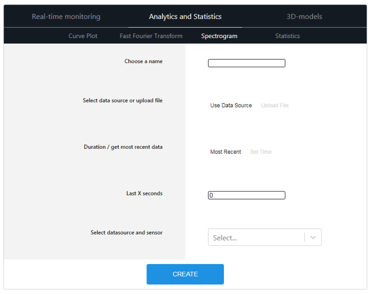
Select datasource: Select...

Set FFT parameters: Sample spacing 0.01

(a) Configuration of FFT



(b) FFT plot



Choose a name

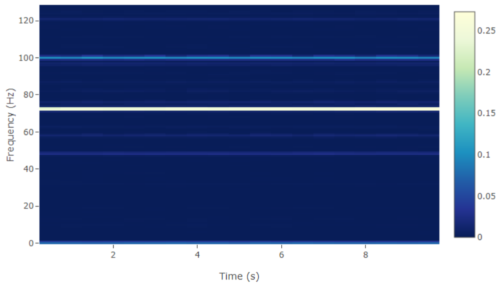
Select data source or upload file: Use Data Source / Upload File

Duration / get most recent data: Most Recent / Set Time

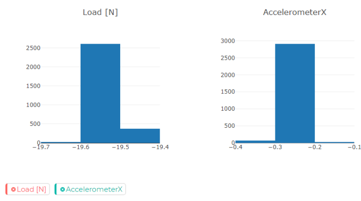
Last X seconds: 0

Select datasource and sensor: Select...

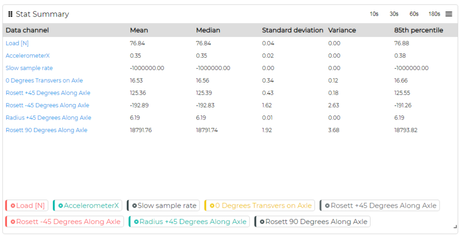
(c) Configuration of spectrogram tile



(d) Spectrogram plot



(e) Distribution of acceleration and load values


10s
30s
60s
180s
Load [N]
AccelerometerX
Slow sample rate
0 Degrees Transvers on Axle
Rosett +45 Degrees Along Axle
Rosett -45 Degrees Along Axle
Radius +45 Degrees Along Axle
Rosett 90 Degrees Along Axle

Stat Summary

Data channel	Mean	Median	Standard deviation	Variance	85th percentile
Load [N]	76.84	76.84	0.04	0.00	76.88
AccelerometerX	0.35	0.35	0.02	0.00	0.38
Slow sample rate	-1000000.00	-1000000.00	0.00	0.00	-1000000.00
0 Degrees Transvers on Axle	16.53	16.56	0.34	0.12	16.66
Rosett +45 Degrees Along Axle	125.36	125.39	0.43	0.18	125.55
Rosett -45 Degrees Along Axle	-192.89	-192.83	1.62	2.63	-191.26
Radius +45 Degrees Along Axle	6.19	6.19	0.01	0.00	6.19
Rosett 90 Degrees Along Axle	18791.76	18791.74	1.92	3.68	18793.82

(f) Statistical summary for sensors

Figure 6.12: (a) configuring and (b) displaying FFT, (c) configuring and (d) displaying spectrogram, displaying statistics tools: (e) distribution and (f) statistical summary of sensor data.

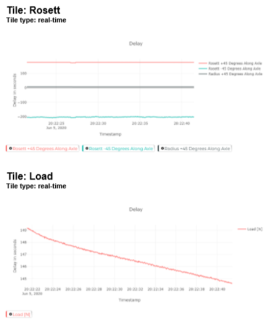
The user can generate a report by clicking the *generate report* button on the dashboard. The report saves the current tiles in the dashboard and downloads them as a PDF. Figure 6.13 (a) and (b) shows an example of a report. It is also possible to upload a CSV file and view detailed statistics about the dataset. Figure 6.13 shows parts of the results for a dataset from the torsion bar suspension rig, (c) statistical information and (d) correlation between the variables. The inspection can be used to determine the sensors that should be used for making predictions.

NTNU

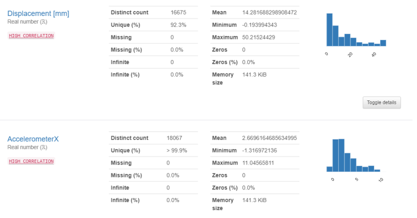
Digital Twin Cloud Platform

Project: sdasd
Dashboard: CBMS

(a) Front page of a report



(b) Plots in a report



Displacement [mm]
Real number (ℝ)
HIGH CORRELATION

Distinct count	16675
Unique (%)	92.3%
Missing	0
Missing (%)	0.0%
Infinite	0
Infinite (%)	0.0%

Mean	14.281688298908472
Minimum	-0.193994343
Maximum	50.21524429
Zeros	0
Zeros (%)	0.0%
Memory size	141.3 KiB

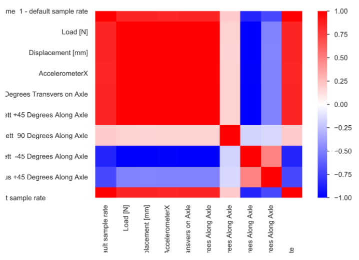
Toggle details

AccelerometerX
Real number (ℝ)
HIGH CORRELATION

Distinct count	18067
Unique (%)	> 99.9%
Missing	0
Missing (%)	0.0%
Infinite	0
Infinite (%)	0.0%

Mean	2.6696164685634995
Minimum	-1.316972136
Maximum	11.04565811
Zeros	0
Zeros (%)	0.0%
Memory size	141.3 KiB

(c) Descriptive statistics for the variables in inspect dataset



(d) Correlation between the variables in inspect dataset

Figure 6.13: Users can (a) (b) download a report from a dashboard or inspect a dataset and view (c) statistics about variables and (d) correlation between variables.

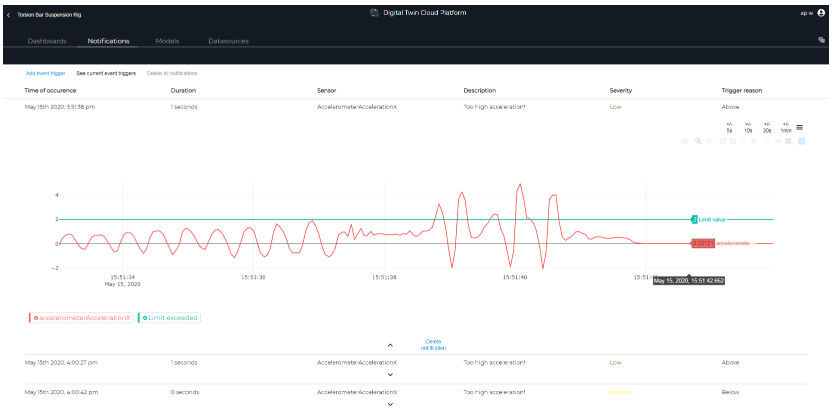
Notifications

Notifications keep track of the events in the system. The user registers boundaries for sensor values as event triggers, which generate notifications when the limits are exceeded. The notifications contain information about the event, and a plot of the event can be displayed as shown in Figure 6.14. Figure 6.15 shows the different notifications the user can receive.

Models

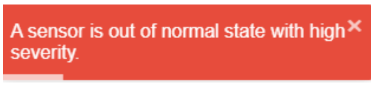
Models can be visualized in a dashboard tile, but there is a separate tab dedicated to 3D visualization with more functionality. One can visualize the model with different draw styles, including a surface with or without outlines or mesh, or as points or lines. Visualizations of the torsion bar suspension rig displayed with and without mesh are shown in Figure 6.16.

Two additional features are implemented for demonstration: adding a sensor and visualizing the model with colors. These features are shown in Figure 6.16b and 6.16c. A sensor can be added by clicking on the model after selecting to add a new sensor. The sensor is automatically positioning itself based on the angle of the surface. The color scheme shows the distances to a node in the model from a global and local coordinate system.

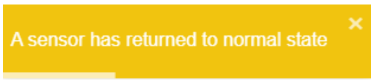


Time of occurence	Duration	Sensor	Description	Severity	Trigger reason
May 15th 2020, 3:51:38 pm	1 seconds	AccelerometerAccelerationX	Too high acceleration!	Low	Above
May 15th 2020, 4:00:27 pm	1 seconds	AccelerometerAccelerationX	Too high acceleration!	Low	Above
May 15th 2020, 4:00:42 pm	0 seconds	AccelerometerAccelerationX	Too high acceleration!	Medium	Below

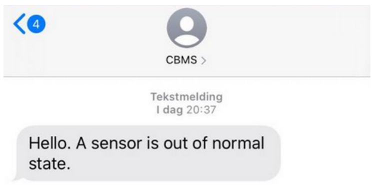
Figure 6.14: The notification tab displays the notifications in the project. They can be expanded to render a plot of the sensor values of the incident. The user can choose to view data 5, 10, 30 or 60 seconds before and after the incident.


A sensor is out of normal state with high severity.

(a) Sensor is not in normal state


A sensor has returned to normal state

(b) Sensor is back in normal state

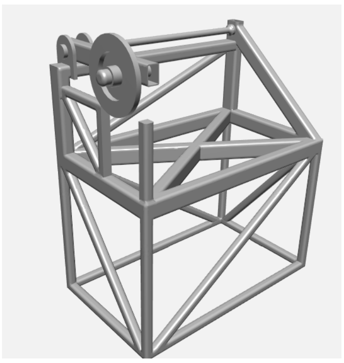

CBMS

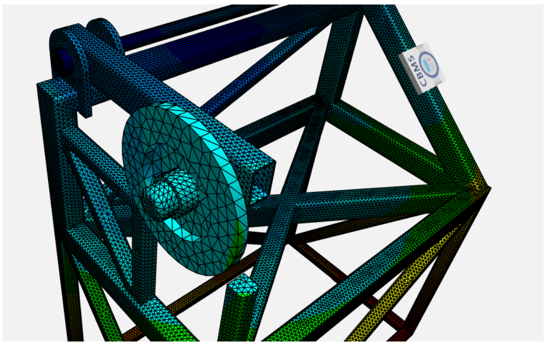
Tekstmelding
I dag 20:37

Hello. A sensor is out of normal state.

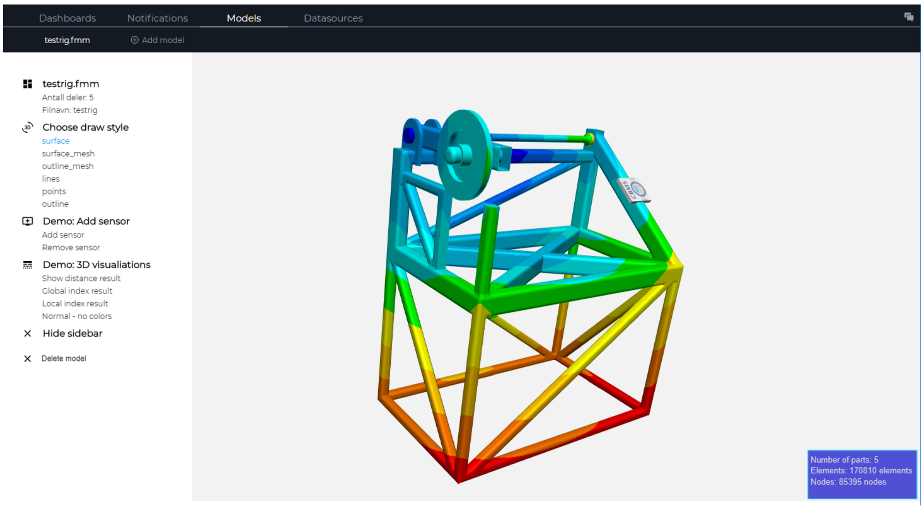
(c) SMS notification

Figure 6.15: The notification in the platform when a sensor is out of normal state (a) and returned to normal (b). (c) shows an example for an SMS when a sensor is out of normal state.


(a) Model without mesh



(b) Model with mesh



testrig.fmm
Antall deler: 5
Filnavn: testrig

Number of parts: 5
Elements: 170810 elements
Nodes: 85395 nodes

(c) Model with color visualization

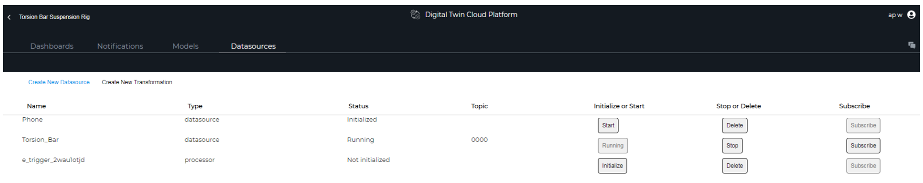
Figure 6.16: Visualization of the torsion bar suspension rig (a) without mesh, (b) with mesh, and (c) with color visualization. A sensor can be added as shown in (b) and (c).

Datasources

The last view in a project is the datasources page. Datasources are shown here, and the user can see if they are created or started. They can be stopped and deleted as well. More information appears by clicking on the source, *e.g.*, the sensor values it contains. This page allows the user to create a new datasource or apply a filter to an existing source, shown in Figure 6.17.

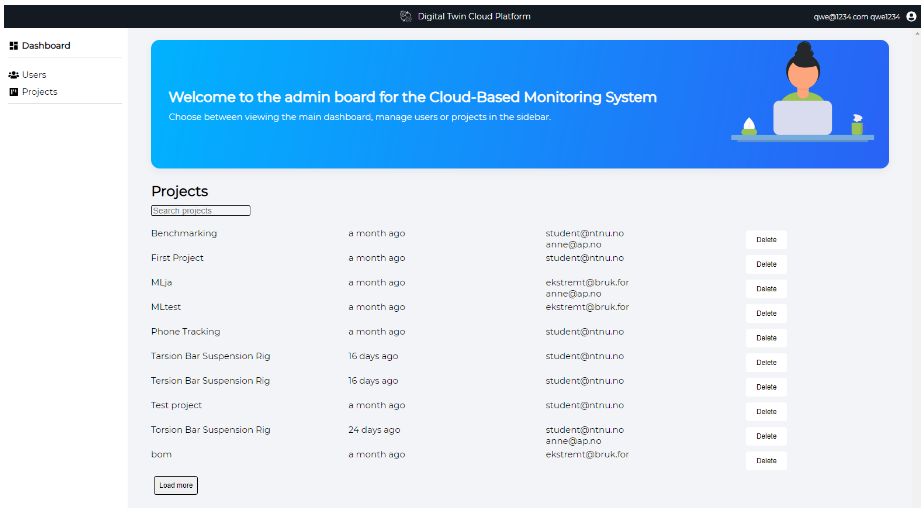
6.1.5 Admin page

The administrator page makes it possible for administrators to manage users and projects. The admin page is shown in Figure 6.18.


Torsion Bar Suspension Rig
Digital Twin Cloud Platform
ap w
Dashboards
Notifications
Models
Datasources
Create New Datasource
Create New Transformation

Name	Type	Status	Topic	Initialize or Start	Stop or Delete	Subscribe
Phone	datasource	Initialized		Start	Delete	Subscribe
Torsion_Bar	datasource	Running	0000	Running	Stop	Subscribe
e_trigger_2wau1otjd	processor	Not initialized		Initialize	Delete	Subscribe

Figure 6.17: Datasources page showing existing datasources in the project



Welcome to the admin board for the Cloud-Based Monitoring System

Choose between viewing the main dashboard, manage users or projects in the sidebar.

Projects

Search projects

Benchmarking	a month ago	student@ntnu.no anne@ap.no	Delete
First Project	a month ago	student@ntnu.no	Delete
MLja	a month ago	ekstremt@bruk.for anne@ap.no	Delete
MLtest	a month ago	ekstremt@bruk.for	Delete
Phone Tracking	a month ago	student@ntnu.no	Delete
Tarsion Bar Suspension Rig	16 days ago	student@ntnu.no	Delete
Tersion Bar Suspension Rig	16 days ago	student@ntnu.no	Delete
Test project	a month ago	student@ntnu.no	Delete
Torsion Bar Suspension Rig	24 days ago	student@ntnu.no anne@ap.no	Delete
bom	a month ago	ekstremt@bruk.for	Delete

Load more

Figure 6.18: Administrators can access the admin board to manage users and projects.

6.2 Usability

Table 6.2 shows the results from the questionnaires from the usability testing. The first question about the background of the user is in Table 6.1. A more detailed overview of the answers from the questionnaire is described in Appendix D.

	Mechanical Eng.	**Computer Science**	**Other**
Distribution	0%	75%	25%

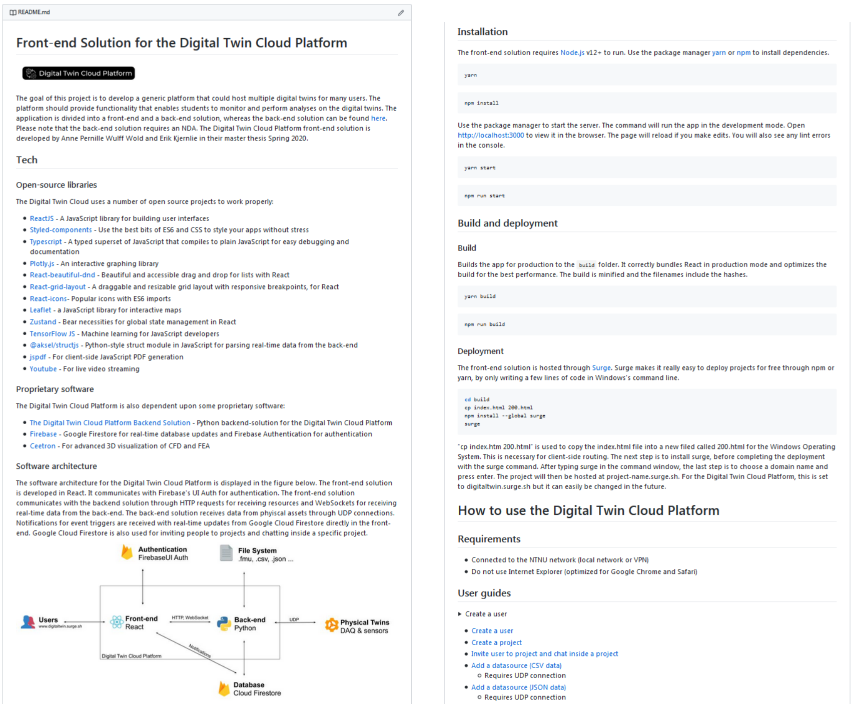
Table 6.1: Background of the usability testers

ID	**Statement**	**Avg. Score**
1	I am familiar with the field of digital twins	2.75
2	The platform is easy to use	4.00
3	It is easy to navigate within the platform	4.00
4	The platform is fast and responsiv.	4.50
5	It is easy to get an overview of available features	3.75
6	I will likely return the platform in the future	3.25
7	I find the platform attractive	4.75
8	The platform has a clean and simple presentation	4.50
	Average score of the questions related to usability	4.11

Table 6.2: Results from four participants answering the questionnaire after they have tested the platform. Each question is answered with a linear scale from 1 to 5, where users select 1 if they strongly disagree with an assertion and 5 if they strongly agree.

6.2.1 Documentation

README files have been created in each folder of the front-end repository that describe the files in the folder and their respective functions. Links to video user guides are provided in the README files. Figure 6.19 shows the main README. Appendix E contains user guides describing how to use the client of the platform. Appendix F is an installation guide on how to set up the project for future development. Appendix G shows the Surge deployment process.


README.md

Front-end Solution for the Digital Twin Cloud Platform

Digital Twin Cloud Platform

The goal of this project is to develop a generic platform that could host multiple digital twins for many users. The platform should provide functionality that enables students to monitor and perform analyses on the digital twins. The application is divided into a front-end and a back-end solution, whereas the back-end solution can be found here. Please note that the back-end solution requires an NDA. The Digital Twin Cloud Platform front-end solution is developed by Anne Pernille Wulff Wold and Erik Kjernlie in their master thesis Spring 2020.

Tech

Open-source libraries

The Digital Twin Cloud uses a number of open source projects to work properly:

- ReactJS - A JavaScript library for building user interfaces
- Styled-components - Use the best bits of ES6 and CSS to style your apps without stress
- Typescript - A typed superset of JavaScript that compiles to plain JavaScript for easy debugging and documentation
- Plotly.js - An interactive graphing library
- React-beautiful-dnd - Beautiful and accessible drag and drop for lists with React
- React-grid-layout - A draggable and resizable grid layout with responsive breakpoints, for React
- React-icons- Popular icons with ES6 imports
- Leaflet - a JavaScript library for interactive maps
- Zustand - Bear necessities for global state management in React
- TensorFlow JS - Machine learning for JavaScript developers
- @aksel/structjs - Python-style struct module in JavaScript for parsing real-time data from the back-end
- jspdf - For client-side JavaScript PDF generation
- Youtube - For live video streaming

Proprietary software

The Digital Twin Cloud Platform is also dependent upon some proprietary software:

- The Digital Twin Cloud Platform Backend Solution - Python backend-solution for the Digital Twin Cloud Platform
- Firebase - Google Firestore for real-time database updates and Firebase Authentication for authentication
- Ceetron - For advanced 3D visualization of CFD and FEA

Software architecture

The software architecture for the Digital Twin Cloud Platform is displayed in the figure below. The front-end solution is developed in React. It communicates with Firebase's UI Auth for authentication. The front-end solution communicates with the backend solution through HTTP requests for receiving resources and WebSockets for receiving real-time data from the back-end. The back-end solution receives data from phyiscal assets through UDP connections. Notifications for event triggers are received with real-time updates from Google Cloud Firestore directly in the front-end. Google Cloud Firestore is also used for inviting people to projects and chatting inside a specific project.

Installation

The front-end solution requires Node.js v12+ to run. Use the package manager yarn or npm to install dependencies.

```
yarn
```

```
npm install
```

Use the package manager to start the server. The command will run the app in the development mode. Open http://localhost:3000 to view it in the browser. The page will reload if you make edits. You will also see any lint errors in the console.

```
yarn start
```

```
npm run start
```

Build and deployment

Build

Builds the app for production to the `build` folder. It correctly bundles React in production mode and optimizes the build for the best performance. The build is minified and the filenames include the hashes.

```
yarn build
```

```
npm run build
```

Deployment

The front-end solution is hosted through Surge. Surge makes it really easy to deploy projects for free through npm or yarn, by only writing a few lines of code in Windows's command line.

```
cd build
cp index.html 200.html
npm install --global surge
surge
```

"cp index.htm 200.html" is used to copy the index.html file into a new filed called 200.html for the Windows Operating System. This is necessary for client-side routing. The next step is to install surge, before completing the deployment with the surge command. After typing surge in the command window, the last step is to choose a domain name and press enter. The project will then be hosted at project-name.surge.sh. For the Digital Twin Cloud Platform, this is set to digitaltwin.surge.sh but it can easily be changed in the future.

How to use the Digital Twin Cloud Platform

Requirements

- Connected to the NTNU network (local network or VPN)
- Do not use Internet Explorer (optimized for Google Chrome and Safari)

User guides

▸ Create a user

- Create a user
- Create a project
- Invite user to project and chat inside a project
- Add a datasource (CSV data)
  - Requires UDP connection
- Add a datasource (JSON data)
  - Requires UDP connection

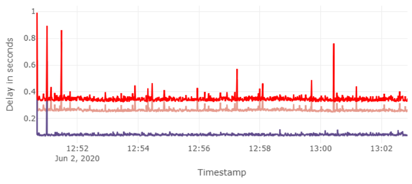
Figure 6.19: Main README for the front-end application.

6.3 Latency

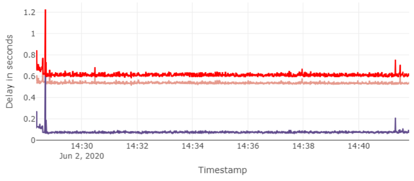
The latency is measured for curve plots and filters, the time it takes to receive a notification, and the time it takes to predict a value with a machine learning model.

6.3.1 Curve Plots and Filters

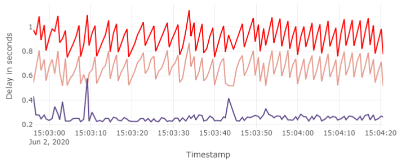
Figure 6.20 shows latency in four settings: (a) sending sensor data over Wi-Fi and (b) 4G, (c) the effect of rendering multiple tiles using different datasources and (d) the effect of applying a filter to a datasource. The red, pink and purple lines represent the total delay, delay from the asset to the front-end and internal front-end latency respectively. Table 6.3 lists average values obtained from the latency assessment.



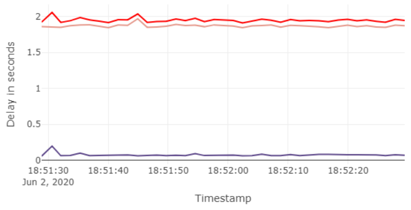
(a) Latency in signal over Wi-Fi



(b) Latency in signal over 4G



(c) Effect of rendering multiple tiles on front-end



(d) Effect of applying filter on back-end

Figure 6.20: Latency obtained over (a) Wi-Fi and (b) 4G of the SensorLog application, (c) the effect of multiple tiles and (d) the effect of applying a filter. Red, pink and purple lines represent total, asset to front-end and internal front-end delay respectively.

Asset	Until client	Client	Total	Network	Comment
SL	266	81	347	Wi-Fi	
SL	537	75	612	4G	
TBSR	1 676	74	1 750	Ethernet	
SL	663	257	920	4G	Multiple tiles
TBSR	1 785	210	1 995	Ethernet	Multiple tiles
SL	923	73	998	Wi-Fi	Filter, buffer size 1
SL	1 869	75	1 947	Wi-Fi	Filter, buffer size 20

Table 6.3: Delay in milliseconds of rendering one asset using Wi-Fi, 4G and ethernet, effect of rendering multiple tiles in front-end and filters using a buffer of size 1 and 20 respectively. Sampling rate is 20Hz. SL is the SensorLog application, and TBSR is the torsion bar suspension rig

6.3.2 Notifications

Figure 6.21 and Table 6.4 show the latency in the notifications. It takes on average 43 milliseconds from a sensor value is registered in an asset until a notification is created in the back-end. Almost 300ms later, a user is notified of the event.

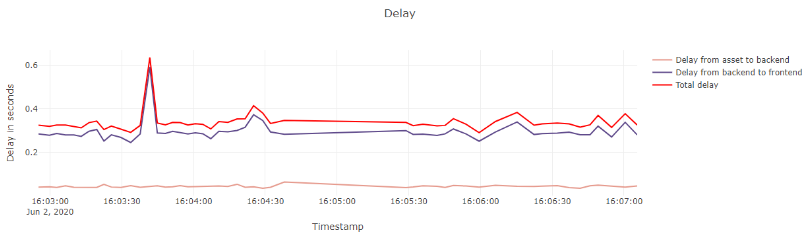


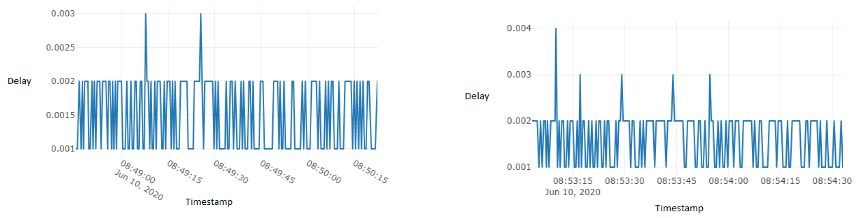
Figure 6.21: Delay from a sensor sends a value until the value is received in the back-end and from the value arrives until a notification is registered in the client.

Variable	Value
Number of notifications	50
Asset to back-end	43ms
Back-end to front-end	297ms
Total delay	340ms

Table 6.4: Average total delay for 50 notifications is 340ms; 43ms from the asset to the back-end and 297ms from back-end to the notification arrives in the front-end.

6.3.3 Predictions

The time for predicting with a pre-trained model of one sensor based on one and two input sensor are shown in Figure 6.22a and 6.22b. The plots are summarized in Table 6.5.



(a) Predictions delay plot with one input (b) Prediction delay plot with two inputs

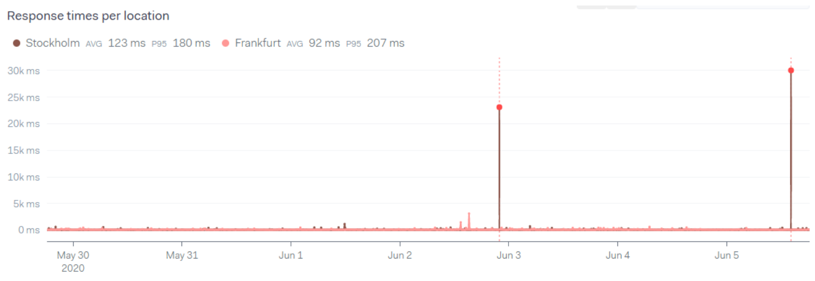
Figure 6.22: The time for making predictions when the model is trained with one (a) and two (b) inputs on a single output.

Number of inputs	Number of points	Average delay
One input	200	15ms
Two inputs	200	17ms

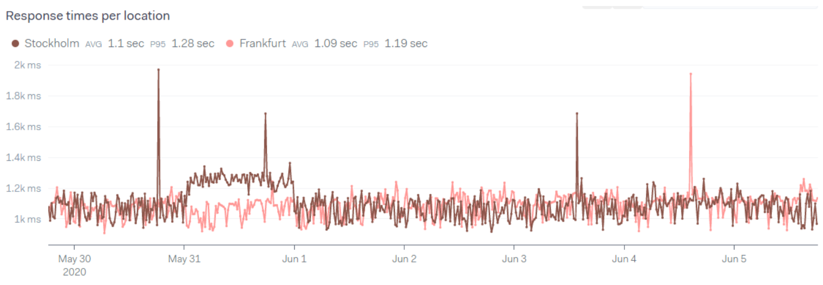
Table 6.5: The average delay for a prediction with 200 samples for a model trained on one and two inputs.

6.4 Availability and Performance

The statistics for the performance and availability of the CBMS generated by Checkly is shown in Table 6.6 and Figure 6.23.



(a) Response times for the back-end



(b) Response times for the front-end

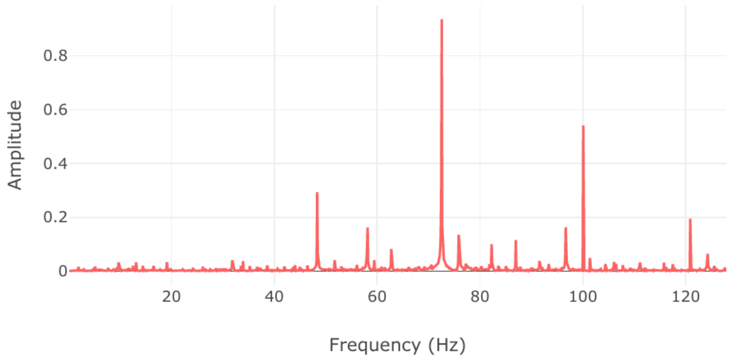
Figure 6.23: Performance metrics for (a) the back-end and (b) the front-end.

	24h ratio	**7d ratio**	**30d ratio**	**Avg.**	**p95**	**p99**
Server	99.653	99.900	99.923	177	153	380
Client	100.000	100.000	100.000	1100	1190	1230

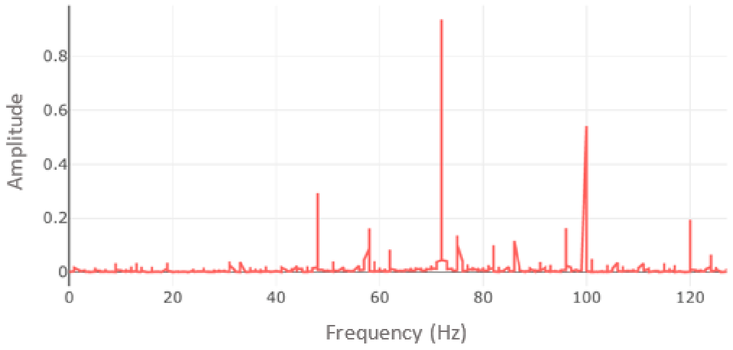
Table 6.6: The *ratios* show availability in percentage for the indicated period. *Avg.* is average response time for a request and *p95* and *p99* are the 95th and 99th percentiles. Response times are in milliseconds.

6.5 Functionality Validation

The results from the validation process described in Section 3.5 are illustrated in this section. Figure 6.24 shows (a) the FFT plot generated by the platform, and (b) a plot of SAP's FFT values. Table 6.7 shows the difference in frequency step values. Figure 6.25 contains (a) the generated spectrogram along with (b) the spectrogram plot provided by SAP.



(a) FFT generated from raw data

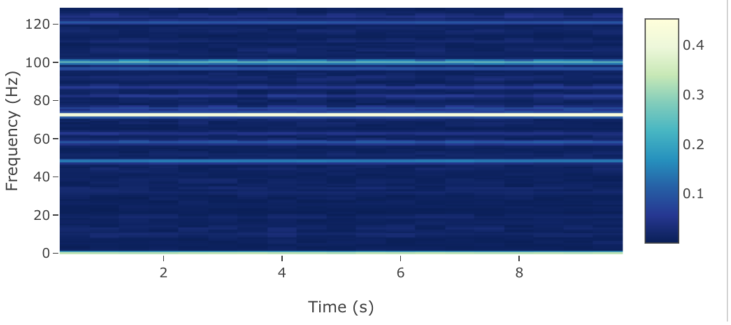


(b) Plot of SAP's FFT data

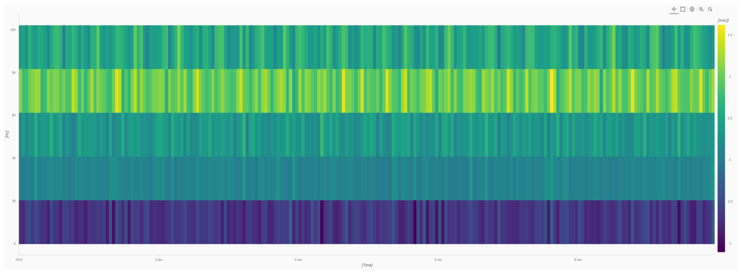
Figure 6.24: FFT generated (a) in the platform and (b) by SAP.

Variable	Generated Value	SAP Value
Frequency step	0.100039	0.100000

Table 6.7: Frequency step values obtained during the FFT generation in the platform and by SAP.



(a) Spectrogram generated in the platform



(b) SAP's spectrogram

Figure 6.25: Spectrogram plots generated (a) in the platform and (b) by SAP.

Chapter 7

Discussion

This chapter discusses the implementation and assessment of each objective defined for the thesis. The specific requirements are discussed, and the overall fulfillment of each objective is evaluated. Aspects related to the objectives are discussed, and suggestions for extensions and new research are listed.

7.1 User-friendly Client

The user experience of the CBMS is emphasized during development as the previous front-end solution did not meet usability requirements, as explained in Section 1.3. It is essential to hide unnecessary logic from the user while keeping the platform configurable so the user can customize it to their needs. Since the thesis is part of a long-term project, delivering a product that can be extended in the future is a success criterion of this objective. This section discusses the development of the front-end in terms of (1) meeting the defined usability requirements and (2) future development.

Execution of usability testing is described and evaluated. Usability testing could not be carried out according to the plan described in Section 3.1 due to the effects of COVID-19. However, some testing is executed, resulting in the statistics listed in Section 6.2. The results indicate that the requirements of the objective are met.

7.1.1 Usability

Requirement NFR03 is *The platform should be intuitive and easy to interact with.* A user-friendly system should be easy to learn, efficient to use, reduce the impact of errors, adapt to the user's needs, and increase the user's confidence and satisfaction. This is not a trivial task, and the design checklist described in 3.1 is used to comply with NFR03.

Learning the Features of the Platform

Allocation of responsibilities, *Data model*, and *Coordination model* are leveraged during development to implement a system that is easy to learn. Requirement NFR04, *Detailed documentation should be generated for the system in the form of README files, instruction videos, and user guides* manifests the importance of a platform that is easy to learn.

The platform has a landing page, videos, and detailed user guides to help users get started. Each project has a separate page with different tabs for managing and configuring features, e.g., the *datasources* tab for controlling data sources and processes and the *models* tab to visualize 3D models. The admin page manages the platform while the sign-in page is dedicated to log-in and registering a user. This is the separation and allocation of responsibilities, which clarifies and simplifies the learning process for the user.

The implementation of the system in terms of data and coordination model directly reflects how many and what kind of actions users have to perform to achieve their tasks. Work is put into abstracting logic to remove unnecessary configuration and execution of tasks by the user, for example by making it possible to visualize a data source with a filter in one operation, instead of multiple operations as required in the client developed in Børhaug and Sande (2019). Another example is the automation of 3D file generation when a user uploads an FMU file. Data abstractions and coordination within the client is essential to make it easier to learn and use the CBMS.

The system gives feedback to users to tell if they are using the system correctly. Input elements have a description or label to explain what the user has to fill in. Error messages are displayed with toasters or banners to tell the user if something is not correct. A user interface without feedback makes users wonder if they are doing something wrong or if there is an error in the system, which is essential for learning how to use the system.

Using the System Efficiently

Data model, *Coordination model*, *Resource management, mapping among architectural elements and binding time* are used to enable efficient use of the platform.

The efficiency of the system is related to the number of operations the user has to perform to achieve a goal. Web storage persists non-sensitive information of user preferences in the client. It keeps the data that is convenient for the user to see if the browser refreshes or if he or she is re-visiting the client from the same computer. This can, for instance, be the last tab the user visited, such as a specific dashboard in the dashboards tab. Whenever a user returns to a project, he or she is navigated directly to the tab last visited on the same computer.

The users trigger actions that use resources. Resource-consuming operations can potentially block the entire system and create long response times for the user. Configurations that take long, *e.g.* processing an FMU, are moved to a separate process in the back-end to avoid blockage. Users can create a data source or upload a model when creating a project or do it later, increasing the platform's flexibility, which directly impacts binding times. The user does not need to know about the processes running on the client or server-side. The machine learning training and predictions are both executed client side to reduce the load on the server, but the user does not know where it is happening.

Minimizing the Impact of Errors

Choice of technologies, *Data model*, *Coordination model*, and *Allocation of Responsibilities and Resource Management* are used to develop a system where the impact of errors is minimal.

Technology choices decide if the functionality is to be built from scratch or implemented using third-party libraries. External libraries are implemented as described in Section B.1 to save implementation time and reduce errors. An important reason for choosing an external library is that they are often less error-prone if used in other projects.

The platform is designed to prevent errors from happening. It should not be possible to click on buttons if not all the necessary fields are filled in, complying with FR08: *The user should get an error message if something goes wrong and be told how they can fix the problem.* If a user tries to create an account with an invalid e-mail, an error message is shown instead of creating an invalid user. Components are designed to set expectations of interactions to the desired action. For example, in the *Dashboards* tab, the button for adding a new tile is highlighted in blue, while deleting a dashboard is in faded grey, steering the user in the direction of the add tile-button and away from the delete dashboard-button. Actions with irreversible consequences, such as deleting a dashboard or a tile, requires the user to confirm that he or she wants to go through with the action.

Real-time data arrives with high frequency and is handled in a separate store. If this data is not handled efficiently, it can cause the browser to halt and crash. Instead of updating the entire interface every time a data point arrives, the store buffers the data and shares it with the rest of the application every 100 milliseconds. If a dashboard has any tiles using real-time data, only the components listening to the data are re-rendered. The CBMS is optimized to reduce CPU usage while it is run as a background task by preventing real-time updates if it is not an active tab in the browser. Without this optimization, it causes the browser to halt, as it would continue to re-render the application even though it is not in use.

Adapting the System to User Needs

Mapping Among Architectural Elements abstracts location of the user's data, and the *data model* is designed to adapt to the user's needs.

Mapping among the front-end, authentication service, storage units, and the back-end abstracts the data's location from the user. Communication between the elements makes sure the system is always up to date with the user's actions. Stores in the front-end receive updated data from all the architectural elements and distribute them to the components of the front-end that depend on it. The web storage saves non-sensitive information of user preferences, as previously mentioned. This information does not need to be stored on a remote server, as multiple users can access the same project. A user should not be navigated to a tab navigated to by another user. Hence, web storage creates a better experience for the user.

Increasing the Confidence and Satisfaction of the User

Data and Coordination Model and Resource Management. Instant feedback and consistent design increase the confidence and satisfaction of the user. Reusable components such as buttons and text inputs appear throughout the platform to make a consistent and professional impression. Preventing errors before they occur is an important step in increasing the confidence of the user. This is achieved by assisting and guiding the user, for example, by providing descriptive labels for filling in input fields. Another example is when a user fills in the log-in information when he or she is trying to register a user. If the user then switches to the tab for registering a user, the already entered information is automatically filled in the input fields for registering a user to increase the satisfaction for the user. Communicating with progress bars and offering suggestions make their actions more precise, which leads to fewer errors and higher confidence. All of the elements described in this section provide a positive experience for the user and impacts the satisfaction.

Results from Usability Testing

Due to the effects of COVID-19, usability testing could not be executed according to the plan, as mentioned. In the end, a total of four tests are carried out using the method described in Section 3.1. The results are shown in Table 6.2. The tests are included and discussed to demonstrate how it would be done if it was possible to follow through with the testing as desired, **not** to draw any conclusions as four tests is far from sufficient for that purpose. Ideally, a minimum of 25 testers with adequate backgrounds is desirable. Most of the students have a computer science background, and they have an average score of 2.75 for being familiar with the field of digital twins. Hence, the participants are not in the platform's target audience, which influences their perception of the system.

The platform can be overwhelming for a user who is unfamiliar with the field of digital twins. The platform still receives an average score of 4.00 (Q2) and 3.75 (Q5) for being easy to use and getting an overview of available features, respectively. This indicates that users can learn how to use the system efficiently. The platform receives a high score for being fast, responsive, and easy to navigate, where it gets a rating of 4.5 (Q4) and 4.00 (Q3), respectively. This indicates that the system helps the user to use the system efficiently and minimize the impact of errors. Q3 and Q6 also suggest that the platform adapts to the user's needs by automating tasks to make the platform less complicated and confusing. The platform receives a high score for being attractive and having a clean and simple design, both questions Q7 and Q8 gets a score above 4.5.

The average score of 4.11 for all the questions meets the requirements of being a user-friendly design according to the initial plan, where the requirement is a score above 3.93. The score, therefore, suggests that the platform satisfies the usability requirements of the user. The initial score of 3.93 is based on having an audience that is familiar with the field of digital twins. As the average participant is not in the platform's target audience, question 6 is not relevant, as there is a low likelihood that the user will return to the platform. If the score of this question is not included in the calculation of the average score, it increases to 4.25. This is a strong indication that the platform meets the requirements related to the usability of the platform.

7.1.2 Developing in React

A single-page React application is created to comply with NFR02: *The front-end should be developed using React.* It complies with the browsers required in NFR01: *The platform should be available in Google Chrome, Microsoft Edge, Mozilla Firefox, and Safari.* Libraries should be selected to comply with NFR01, and contribute to a user-friendly design. The application should also be easy to further develop.

External libraries

The first decision to make when developing new features of an application is whether to use external libraries or build them from scratch. An advantage is that it simplifies the implementation process. Still, one suffers a loss of flexibility by giving up control, and maintenance could get complicated if the library provider stops maintaining it.

One of the significant advantages of using React is its large active community, offering a wide range of opportunities in terms of libraries and resources available. One can usually find a library that simplifies the implementation no matter what features are needed, but one must be deliberate in the selection process. The library should be well-documented, have many downloads, and the library developers should still be active. A well-documented library simplifies the integration of the desired functionality as it is easy to find examples and use cases. The probability of bugs is lower if it has many downloads, and the developers of the library are still active. It is easier to get help on public forums if problems occur. If a library is not open-source, one should consider if the library is worth the money if there are other open-source options. Open-source software is preferred since the goal of the CBMS is to be available to anyone.

The tradeoff between time-constraints, complexity, and ease of use for future development is evaluated. The only proprietary software used in the system is Firestore for the real-time database and Ceetron for 3D visualization of FEM models. Ceetron is chosen as it is used in the earlier development of the CBMS (Børhaug and Sande 2019). There are no available open-source libraries for visualization of CAE models in the browser that currently meet the requirements for the CBMS.

TypeScript

An essential non-functional requirement for the CBMS is that it should be easy to continue to develop the software in the future. It is decided to use React with TypeScript, as TypeScript is more readable, and it is easier to understand the processes in the application than JavaScript. JavaScript is simpler and faster to develop, but Typescript is considered as a better option for a large scale project developed through multiple stages due to the potential amount of work needed to debug an error in an application with JavaScript for new developers. This is one of the main reasons why the old client (ibid.) is abandoned.

7.2 Personal and Persisting Projects

An authentication system to enable user log-in and a database are implemented to facilitate support for projects in the platform. The user-generated data in a project is stored in the database. An assessment of completion of the objective is performed successfully: the day after registering user accounts and configuring projects, both the testers logged in to find their projects as they had been configured the day before. Hence, the objective is fulfilled.

7.2.1 Authentication

To comply with requirement FR11: *The user should be able to register a unique account using their e-mail*, an authentication service is necessary. As stated in 2.4.3, there are many external services available which simplifies the implementation. Firebase is selected due to integration with the database. Firebase is free to use, and Google is responsible for the security when transmitting user credentials. It is implemented through a service layer in the client, as described in Section 5.2.1. The service layer simplifies the process of changing the authentication provider in the future if it is needed. Firebase Authentication guarantees a secure connection, and hence there is no need to go through the back-end to authenticate users. A request is therefore sent directly to the cloud provider with the user's credentials.

7.2.2 Database

The configuration of projects does not require the creation of users or the implementation of a database. The system could provide the same functionality in an ad-hoc format. Requirement FR12 is that *The user should be able to* *configure projects by adding models and connect to datasources. It should be possible to make a personal dashboard that is saved for later sessions.* Storing projects for a *particular user* over a longer time requires a database as web storage does not suffice for storing and sharing project data for multiple users, as explained in Section 2.4.4. Hence, a database is a prerequisite for compliance with requirement FR13, *It should be possible to share projects among multiple users by sending invitations.*

Two alternatives for a database are examined in this thesis; Firestore or a local SQL database running on one of NTNU's servers. Firestore offers a flexible solution with a simple integration process for both client- and server-side development. Access to the database from both the client and the server enables fast prototyping. Functionality can first be implemented directly in the client, then converted to Python, and moved to the back-end later. An SQL database running locally on a server takes more time to implement as it has a more complicated configuration process. A NoSQL database is preferable for the development of the CBMS in this thesis. The data requirements of the project change with fast, iterative prototyping, and a database with dynamic schemes makes it easier to add data at a later stage. Firestore is selected because of its flexibility and simple integration. As one gets closer to the production phase of the project, the need for flexibility decreases since the data structure converges. A relational database could replace Firestore as they provide a more powerful query language. Like with the implementation of the authentication service, the code for the database is implemented in a service layer, which simplifies the process of changing the database in the future.

The only drawback encountered during the process of implementing the Firestore database is deleting documents containing sub-collections. If one tries to delete a document that includes collections, it is not adequately removed, and might still be fetched in the client. Firestore provides one function for *deep* deletions, which is only compatible with JavaScript. Direct communication between the client and Firestore should be minimized, as it is easier to change the database in the future if the logic related to the database is implemented in the back-end, which is why the implementation of deleting projects is not implemented in the front-end. The only option for proper deletion is a customized implementation in the back-end, which can be complicated and costly in terms of execution time. The CBMS does not require frequent delete operations of large data structures, but future work should evaluate whether another database implementation is preferable.

7.3 Generic Configuration System

The implementation of a generic configuration system allows the user to connect to a data source and upload a model. A new asset is not made available for assessment due to COVID-19 restrictions. Hence, the configuration of a new digital twin and assessing the fulfillment of this objective is not possible following the method described in 3.3. Thus, there is uncertainty about how the platform performs with new assets. However, the system complies with the requirements related to the objective of implementing a generic configuration system.

7.3.1 Datasources

Functional requirements FR01a and FR01b are *The user should be able to connect to streams of sensor data by providing IP address* and *The platform should accept data streams in JSON and CSV formats.* Compliance with the requirements involved creating a configuration process for data sources in the client, and modifying datasource configuration in the back-end: removing port number and facilitating JSON object compatibility.

There are two reasons why the port number is no longer in use. The first reason is that the operating system can automatically select a port, which is often preferable as the operating system knows which ports are available. An effect of only using the IP address is that there can be only one connection for each device. It is not a loss of functionality as one should be able to collect and send all sensor data from an asset as it is easier to handle fewer connections. After implementing the video streaming feature through YouTube, it is a reasonable requirement that all sensor data arrives in one signal. Furthermore, port numbers might be challenging to configure from some applications. The SensorLog application used during development does not allow the user to see or edit port numbers. One would have to send data to an intermediate link that could forward the data using a specified port number, which more complicated.

When it comes to expanding support of the data format of the streams, there are several aspects to consider as there are advantages and disadvantages to the different formats. An effect of accepting multiple formats is more complex code for data handling in the back-end. The main advantage of transmitting the data as an array of data points (CSV) is that it is compact, and uses the actual sensor data. Handling this code on the server-side is simpler than JSON, as the data usually arrives in one message. This is because the size of the CSV data in bytes is assumed to be below the limit of the maximum transmission unit (MTU) on the network. However, the user needs to know precisely how the data is sent and specify the structure. It is cumbersome to fill out this information manually, and it might be challenging to find the configuration of the data.

Data sent as JSON objects is on a key-value format. The data points arrive with descriptive labels, simplifying the configuration of the data source for the user as the server automatically extracts the sensors and displays them to the user. Hence, the user selects the sensors they are interested in and does not need to know the details of the messages. However, there is an overhead of sending the data on a key-value format. If the packet size is larger than the MTU, it needs to be fragmented into multiple pieces. As the size of the message increases, the probability of packet fragmentation increases, meaning that the server must handle both packets arriving in non-chronological order and match packet fragments to retrieve the entire message. The data is sent in a CSV format when it is forwarded to the client, which restricts the issues mentioned above to the transmission from the physical asset to the server.

7.3.2 Models

There is one requirement concerned with the configuration of models, namely FR02: *The user should be able to generate a 3D model by uploading an FMU file.* Functionality for uploading virtual assets is implemented to accept models on FMU or FMM format. FMU is, as stated in Section 2.4.6, an instance of an open-source standard for dynamic simulation of models.

The user can upload a model, and the back-end starts the generation process. The server reads the file and generates JSON files for visualization, a resource-intensive task. Initially, the back-end did not respond to the upload request until the entire process was complete. The process can take minutes to perform, and the back-end temporarily blocked the connection between the server and other clients. The process is modified so that a response is sent to the user when the file is *received* in the back-end, which enables the processing of the file to run in a background process on the server. When the process has finished, a notification is sent to the client through the front-end, informing the user that their model has finished processing and can be visualized in the models tab.

7.4 Monitoring

The monitoring objective requires the implementation of tools that allow the user to see the current state of their assets. The user should get updates in real-time. The objective is evaluated according to the plan, and the results show that it is fulfilled.

7.4.1 Monitoring Tools

Requirements FR03 and FR04 concern monitoring tools that can visualize the state of the asset in different ways: *The system should visualize real-time data streams from physical assets* and *The system should visualize and update 3D models in real-time.* Curve plots, maps, and video streaming are implemented to comply with these requirements. A Butterworth filter provides processing of sensor values in real-time to remove noise from the signal per requirement FR10: *The user should be able to apply filters to datasources.* Real-time predictions of sensor values and event triggers notify the end-user of alarming sensor values and deviation from expected behavior. Requirement FR09 is *The system should provide predictive maintenance functionality to avoid equipment failure.*

Curve Plot

Curve plot exposes information about the state of the asset efficiently. Scatter and line graphs are available. Plotly offers a selection of graph types and configurations, and be used to add other plot types. Real-time data requires efficient handling, and the curve plot component is only re-rendered when new data arrives. Plotly has a function called *scattergl*, which is implemented to give better performance when a large number of data points are involved.

Filter

Configuration of processors uses four HTTP requests from the client to the server to create, configure and start a processor, which takes several seconds. The process should only use one request that contains all necessary configuration data as one usually wants to start a processor right away.

Maps

The current map implementation works well for the intended use. Possible additions include plotting historical positioning data in a map or tracing the position in a dynamic map. The two examples require similar code modifications and are possible to add to the current solution. React leaflet and leaflet.js are leveraged in the implementation, and provide a large number of components and functionality.

Video Streaming

Real-time video streaming supports inspections of physical twins from remote locations. In Kjernlie and Wold (2019), implementing video streaming is classified as "won't have" functionality as it would have required a lot of resources, both in terms of implementation time and storage capacity. An alternative solution is discovered and implemented through YouTube's streaming functionality. The user can set up a video stream through YouTube, which can be implemented in the front-end solution without implementing or storing any data. YouTube stores the live-stream for up to 12 hours, which makes it possible to rewind to view previous events. There are three delay options; *normal latency*, *low latency*, and *ultra-low latency*. Normal latency could be as high as 10 seconds, whereas the ultra-low latency reduces to a couple of seconds. Ten seconds is not acceptable as it is not within the threshold of "real-time"; hence ultra-low latency is preferable.

Models

Scripts from Ceetron facilitate model visualization in the front-end. Ceetron is a commercial company that works with 3D visualization, and the files are not in the Github repository due to NDA restrictions. An open-source solution is desirable, so future work should research an alternative to the Ceetron scripts.

It is possible to add a sensor to the model and select a data source. In the future, one should simulate stresses and strains in the entire object, and give an estimate of the chosen position's value by interpolation. This value can be visualized in the sensor's position. Stresses and strains, can be visualized with a color scheme, as shown in Figure 6.16c. The color scheme implemented in this project only uses dummy data for demonstration purposes.

An FMU processor is implemented in Jensen (2019) to update the state of the models. It was implemented in the client during development, but had a latency of several minutes. After discussions with the stakeholders, it was decided that the server of the platform should be improved before these processors are implemented due to the latency. Hence, visualizing displacements or stresses and strains in real-time is removed from the scope of the thesis, as it is not realistic to achieve real-time updates with the server's properties. However, this visualization is an essential part of digital twin application, and should be given a high priority in future work with the platform.

Event Triggers

Event triggers notify the user if a sensor is outside a defined *normal state* in terms of a minimum and maximum threshold. If the threshold is set to the limits where the equipment is no longer working, it alerts users about failure to trigger reactive (breakdown) maintenance. For preventive maintenance policies, thresholds should be set to trigger when the system degrades. This prevents sudden breakdowns. It is up to the user to select a maintenance policy, and the event triggers should be configured accordingly.

The event triggers are implemented as processors in the back-end, but send notifications to the front-end. If the user is not actively using the system, he or she is not informed about the event before they log in. However, one can easily extend to notify users through other mediums, *e.g.*, text messages, and e-mails. A text message notification prototype is implemented in the front-end, as shown in Listing 5.23. The user is notified with a text message from "CBMS", as shown in Figure 6.15c. The notification is only sent if the user is in the client, as the SMS is sent when a notification is received in the client. This can be implemented in the back-end in the event trigger processor or as *cloud functions* in Firebase. Cloud functions is a serverless framework that makes it possible to automatically run back-end code in Firebase in response to events in the Firestore database. When Firestore updates the database with a new notification, an event can be triggered to run the same code as in Listing 5.23. This solves the problem, but it requires knowledge about cloud functions. The most logical way is to implement it in a processor. However, this is a demanding task as processors do not handle asynchronous calls well.

An extension of event triggers is to implement composed event triggers, creating notifications if multiple members in a set of sensors have alarming values. One can also make event triggers based on location by defining a geographical area of operation. If the asset moves outside the area, the user can be notified. This is useful to control the position of a moving asset, *e.g.* an autonomous car or a research vessel.

Predictions

Predictions make it possible to predict if there is something wrong with the system. The current and predicted future state of the system can be used to make decisions, for example, if a sensor or component is malfunctioning and needs to be changed or fixed, or the load applied reduced. When these events occur, the user can be notified.

Predictions are implemented directly in the browser. The average prediction time with a single input value is 15ms, which increases to 17ms with two inputs as shown in Section 6.3.3. The predictions are therefore made in real-time. Real-time predictions are useful to identify a faulty or malfunctioning sensor by triggering an alarm if the predicted value deviates from the actual value with a certain threshold. Figure 6.11f shows differences between actual and predicted values in the three last peaks. In this example, the model is trained on a dataset from the torsion bar suspension rig, where the load is predicted based on an accelerometer value. Applying unexpected external forces such as a person putting weight on top of the torsion bar suspension rig results in a change in load, but the acceleration is not the same as it is during normal operations with the same load. The input parameters for the prediction do not change the same way as the output, which shows a clear deviation between predicted and actual values as shown in the figure.

The information from the predictions is useful to determine if there is something wrong with the system, such as the unexpected load, but this is still not a *predictive* system. A predictive system should alert the user if something is going to happen in the future; for example, if a sensor has a load of 400 N over two minutes, it tends to break down. These predictions can be used to prevent failure. For the torsion bar suspension rig, the model is only trained on a dataset made during normal operations. There is no data available for system failure events, so sensor or component failure cannot be predicted. Triggering alarms based on certain thresholds may trigger false alarms, as there might not be anything wrong with the system. Still, it can give a good indication if there is something that needs to be inspected more closely. Requirement FR09 is covered by the event trigger functionality. However, the predictions implementation is a solid foundation for advanced predictive maintenance in future iterations.

The training of the machine learning model is implemented in the front-end. Training in the browser is slower than a server solution, but this is not a problem if the RAM is able to allocate resources to handle the data. The assets integrated with the CBMS, such as the torsion bar suspension rig, are characterized by having small configuration spaces, which makes it possible to train an accurate model with a small dataset. *TensorFlowJS* is a JavaScript implementation of the *TensorFlow* library, a machine learning library compatible with multiple languages, such as Python. TensorFlowJS is a vast leap in bringing artificial intelligence capabilities directly to the browser, making it possible to make real-time predictions. TensorFlowJS is used due to ease of implementation because of experience and limited server capabilities mentioned in 4.3.3. The server has a limited number of CPUs, which might cause problems if multiple users want to train a model in the back-end at the same time. Doing these calculations in the browser helps to reduce the load on the server, as everything related to machine learning happens client-side. Training of models and real-time predictions should at a later stage happen on the server instead of the client. TensorFlow's tools and libraries in Python are more flexible and offer higher performance than TensorFlowJS. It is relatively easy to convert TensorFlowJS code from the front-end to Python code in the back-end, as both libraries are based on the same building blocks. Existing pre-trained models from TensorFlowJS can also be converted into a python compatible formats. Hence, models trained in the front-end can migrate to the back-end, which is essential to avoid data loss and corruption.

The model only predicts when it receives data, requiring an open browser at all times. It is a limitation, as it should be possible for a user to return to the platform and view historical events and alerts. Moving real-time predictions to the back-end makes it possible to continuously predict in the background and notify users if something appears to be wrong. A processor on the blueprint format can run predictions on incoming data the same way a filter or another processor works. If predicted values deviate too much from real-time values, the user should receive a notification.

7.4.2 Latency

Low delay is necessary to facilitate real-time updates of the monitoring tools, manifested in requirement NFR05: *The real-time data should have a latency of at most two seconds.* Ideally, data should appear instantaneously in the client for visualization. Several factors create delay: transmission of the sensor data from the physical assets to the client via the back-end, processing time in the back-end, and the characteristics of the system's hardware.

Transmission

Since the data sources transmit *real-time* data from the physical assets to the client via server, UDP is preferable over TCP due to high throughput. In a real-time monitoring application, it is vital to reduce the delay as much as possible. Packet loss is a small sacrifice to make to achieve minimal delay. UDP communication is already in place in the previous version of the platform, so no changes are required.

The server parses the data and forwards it to the front-end in a WebSocket connection using Kafka for message distribution. UDP and WebSocket aim at transmitting data with minimal delay and optimal throughput, as stated in Section 2.4.5, making them preferable in this setting compared to the alternatives. The sensor data must go through the server for several reasons: It must be processed, be available to all the users, and be stored for analytics. Web applications cannot communicate through UDP, which means that physical assets have to transmit sensor data using a WebSocket, which might not be supported. The SensorLog app only streams data using UDP or TCP. Filtered sensors have an additional delay due to having to go through the extra filtering step in the back-end.

Notifications from event triggers arrive in the front-end directly from Firestore. Firestore is suitable for this purpose, as it is simple to implement and keeps the data in sync across clients with real-time listeners. Listeners for notifying users if assets behave abnormally are implemented in the front-end, but can be moved to the back-end. Notifications can be transmitted in the existing WebSocket connection directly from the event trigger processor. This reduces the delay and eliminates the need for listening to notifications from event triggers in Firestore. However, Firestore must still be implemented in the front-end to receive real-time notifications such as invites to other projects.

Latency Results

The overall results show that the system has a total delay of less than the required two seconds. The delay is divided into two intervals: delay from the data is sent from the physical asset until it arrives in the front-end, and the delay in the front-end for receiving, processing, and visualizing the data. The latency is measured for curve plots and filters, but the results are similar for maps as the data is not processed in the front-end.

The internal delay is 75.6 ms on average for one asset, and the deviation for different assets is small, as Table 6.3 shows. This is expected since all data that arrives in the front-end is handled the same way. Since the tiles in a dashboard do not render simultaneously, there is an increase of 157.9 ms for the tiles that must wait when there are multiple tiles from distinct data sources. Figure 6.20c shows the increase in internal delay for the SensorLog application.

Inaccuracy in synchronization, back-end processing, and transmission time between the physical assets and the server, and the server and client represent additional delay. All the components are synchronized before initiation, but it is unlikely that all the components are in perfect synchronization. This can both add and reduce delays. Figure 6.20 (a) and (b) show delay from the SensorLog application sent over Wi-Fi and 4G respectively. Wi-Fi is approximately twice as fast, but the signal is more stable over 4G. Stability and speed depends on the individual connections, and can vary. Figure C.1 in Appendix C shows delay from the torsion bar suspension rig sent over an ethernet connection on the NTNU network. The delay from the asset to the front-end is significantly higher than the delay from the two others. The large delay is attributed to the synchronization of the DAQ software that transmits the data to the server. Visual inspection by jacking the rig while watching the client with a timer showed a delay of less than one second. Hence, the delay is not representative. Figure 6.20d shows the performance of a filtered sensor from the torsion bar suspension rig with a buffer size of 20. The additional delay of approximately 1.5 s compared to the delay of the unfiltered data is due to the additional processing in the back-end.

Figure 6.21 shows the delay in the signal from the asset to the back-end and from the back-end until a notification arrives in the front-end. There is a total delay of 340 ms, as Table 6.4 shows. The largest delay is between the server and client, which is not surprising as it is transmitted through Firestore. 340 ms is within the requirements for notifications.

7.5 Post-Processing and Analytics

Performance of post-processing and analyses require datasets and functionality applicable to the data. It is concluded that the objective is completed as the assessment proved that the implemented feature produces correct results from unknown data.

7.5.1 Datasets

Requirements FR05 and FR07 state that *The user should be able to visualize historical data* and *The user should be able to upload files in CSV or XLSX format*. To comply with these, the storage of historical data and data files is necessary.

Sensor Data Storage

Sensor data can be stored in Kafka for an unlimited time but requires a large storage space. Kafka's default retention time is seven days, but it is changed to three days during development to save storage space. The sensor data is stored in the original resolution, *e.g.*, 100 records per second if an asset samples sensor data with a 100 Hz frequency, which quickly fills up the current space of the server. Fetching historical data from sensors is not optimal due to inadequate querying tools and it blocks the entire platform because of concurrency issues in the same way as described in Section 7.3.2.

One should organize data storage according to how crucial the data is and how frequently it is accessed. Frequently accessed data should be located in *hot* storage, while other data should be in *cold* storage. Hot storage is more resource-intensive, as data retrieval and response times are much higher than cold storage services. Recent data should be stored in high, possibly original, resolution in hot storage for minimum latency access. The cold storage should comprise a more extended time series with a lower sample rate, as cold storage services generally support fewer inserts per second. Changing the storage model of the sensor data is not within the scope of the thesis and is thus not implemented. However, future work with the project should consider investigating a new storage model, as it would solve the performance problem of fetching records.

Server File System

Files used for analytics are stored in the file system on the server. One could first perform the analysis or transformation and store the result, but saving the original file enables use in other analyses. The drawback of this solution is that the analysis executes every time the tile using the file is rendered, slowing down performance slightly. However, it is considered more efficient due to the limited storage space, and the cost of storing each analysis result would have a larger impact.

7.5.2 Analytics Functionality

According to FR06, *The user should be able to post-process data, e.g., by computing FFTs.* Generation of FFTs and spectrograms is implemented to comply with this requirement. Section 6.5 illustrates the results obtained from the evaluation process described in Section 3.5.

Fast Fourier Transform

The validation process resulted in an almost identical FFT plot. Intensity values are similar; however, due to inaccuracy in sample spacing (not exactly 0.0039), there is a slight difference in the step value for the frequency. Still, it is only 0.04% larger than SAP's frequency step value, which is an acceptable difference.

A change that mitigates this imperfection is to use the timestamps from the data to calculate sample spacing to obtain more accurate values. The change can be quite easily implemented both for FFTs built on data sources and file data.

It can be challenging to interpret the result of an FFT. Noise from the signal is visible, and it is not always easy to separate it in the final FFT. The Welch method reduces some of this noise. There is no better implementation, as it depends on the use case. The industry usually applies the Welch method, whereas FFTs can be used in scientific environments as the end-user has more knowledge of signal analysis, and knows how to interpret the transformation.

Spectrogram

The spectrogram generated with data from SAP in the platform has the same axis values as SAP's spectrogram, and it is clear that the two spectrograms illustrate the same trend: highest values at about 75 Hz and smaller peaks around 50 Hz and 100 Hz. However, there are two substantial differences; the resolution of the axes and the values of the color plot.

One must decide whether one wants a high resolution of time slices or frequency values in the spectrogram. When the resolution of one axis increases, the other decreases. The main advantage of the high resolution of the time axis is accuracy in time of change of frequency. The exact rates before and after are not possible to deduct from this plot, only frequency intervals. Spectrograms with high resolution in the frequency axis have opposite characteristics: one knows the frequencies more accurately pre and post change but not the exact time of change. The spectrograms generated using the platform and by SAP have different configuration. The difference in values of the colors in the plot is a consequence of different resolutions: large frequency intervals gives a larger value interval than small. Hence, despite the visual differences, both the spectrograms are correct.

The configuration of a spectrogram only requires a dataset and a sampling frequency or duration. In the future, one should investigate whether the user should get more options when configuring a spectrogram, *e.g.*, setting the resolution parameters. It is a consideration of flexibility versus code complexity.

7.6 Other Aspects

During development, aspects that are not directly related to the objectives are encountered and investigated. For instance, the back-end was only available internally on the NTNU network before this thesis, which still limited prospective users to those who have an NTNU account despite changing from Feide to Firebase authentication. Additionally, parallelization is examined during the implementation of the generic system due to the blockage produced by the uploading process discussed in Section 7.3.

7.6.1 Accessibility and Deployability

The front-end is deployed and hosted through Surge, which makes it possible to access the front-end solution from any network around the world, as it is globally available. Even though the app is deployed, it needs a proper security review before it is used for more than research purposes. Surge is selected due to a simple hosting process, but it can also be hosted through other cloud services, such as stackit or Firebase, in the future. There have not been any problems related to hosting the platform through Surge, but it might be preferable to host everything through NTNU's cloud solutions. A meeting with stakeholders of the project at the department of mechanical engineering and the IT department should be conducted to discuss the best server and hosting solutions in the future.

The back-end is not ready for deployment due to security reasons, as described in the next section. Stackit makes it possible to host a VM internally on their *ntnu-internal* network, or globally for everyone through the *ntnu-global* network. As described in Section 4.4.4, Feide is not used as an authentication service, as the stakeholders desire that the CBMS should be available to anyone. The VM hosting the back-end solution is therefore connected to the *ntnu-global* network and is now accessible to anyone. This also makes it possible for sensors and physical assets to transmit data over a global network, without being logged onto the NTNU network or through a VPN.

7.6.2 Scalability

The back-end runs on the VM described in Section 4.3.3, which has 60 gigabytes of storage. Default applications occupy much of the space, which is why all unnecessary applications and files are deleted from the server at the end of the development phase to utilize the available storage space. The space available is still under 20 gigabytes after the cleanup process. Storage of datastreams in Kafka should not be a problem with a low number of users as data is only stored for three days. It is preferable to store data streams for an unlimited time, as discussed in Section 7.5.1, which requires either horizontal or vertical scaling of the server in the future. Another problem related to storage capabilities is that users can upload files of any size. The FMU file for the torsion bar suspension rig is over one gigabyte, which is a problem as only one FMU file can use more than 5% of the total available space. The best solution is to increase the server's resources, but one can also set a limit on the maximum file size.

7.6.3 Concurrency

The back-end solution needs to handle multiple processes simultaneously. The current server has four CPUs, making it possible to run four tasks in parallel. The CBMS manages many users simultaneously, and each user should be able to run multiple processes to filter and analyze data. It should also be possible to upload and process files while other tasks are running. This is not possible due to threading problems in the back-end. The *Global Interpreter Lock* (GIL) in Python prevents two threads from executing simultaneously, limiting parallel programming.

The multiprocessing package in Python side-steps the GIL by using sub-processes instead of threads, making it possible to leverage multiple CPUs at the same time, but the server still has limitations. It works well for processes that communicate through the WebSocket, such as filtering and processing a real-time data source. However, if a user sends an HTTP request that requires processing, such as generating an FFT, no other HTTP requests are managed until the result of the FFT is returned. When a user uploads a large FMU, it takes several seconds to process it to generate the 3D visualization files, which causes an availability issue. No one can access the CBMS while the server processes the FMU, as it cannot handle any other requests.

The concurrency issue when uploading an FMU is solved with the process explained in Figure 5.4. A separate process is started in parallel when the user uploads a file. The user is told that he or she receives a notification when the server has processed the model. When the back-end finishes the task, it updates Firestore with the information. The client is listening to Firestore and receives a notification. This is not an optimal solution, but it releases the blockage created by the FMU upload. As the problem occurs every time a user makes a request that demands processing in the back-end, it should be fixed with a solution that works well in all scenarios. The problem can be solved by using scripts in a language that handles parallel processing better, *e.g.,* Julia or C to execute the tasks.

It might be necessary to parallelize the back-end in the future. If a data stream is running, it is impossible to fetch historical data from the same data stream without interrupting the current data stream. This is related to the problem described above, but disappears if hot and cold storage of sensor data is implemented. Then, the DBMS handles all retrieval of historical sensor data. This would effectively side-step the GIL, as the back-end does not perform the heavy work related to reading messages from the file system.

7.6.4 Performance and Availability

As described in Section 7.6.3, concurrency issues can result in availability issues. To automatically monitor the availability and performance of the CBMS, a monitoring service called Checkly is set up. This is not possible if the back-end solution is hosted on an internal server, which was an incentive for transferring to NTNU's global network.

The results in Table 6.6 indicate that the client has 100% up-time and never fails. This is because the client is hosted through Surge, and it will only fail if Surge's servers fail. The average response time is over one second, which is the time it takes to load the landing page. The back-end server runs locally on one of NTNU's VMs. Checkly's API endpoint check fails if the back-end crashes or is down due to maintenance. The API check will also fail if *e.g.*, a concurrency issue occurs, and the back-end is not able to respond to requests. The average response time for a request to the server is 177 ms, which is much lower than the client. This is most likely due to the loading time of the web page before returning the response or the location of Surge's servers. Checkly performs request from Germany and Sweden, and the stackit server is located in Norway. However, the location of the front-end server is unknown. If the client is hosted on a server further away, it can have a negative impact on the response times. The availability of the back-end does not comply with NFR06: *The platform should have an up-time of at least 99.95%*, as it has an availability of 99.90% over the last seven days. An uptime of at least 99.95% is difficult when the back-end is not in production, as it is continuously being developed. The client fulfills the requirement, and it qualifies as a system of having *high availability*, which is typically referred to as having an availability of 99.999% or greater (Bass, Clements, and Kazman 2015). However, the client relies on the back-end, so the real up-time for the client can be argued to be the same as the back-end.

7.6.5 Security

Security has not been addressed during the development of the front-end solution or implementation of authentication and the database in Firebase, and should be considered in future work. It is also out of scope of the back-end development in Jensen (2019). It is crucial to keep users' data safe and protect the platform against unauthorized access. Additionally, GDPR increased businesses' responsibility to keep their users' personally identifiable information secure and transparent to the user.

Data should be encrypted when being stored, and sensitive personal information should not be used in the logic of the application, it should be replaced with an id. E-mail addresses link user profiles to projects, which should be replaced with unique identifiers so other users cannot access this information. Users should consent to the use of data and be told how their data is used upon registration, and they should be able to delete the data associated with their user. All communication between the components of the platform should be encrypted.

A globally accessible application comes at a risk, as anyone in the world can access the platform. An unencrypted HTTP protocol transmits data between the client and the server, making it possible for an attacker to intercept the communication during transmission. It is therefore recommended to configure the server to use HTTPS in the future for encrypting the connection between the client and the server.

As described in Jensen (ibid.), the back-end does not check all the input through the API. Modifications of the back-end for security reasons is not within the scope of this thesis, but should be considered in the future. The front-end checks that variables are correctly formatted before it sends requests. The back-end solution is vulnerable to cross-site request forgery (CSRF), as critical operations can be executed using GET methods, such as deleting projects, processors, or user profiles. These methods can be performed by anyone to read, modify, or delete information about a user, which violates the GDPR.

7.6.6 Digital Model, Shadow or Twin

There is no way of automatically sending instructions back to the assets as required by the definition of a digital twin in Section 2.1. Hence, the platform only supports digital shadows by definition. Providing two-way communication between the physical and virtual models is out of the scope of this thesis, which means that the platform strictly speaking cannot be more than a CBMS for

Digital Shadows at this point. As stated, this thesis is part of an ongoing long-term project conducted at the department of mechanical and industrial engineering at NTNU, and it is not realistic to finish the project in the course of this thesis. The objectives comprised developing other features of a platform that can become a digital twin platform in the future. Two-way communication should be in the scope of future work on the platform to accomplish the long-term ambitions.

Chapter 8

Further Work

Chapter 7 proposes many measures that can and should be taken to improve the CBMS. [Skrive litt at de krever forskjellig og vil gi forskjellig utslag og til slutt at de vi mener er viktigst kommer her] The authors believe the following aspects will have the most influence on the project:

1. Improve existing server solution to comply with non-functional requirements
2. Implement model simulation
3. Change storage of sensor data to hot and cold storage
4. Implement two-way communication
5. Extend Monitoring and Analytics functionality
6. Analyze security and ensure compliance with GDPR

Improve existing server solution to comply with non-functional requirements The server that the back-end runs on is a VM on NTNU's cloud server with a limited amount of resources and computing power. Even for a few users, these resources are too limited as the storage space fills up quickly. The discussion suggested vertical scaling as a temporary solution by adding more storage space and CPUs, and enabling automatic horizontal scaling as a long-term improvement. These changes improve the performance and open up for an increase in total simultaneous users.

Implement Model Simulation Simulation of 3D models is inevitable in the CBMS, as it is one of the core elements of a digital twin application. It is not implemented during development due to the poor performance of the prototype. Hence, the previous server should be improved as stated in the previous paragraph before model simulation can be implemented to give any valuable insight to the end-user.

Change storage of sensor data to hot and cold storage At this stage, the system only saves sensor data for three days due to storage capacity limitations in the server. It is desirable to keep recent sensor data stored in original frequency for a short period, and more extended time series in lower resolution, *i.e.*, a *hot* and *cold* storage solution. This way, the user obtains detailed information about recent events while still being able to see historical data.

Implement two-way communication: As stated in the definition of a digital twin, automatic data transfer *both* ways is required in a digital twin, meaning that as of now, the platform only supports digital shadows. Another Master's thesis has investigated possible implementations of bi-directional communication during the spring of 2020, whose findings might be compatible with this platform.

Extend monitoring and analytics functionality There are still several features that should be implemented in the platform, such as more filters and fatigue analysis, and the suggested extensions of existing functionality discussed in Chapter 7. A complete predictive maintenance system should be implemented to avoid failure.

Analyze security and ensure compliance with GDPR Security is out of the scope of all theses related to the CBMS project so far. Security measures to be taken should be investigated and implemented as the platform needs to be GDPR compliant before deployment.

Chapter 9

Conclusion

A user-friendly web application has been built to interact with the existing back-end solution of the Cloud-Based Monitoring System. An authentication service and a database have been set up to facilitate personal and persisting projects for the user. A generic configuration system has been implemented to enable adaption to other digital twin applications. The system allows the user to upload 3D models and connect to streams of sensor data from their physical twins using open standard formats. The physical assets can be monitored in real-time through a variety of features such as curve plots and maps. Notifications from event triggers and real-time predictions lets the user know if sensors are deviating from their expected values. The user can post-process and analyze historical data to gain insight about the physical asset.

User guides and instruction videos are produced for end-user support. Implementation has been carried out keeping future developers in mind by providing thorough documentation, including installation guides, README files, and a detailed implementation chapter.

Functionality and requirements have been evaluated and discussed. Interesting directions for future research have been presented. We conclude that the thesis has contributed to the CBMS project by complying with the defined objectives.

Bibliography

9126-4, ISO/IEC TR (2004). *Software engineering — Product quality — Part 4: Quality in use metrics*. URL: https://www.iso.org/obp/ui/#iso:std:iso-iec:tr:9126:-4:ed-1:v1:en.

9241-11, ISO (2018). *Ergonomic requirements for office work with visual display terminals (VDTs) — Part 11 Guidance on usability*. URL: https://www.iso.org/obp/ui/#iso:std:iso:9241:-11:ed-2:v1:en.

Aiyegbusi, Olalekan Lee (2019). "Key methodological considerations for usability testing of electronic patient-reported outcome (ePRO) systems". In: *Quality of Life Research* 29.2, pp. 325–333. DOI: 10.1007/s11136-019-02329-z.

Alonso-Ríos, D. et al. (2009). "Usability: A Critical Analysis and a Taxonomy". In: *International Journal of Human–Computer Interaction* 26.1, pp. 53–74. DOI: 10.1080/10447310903025552.

Baldini, Ioana et al. (2017). "Serverless Computing: Current Trends and Open Problems". In: *Research Advances in Cloud Computing*, pp. 1–20. DOI: 10.1007/978-981-10-5026-8_1.

Barros, Anne (2019). *Compendium in Data Driven Prognostic and Predictive Maintance. TPK4450.*

Bass, Len, Paul Clements, and Rick Kazman (2015). *Software architecture in practice*. Addison-Wesley.

Batty, Michael (2018). "Digital twins". In: *Environment and Planning B: Urban Analytics and City Science* 45.5, pp. 817–820. DOI: 10.1177/2399808318796416.

Belshe et al. (2015). "Hypertext Transfer Protocol Version 2 (HTTP/2)". In: DOI: 10.17487/rfc77540.

Bevan, Nigel (1995a). "Measuring usability as quality of use". In: *Software Quality Journal* 4, pp. 115–130.

— (1995b). "Usability is Quality of Use". In: *Symbiosis of Human and Artifact*. Ed. by Yuichiro Anzai, Katsuhiko Ogawa, and Hirohiko Mori. Vol. 20. Advances in Human Factors/Ergonomics. Elsevier, pp. 349–354. DOI: https://doi.org/10.1016/S0921-2647(06)80241-8.

Bevan, Nigel, Jurek Kirakowski, and Jonathan Maissel (Jan. 1991). "What is Usability?" In: *Proceedings of the 4th International Conference on HCI.*

Bhardwaj, S., L. Jain, and S. Jain (2015). "Cloud Computing: A Study of Infrastructure As a Service (IaaS)". In: *International Journal of the Academic Business World.*

Birns, Julie H. et al. (2002). *Getting the Whole Picture: Collecting Usability Data Using Two Methods – Concurrent Think Aloud and Retrospective Probing.*

Bishop, Matt (2005). *Introduction to computer security.* Addison-Wesley.

Blockwitz, T. et al. (2012). "Functional Mockup Interface 2.0: The Standard for Tool independent Exchange of Simulation Models". In: *Proceedings of the 9th International MODELICA Conference, September 3-5, 2012, Munich, Germany.*

Børhaug, A. and O. H. S. Sande (2019). "Developing a Client for a Digital Twin Cloud Platform". MA thesis. Norwegian University of Science and Technology.

Carvalho, Thyago P. et al. (2019). "A systematic literature review of machine learning methods applied to predictive maintenance". In: *Computers & Industrial Engineering* 137. DOI: 10.1016/j.cie.2019.106024.

Daylami, Nozar (2015). "The Origin and Construct of Cloud Computing". In: *International Journal of the Academic Business World* 9.2, pp. 39–43.

Foster, Ian, Carl Kesselman, and Steven Tuecke (2001). "The Anatomy of the Grid: Enabling Scalable Virtual Organizations". In: *The International Journal of High Performance Computing Applications* 15.3, pp. 200–222. DOI: 10.1177/109434200101500302.

Goyal, Sumit (2014). "Public vs Private vs Hybrid vs Community - Cloud Computing: A Critical Review". In: *International Journal of Computer Network and Information Security* 6.3, pp. 20–29. DOI: 10.5815/ijcnis.2014.03.03.

Granevang, M. (2019a). "Backend". In: *Store norske leksikon på snl.no.*

— (2019b). "Frontend". In: *Store norske leksikon på snl.no.*

Grieves, M. (2014). "Digital twin: manufacturing excellence through virtual factory replication". In: *White paper.*

Haak, Maaike van den, Menno De Jong, and Peter Jan Schellens (2003). "Retrospective vs. concurrent think-aloud protocols: Testing the usability of an online library catalogue". In: *Behaviour & Information Technology* 22.5, pp. 339–351.

Horn, H. and E. Kjernlie (2019). “Predictive maintenance and monitoring using Machine Learning”. Project in the course TPK4450: Data Driven Prognostics and Predictive Maintenance.

Hornbæk, Kasper and Effie Lai-Chong Law (2007). “Meta-Analysis of Correlations among Usability Measures”. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI ’07. San Jose, California, USA: Association for Computing Machinery, pp. 617–626. ISBN: 9781595935939. DOI: 10.1145/1240624.1240722.

Hosch, W. (2015). “Client-server Architecture”. In: *ENCYCLOPEDIA BRITANNICA*.

Jensen et al, S. N. (2018). “Cloud Software For Digital Twin Modeling And Monitoring”. MA thesis. Norwegian University of Science and Technology.

Jensen, S. N. (2019). “Building an extensible prototype for a cloud based digital twin platform”. MA thesis. Norwegian University of Science and Technology.

Johansen, C. (2019). “Digital Twin Of Knuckle Boom Crane”. MA thesis. Norwegian University of Science and Technology.

Jokela, Timo et al. (Jan. 2003). “The standard of user-centered design and the standard definition of usability: Analyzing ISO 13407 against ISO 9241-11”. In: *ACM International Conference Proceeding Series* 46, pp. 53–60.

Kjernlie, E. and A. P. W. Wold (2019). “Evaluating the Cloud Based Monitoring System for Further Development”. MA thesis. Norwegian University of Science and Technology.

Kotsiantis, Sotiris B., Dimitris Kanellopoulos, and Panayiotis E. Pintelas (2007). “Data Preprocessing for Supervised Leaning”. In: *World Academy of Science, Engineering and Technology, International Journal of Computer, Electrical, Automation, Control and Information Engineering* 1, pp. 4104–4109.

M. Grieves, J. Vickers (2016). “Digital Twin: Mitigating Unpredictable, Undesirable Emergent Behavior in Complex Systems (Excerpt)”. In: *Trans-Disciplinary Perspectives on System Complexity*.

NSW, Digital (2019). *NSW Digital Twin*. digital.nsw. Retrieved on 30-05-2020.

Pimentel, V. and B. G. Nickerson (2012). “Communicating and Displaying Real-Time Data with WebSocket”. In: *IEEE Internet Computing* 16.4, pp. 45–53.

Rosen, Roland et al. (2015). “About The Importance of Autonomy and Digital Twins for the Future of Manufacturing”. In: *IFAC-PapersOnLine* 48.3, pp. 567–572. DOI: 10.1016/j.ifacol.2015.06.141.

Sandtveit, E. (2019). “Exploring Azure as cloud provider for digital twin monitoring”. MA thesis. Norwegian University of Science and Technology.

Sauro, Jeff (2015). “SUPR-Q: a comprehensive measure of the quality of the website user experience”. In: *Journal of Usability Studies archive* 10, pp. 68–86.

Shimonski, R., M. Cross, and L. Hunter (2005). “Network+”. In: pp. 317–433.

User Datagram Protocol (1980). RFC 768. DOI: 10.17487/RFC0768.

Wathan, Adam and Steve Schoger (2018). “Refactoring UI”. Only published electronically on their web page.

Zhang, W., D. Yang, and H. Wang (2019). “Data-Driven Methods for Predictive Maintenance of Industrial Equipment: A Survey”. In: *IEEE Systems Journal* 13.3, pp. 2213–2227.

Zhou, K., Taigang Liu, and Lifeng Zhou (2015). “Industry 4.0: Towards future industrial opportunities and challenges”. In: *2015 12th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD)*, pp. 2147–2152.

Zimmermann, A., W. Eddy, and L. Eggert (2016). “Moving Outdated TCP Extensions and TCP-Related Documents to Historic or Informational Status”. In: DOI: 10.17487/rfc7805.

Appendix A

File System Structure

```
files
├── Blueprints
│   ├── Butterworth
│   │   └── __init__.py
│   └── Event_trigger
│       └── __init__.py
├── Datasources
│   └── Rig
├── Fmus
│   └── testrig.fmu
├── Fmu_models
│   └── testrig
│       └── Arm.json
├── Processors
│   ├── e_trigger_54543.json
│   └── fft_acceleration.json
└── Projects
    └── DemoProject
        ├── sensor_dataset.csv
        └── data_log.xlsx
```

Figure A.1: Tree

Appendix B

Implementation

B.1 External libraries

Library	**Purpose**
ReactJS	A JavaScript library for building user interfaces
TypeScript	A typed superset of JavaScript that compiles to plain JavaScript for easy debugging and documentation
Firebase	Google Firestore for real-time database updates and Firebase Authentication for authentication
Ceetron	For advanced 3D visualization of CAE, CFD and FEA
Zustand	Bear necessities for global state management in React
styled-components	Use the best bits of ES6 and CSS to style your apps without stress
react-router-dom	Declarative routing for React
react-beautiful-dnd	Beautiful and accessible drag and drop for lists with React
react-plotly	An interactive graphing library
leaflet	JavaScript library for interactive maps
react-leaflet	JavaScript library for interactive maps
Sveve	Simple SMS API
TensorFlowJS	Machine learning for JavaScript developers
stats-lite	A fairly light statistical package
xlsx	Parser and writer for various spreadsheet formats
file-saver	Saving files client-side
HTML2Canvas	JavaScript HTML renderer
jsPDF	PDF Document creation from JavaScript
react-grid-layout	A draggable and resizable grid layout with responsive breakpoints, for React
react-icons	Popular icons with ES6 imports
@aksel/structjs	Python-style struct module in JavaScript for parsing real-time data from the back-end

Table B.1: List of external libraries used in the front-end development

Library	**Purpose**
multiprocessing	Side-steps the Python GIL
mpld3	convert Python matplotlib code to JavaScript
fmpy	Interact with the virtual model
aiohttp	Facilitate HTTP communication
aiohttp_session[secure]	Allows storage of user-specific data into a session object
aiokafka	Facilitates use of Kafka
aiohttp_cors	implements Cross Origin Resource Sharing (CORS) support for aiohttp
numpy	scientific computing with Python
scipy	Ecosystem of open-source software for mathematics, science, and engineering
firebase-admin	Firebase Admin SDK for Python
pandas-profiling[notebook,html]	Tool to create HTML profiling reports
pandas	open source data analysis and manipulation tool
matplotlib	creating static, animated, and interactive visualizations

Table B.2: List of external libraries used in the back-end

B.2 Code Listings

This appendix shows more detailed code from the Implementation section. For more detailts, see the complete code repository. The first section shows code snippets from the front-end solution, while the second section shows code snippets from the back-end solution.

B.2.1 Front-end

A component that wants to access data from a store imports the store and declares the specific content it needs, as shown in Listing B.1.

```
import useProfileStore from "src/stores/profile/profileStore";

const { createProfile } = useProfileStore((state) => ({
    createProfile: state.createProfile,
  }));
```

Code lising B.1: A component imports the `profileStore` and the method `createProfile` from the `profileStore`.

Listing B.2 shows the code for creating *Draggable* elemnts that can be dragged inside a *Droppable* area.

```
<Droppable droppableId="selected">
  {(provided: any, snapshot: any) => (
    <S.ColumnRight
      ref={provided.innerRef}
      isDraggingOver={snapshot.isDraggingOver}
    >
      {sensors.length === 0 && <div>Drop your sensors here</div>}
      {sensors.map((sensor: string, i: number) => (
        <Draggable key={sensor} draggableId={sensor} index={i}>
          {(provided, snapshot) => (
            <S.Selected
              ref={provided.innerRef}
              {...provided.draggableProps}
              {...provided.dragHandleProps}
              isDragging={snapshot.isDragging}
              onClick={() => {
                setIndex(sensor);
                setTimeIndex(sensors.indexOf(sensor));
              }}
            >
              <div>{sensor}</div>
              {sensor === index && (
                <div>
                  <IoMdTimer
                    color="black"
                    size="1.2em"
                    style={{ padding: "1px" }}
                  />
                </div>
              )}
            </S.Selected>
          )}
        </Draggable>
      ))}
      {provided.placeholder}
    </S.ColumnRight>
  )}
</Droppable>
```

Code lising B.2: Droppable elements: one for the right and one for the left column

Listing B.3 shows the complete process of parsing raw binary data in the front-end.

```
parseData: (data: any, sourceID: string) => {
  const datasourceBuffer = get().datasourcesBuffer[sourceID];

    const unpacker = datasourceBuffer.unpacker;
    const unpackIterator = unpacker.iter_unpack(data);
    let unpacked = unpackIterator.next().value;
    while (unpacked) {
      if (get().datasources[sourceID] && get().datasources[sourceID].channels) {
        datasourceBuffer.timestamp_buffer.push(new Date(unpacked[0] * 1000));
        const tempChannelsIds = get().datasources[sourceID].channels.map(
          (it: any) => it.id
        );
        const channelsIds = Array.from(new Set(tempChannelsIds));
        channelsIds.forEach((channelID: any) => {
          if (
            datasourceBuffer.value_buffer &&
            datasourceBuffer.value_buffer[channelID]
          ) {
            datasourceBuffer.value_buffer[channelID].push(
              unpacked[channelID + 1]
            );
          }
        });
        unpacked = unpackIterator.next().value;
      }
    }
}
```

Code lising B.3: Parsing data in the front-end

Listing B.4 shows the function for uploading a file.

```
const uploadFile = (e: any) => {
  setLoading(true);
  const createLink = rootAPI + "/project/datafile";
  e.preventDefault();

  let file = e.target.files[0];
  const formData = new FormData();
  formData.append("file", file);
  if (type) {
    formData.append("type", type);
  } else {
    formData.append("type", "none");
  }
  axios
    .post(createLink, formData, {
      headers: {
        projectName: projectId,
        fileName: file.name.split(".")[0],
        fileType: file.name.split(".")[1],
      },
      onUploadProgress: (progressEvent: any) => {
        setUploadFilePercentage(
          Math.round((progressEvent.loaded * 100) / progressEvent
              .total)
        );
      },
    });
};
```

Code lising B.4: uploadFile function that sends a file to the server

Table B.3 shows the variables used to train the machine learning model in the client.

Variable	**Value**	**Explanation**
test_train_split	0.2	The ration used for splitting the dataset into a training dataset and a test dataset. The value is set to 0.2, which means 20% of the dataset is used for testing.
activation	relu	The activation function is set to "ReLu"; Rectified Linear Unit.
learningRate	0.01	The learning rate is used to control how quickly the model is adapted to the problem.
epochs	10	Epochs are the number of times all of the training data is used once for updating the weights.
optimizer	tf.train.adam(0.01)	Optimizers are used to change the attributes of the neural network to reduce the lessos. This can for example be the weights or the learning rate.
loss	meanSquaredError	Mean squared error is used as the loss function to minimize the error of the model.
min_R2_score	0.5	The minimum required R2 score of the model. The R2 score is a measure on how close the data is to the fitted model.
decent_R2_score	0.8	The satisfying R2 score of the model. The R2 score is a measure on how close the data is to the fitted model.
max_mean_diff	100	The maximum differences in mean between input parameters. This is used to determine if the data should be standardized.
max_std_diff	10	The maximum differences in standard deviation between input parameters. This is used to determine if the data should be standardized.
cov_limit	0.9	The maximum allowed covariance between variables.
max_iterations	4	The maximum number of iterations the model is trying to be trained.

Table B.3: Description of the variables used in the machine learning model

Listings B.5 and B.6 show how statistics are calculated. Listing B.5 uses historical data received from the server to illustrate distribution of the data in a histogram, whereas Listing B.6 calculates statistical values.

```
 getJSONResponse(link).then((response) => {
      const headers = response[0];
      if (response && response[2]) {
        const tempData = response[2].map((data: any, index: number)
             => {
          const histogram = (stats.histogram(data, 5) as unknown)
              as Histogram;
          let testingArray = [] as any[];
          testingArray.push({
            histfunc: "sum",
            y: histogram.values,
            x: histogram.values.map((value: number, index: number)
                => {
              return index + 1 === histogram.binWidth
                ? histogram.binLimits[1]
                : histogram.binLimits[0] + histogram.binWidth *
                    index;
            }),
            type: "histogram",
            name: headers[index + 1],
          });
          return testingArray;
        });
        setStatData(tempData);
      }
    });
```

Code lising B.5: Generating histogram from file

```
getJSONResponse(link).then((response) => {
  const headers = response[0];
  if (response && response[2]) {
    const tempData = response[2].map((datapoints: any, index:
        number) => ({
      name: headers[index + 1],
      mean: stats.mean(datapoints).toFixed(2),
      stdDev: stats.stdev(datapoints).toFixed(2),
      median: stats.median(datapoints).toFixed(2),
      variance: stats.variance(datapoints).toFixed(2),
      percentile85: stats.percentile(datapoints, 0.85).toFixed(2),
    }));
    setStatData(tempData);
    setLoading(false);
  }
});
```

Code lising B.6: Generating statistical information from a file

B.2.2 Back-end

```
@routes.post('/project/datafile', name='upload_datafile')
async def upload_datafile(request: web.Request):
    post_request = await request.post()
    data, headers = post_request['file'] = request.headers
    project = headers["projectName"]
    content_length = int(headers['Content-length'])
    path = request.app['settings'].PROJECT_DIR + "/" + project + "/
        files"
    os.makedirs(path, exist_ok=True)

    if ".csv" in data.filename or ".xlsx" in data.filename:
        with open(path + data.filename, 'wb') as file:
            file.write(data.file.read(content_length))
        return web.HTTPAccepted()
    else:
        return web.HTTPBadRequest()
```

Code lising B.7: `upload_datafile` receives a request that contains a file and writes it to the file system in `project_name/files/data.filename`

Listing B.8 shows the complete process of receiving a parsing JSON data in the back-end.

```
            if address in self.buffers:
                if "{" in raw_data.decode():
                    # print("this { is here")
                    if "}" in self.buffers[address]:
                        self.buffers[address] = raw_data.decode() +
                             self.buffers[address]
                    else:
                        self.buffers[address] = raw_data.decode() +
                             self.buffers[address]
                elif "}" in raw_data.decode():
                    if "{" in self.buffers[address]:
                        self.buffers[address] = self.buffers[
                            address] + raw_data.decode()
                    else:
                        self.buffers = self.buffers[address] +
                            raw_data.decode()
                else:
                    if "{" in self.buffers[address]:
                        self.buffers[address] = self.buffers[address] + raw_data.decode()
                    elif "}" in self.buffers[address]:
                        self.buffers[address] = raw_data.decode() + self.buffers[address]
                    else:
                        self.buffers = self.buffers[address] + raw_data.decode()
            else:
                print("empty")
                self.buffers[address] = "" + raw_data.decode()
            if len(self.buffers[address]) > 0 and \
                    self.buffers[address].count("{") == self.buffers[address].count("}"):
                try:
                    json_data = json.loads(self.buffers[address])
                    data_values = source.output_names
                    incoming_data = []
                    for (index, value) in enumerate(data_values):
                        # print(value, index)
                        if index == source.time_index:
                            incoming_data.append(
                                datetime.timestamp(parse(json_data[data_values[source.time_index]])))
                        else:
                            incoming_data.append(float(json_data[value]))

                    data = struct.pack(
                        source.byte_format, incoming_data[source.time_index],
                        *[incoming_data[ref] for ref in source.output_refs])
                    self.producer.send(topic=source.topic, value=data)
                    self.buffers[address] = ""
                except ValueError:
                    self.buffers[address] = ""
                    print("failed to parse")
```

Code lising B.8: Buffering JSON data

Listing B.9 shows the process of creating a dictionary for viewing available datasources sending JSON data.

```
try:
    if (address + "_data") in self._available_sources:
        self._available_sources[address + "_data"] =
        self._available_sources[address + "_data"] + raw_data.
            decode()
    else:
        self._available_sources[address + "_data"] = raw_data.
            decode()
    if (address + "_data") in self._available_sources and (
            self._available_sources[address + "_data"].count("{")
                == self._available_sources[address + "_data"].count
                ("}")):
        sensors_data = json.loads(self._available_sources[address +
             "_data"])
        sensors = list(sensors_data.keys())
        self._available_sources[address] = {
            "sensors": sensors,
        }
        del self._available_sources[address + "_data"]
        /* more code */
```

Code lising B.9: `self._available_sources` is a dictionary containing available data sources.

Listing B.10 shows how the back-end processes a request and returns a response for creating a new project.

```
@routes.post('/projects/new', name='create_new_project')
async def create_new_project(request: web.Request):
    post = await request.post()
    name = try_get(post, 'projectName', str)
    projectAlreadyExists = await database.check_if_project_exists(
        name)
    if projectAlreadyExists:
        raise web.HTTPBadRequest(text='Project name already exists'
            )
    date, email = try_get(post, 'date', str), try_get(post, 'email'
        , str)
    created = await database.create_project(email, name, date)
    if created:
        return web.HTTPCreated()
    raise web.HTTPBadRequest()
```

Code lising B.10: Receiving the request in the back-end. `try_get` is a helper method used to retrieve relevant attributes from the request

At first, the sensor is below its allowed limits and a notification is shown to the user that the sensor is not in its normal state. When the sensor returns to the normal state, another event is triggered and a notification is sent to the user that the sensor has returned to the normal state. This process is shown in Listing B.11.

```python
if (value > minVal and (sensor in self.trigger) and ("startedAt" in
     self.trigger[sensor]) and (
        self.trigger[sensor]["trigger_reason"] == "lt")):
    newId = self.trigger[sensor]["id"]
    updated = database.update_notification(self.project_id, newId,
        {
        u'finished': True,
        u'endedAt': datetime.datetime.now(pytz.timezone('Europe/
            Oslo')),
        u'valueExceeded': minVal
    })
    del self.trigger[sensor]
```

Code lising B.11: The value has returned to the normal state and the trigger is removed from the dictionary and its state is set to finished in the database

Appendix C

Latency

This appendix shows more detailed graphs from the latency tests. The first section contains latency calculations from plotting one datasource at a time, then the second and third sections show impact multiple datasources simultaneously and of filtering data source with buffers of different size. The red lines represent total delay, divided into delay from asset to front-end (pink) and internal front-end delay (purple).

C.1 Assets alone over different networks

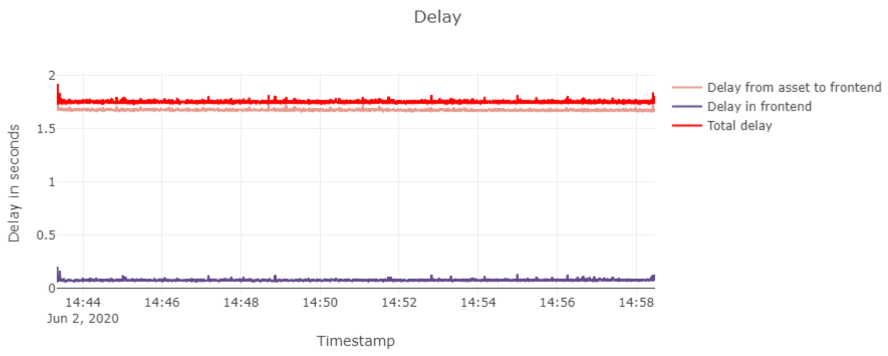


Figure C.1: Delay from the Torsion Bar Suspension Rig over ethernet connection

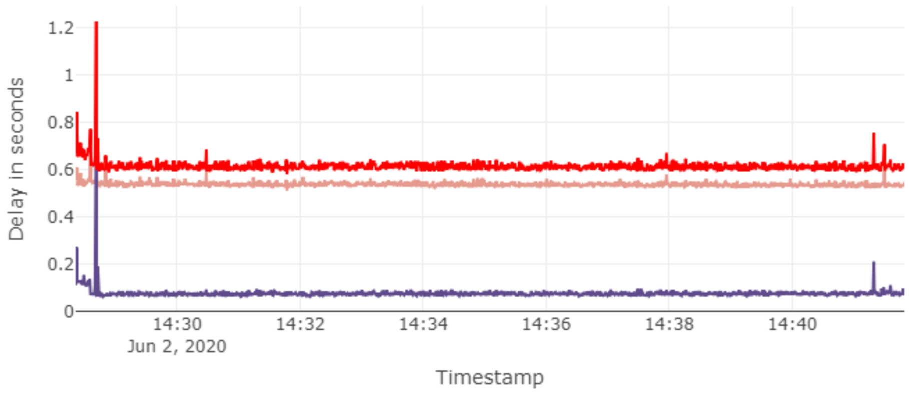


Figure C.2: Delay from the SensorLog application over 4G network

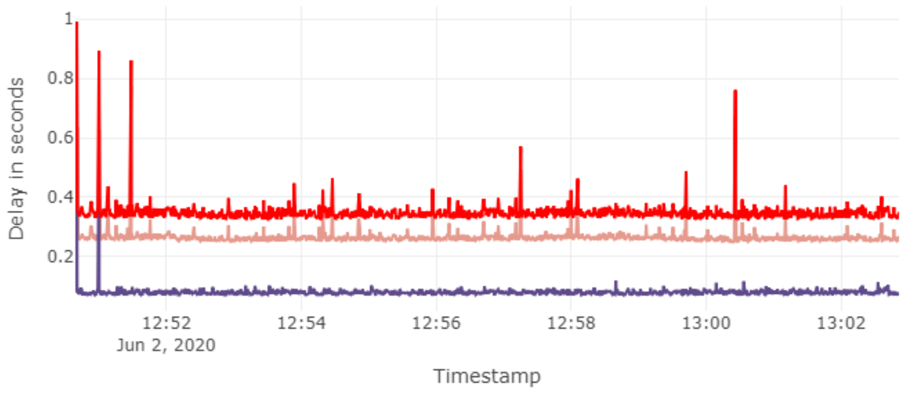


Figure C.3: Delay from the SensorLog application over Wi-Fi connection

C.2 Effect of multiple tiles

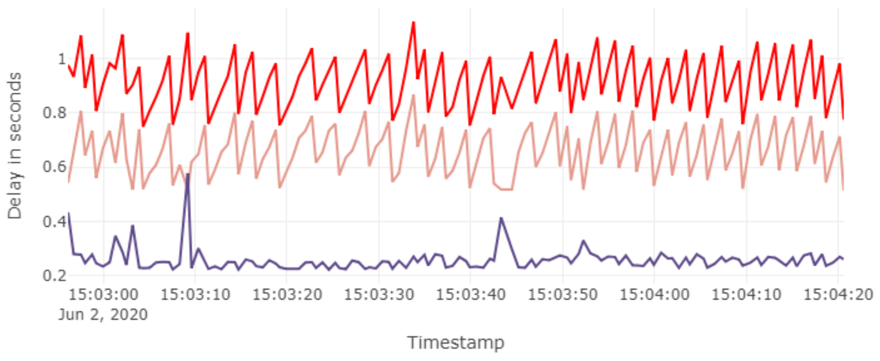


Figure C.4: Delay in transmission from the SensorLog application over Wi-Fi when it is the second to render in a dashboard

C.3 Filtered data

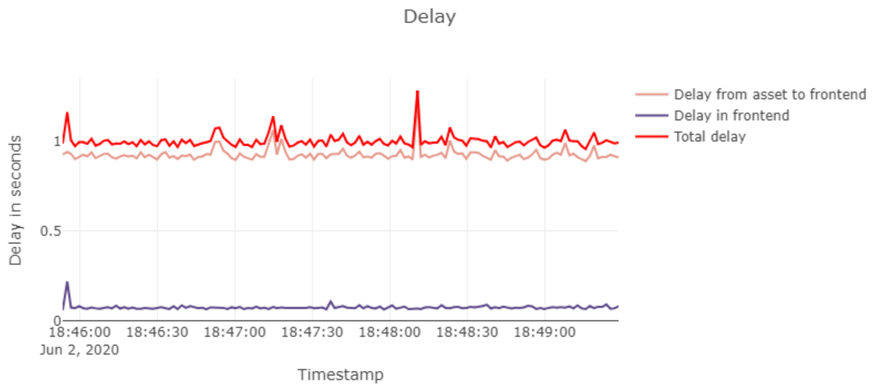


Figure C.5: Delay in transmission from the SensorLog application over Wi-Fi with filtered a sensor and a buffer size of 1

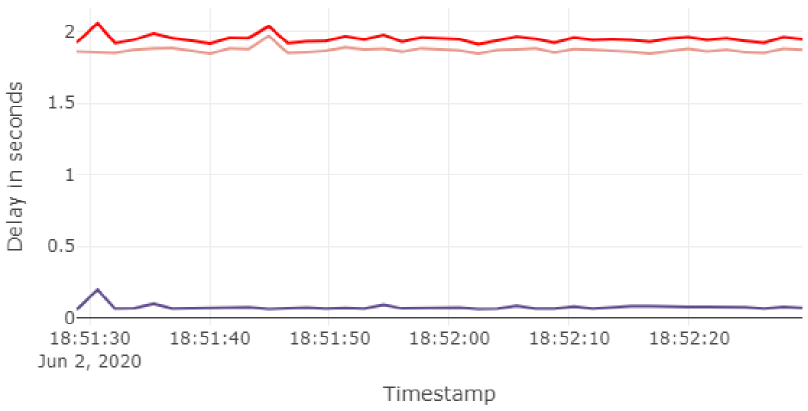


Figure C.6: Delay in transmission from the SensorLog application over Wi-Fi with filtered a sensor and a buffer size of 20

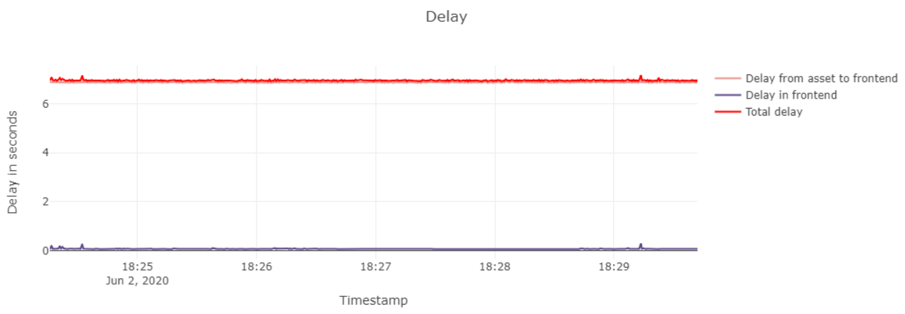
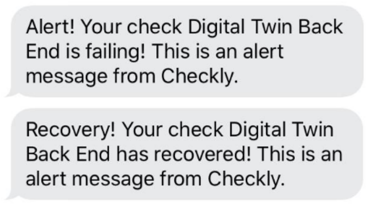


Figure C.7: Delay in transmission from the SensorLog application over Wi-Fi with filtered a sensor and a buffer size of 500



Alert! Your check Digital Twin Back End is failing! This is an alert message from Checkly.

Recovery! Your check Digital Twin Back End has recovered! This is an alert message from Checkly.

Figure C.8: Checkly automatically sends out SMS if the client or back-end solution is failing.

Appendix D

Usability testing

- The website is easy to use.
- It is easy to navigate within the website.
- I feel comfortable purchasing from the website.
- I feel confident conducting business on the website.
- How likely are you to recommend this website to a friend or colleague?
- I will likely return to the website in the future.
- I find the website to be attractive.
- The website has a clean and simple presentation.

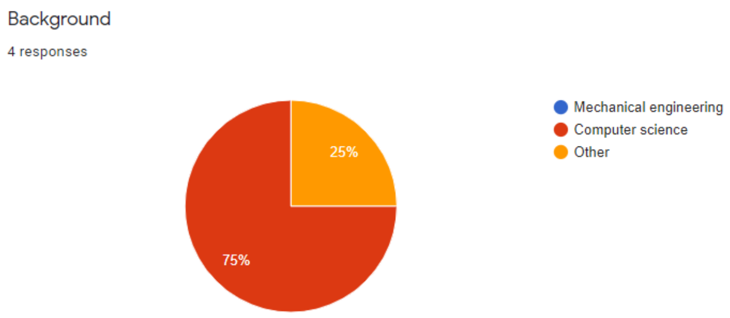


Figure D.1: Background.

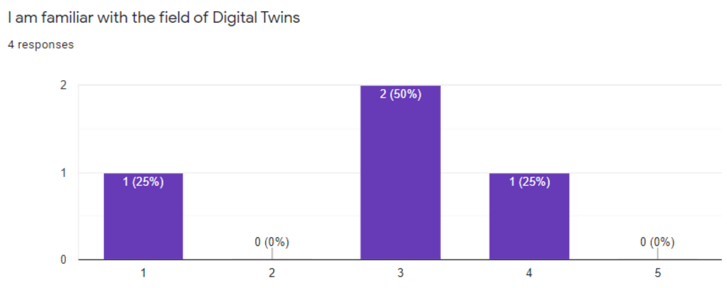


Figure D.2: I am familiar with the field of Digital Twins.

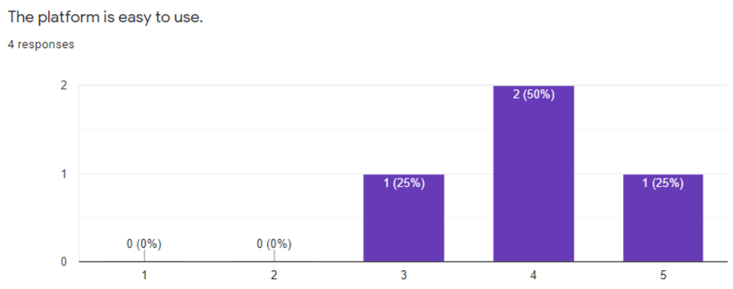


Figure D.3: The platform is easy to use.

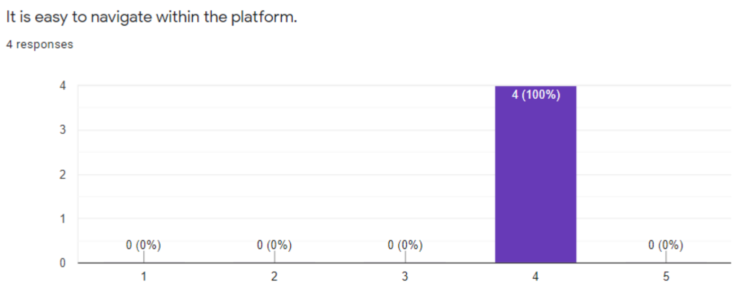


Figure D.4: It is easy to navigate within the platform.

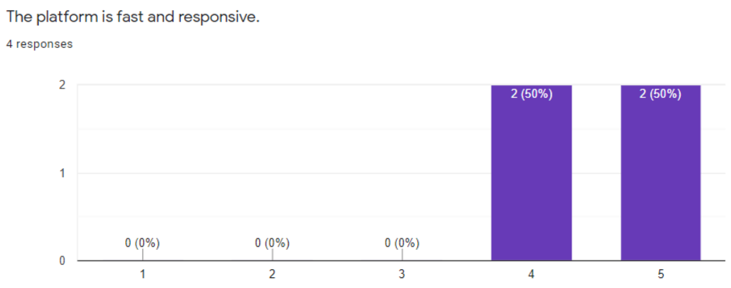


Figure D.5: The platform is fast and responsive.

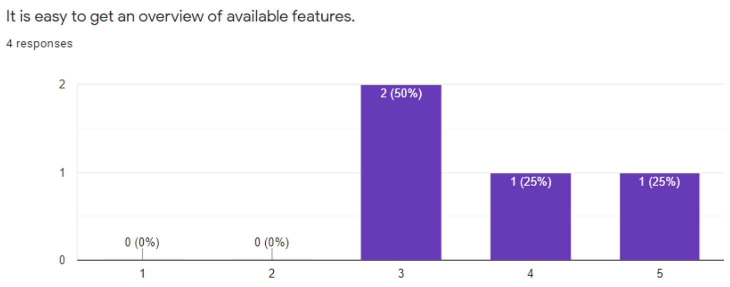


Figure D.6: It is easy to get an overview of available features.

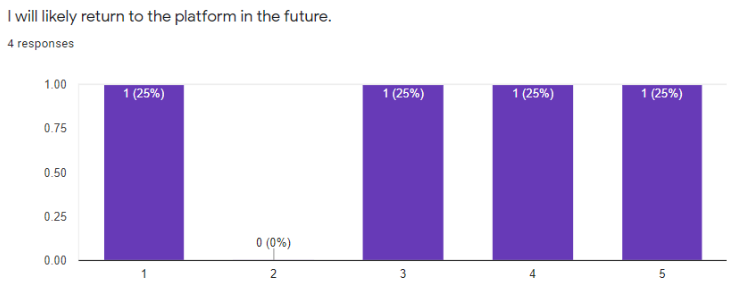


Figure D.7: I will likely return to the platform in the future.

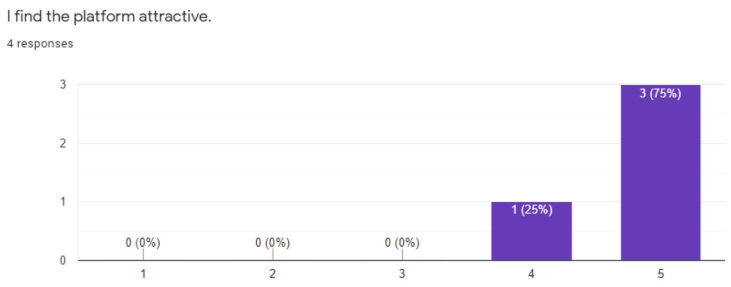


Figure D.8: I find the platform attractive.

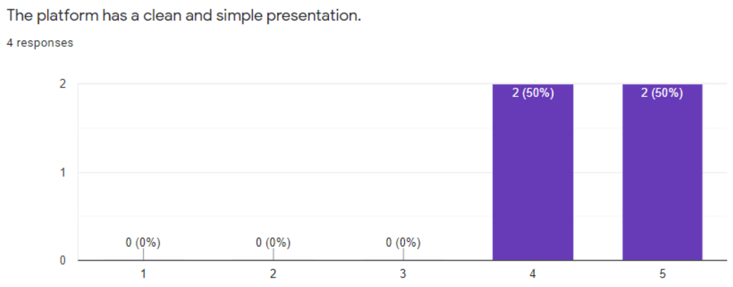


Figure D.9: The platform has a clean and simple presentation.

Appendix E

User Guides

E.1 Starting a session

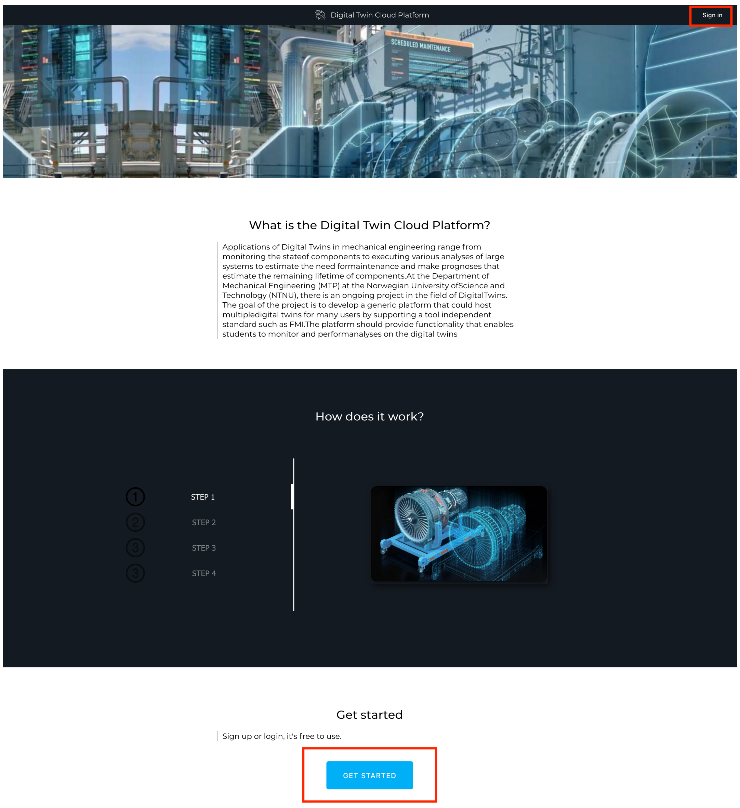
Connect to NTNU's network either via wifi or VPN [1]. Then, go to `cbms.surge.sh/` in Google Chrome, Mozilla Firefox, Opera, Microsoft Edge or Safari. The landing page will render.

E.2 Register User

There are two ways to navigate to user registration: From the landing page, click on either *Sign in* at the top right corner, or the *GET STARTED* button at the bottom of the page marked in red in Figure E.1.

This takes you to the sign in. Click on "Don't have an account? Click here to sign up" marked in red in Figure E.2a. The register page will load. The e-mail needs to be valid and the password must consist of at least six characters. Fill in the fields in E.2b, and a user with these credentials will be created. When the registration process has completed, you will be directed to the projects page as a signed in user.

[1] https://innsida.ntnu.no/wiki/-/wiki/English/Install+vpn



What is the Digital Twin Cloud Platform?

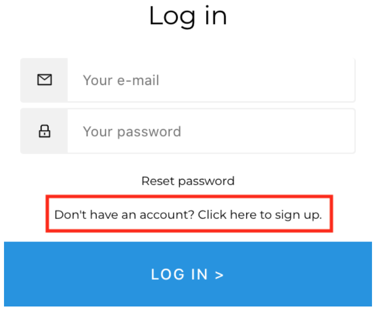
Applications of Digital Twins in mechanical engineering range from monitoring the stateof components to executing various analyses of large systems to estimate the need formaintenance and make prognoses that estimate the remaining lifetime of components.At the Department of Mechanical Engineering (MTP) at the Norwegian University ofScience and Technology (NTNU), there is an ongoing project in the field of DigitalTwins. The goal of the project is to develop a generic platform that could host multipledigital twins for many users by supporting a tool independent standard such as FMI.The platform should provide functionality that enables students to monitor and performanalyses on the digital twins

How does it work?

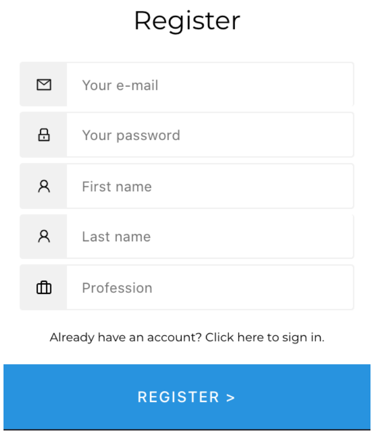
Get started

Sign up or login, it's free to use.

Figure E.1: Illustration of the landing page with highlighted areas.



(a) Sign in page

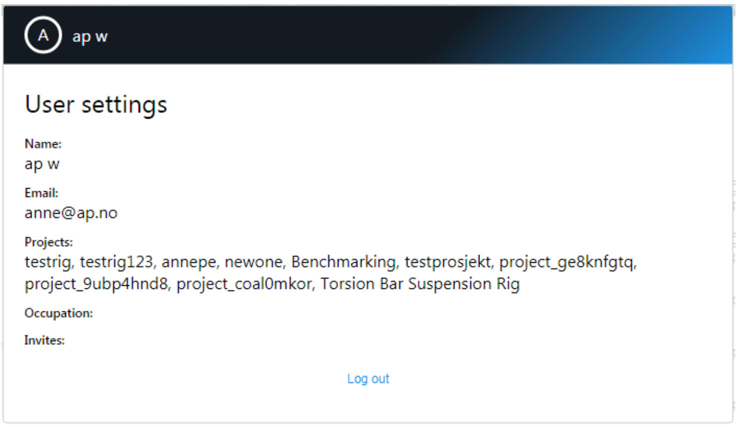


(b) Register user

Figure E.2: Screenshots illustrating how get to register page from sign in and the information to be filled in to register.

E.2.1 View profile settings

Whenever you are signed in, your username and a user icon will be in the top right corner at all times. Clicking on either of the user name and icon opens a window with information about your profile such as name and e-mail address as seen in Figure E.3.



User settings

Name:
ap w

Email:
anne@ap.no

Projects:
testrig, testrig123, annepe, newone, Benchmarking, testprosjekt, project_ge8knfgtq, project_9ubp4hnd8, project_coal0mkor, Torsion Bar Suspension Rig

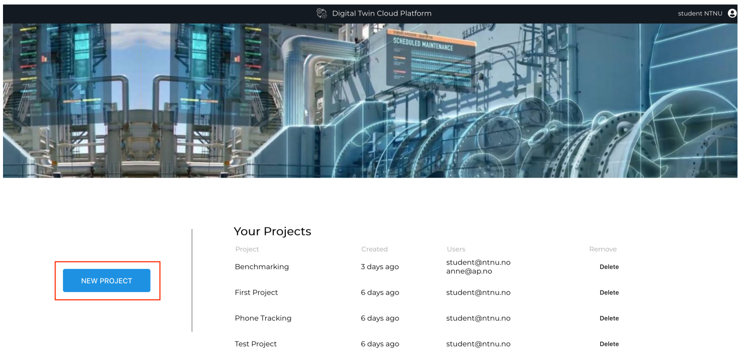
Occupation:

Invites:

Figure E.3: Settings for a profile showing name, e-mail, existing projects, occupation and invites

E.3 Configure a project

In order to configure a project, you need to be signed in. From the projects page, click on the *NEW PROJECT* button to the left located on the middle of the page as seen in Figure E.4.

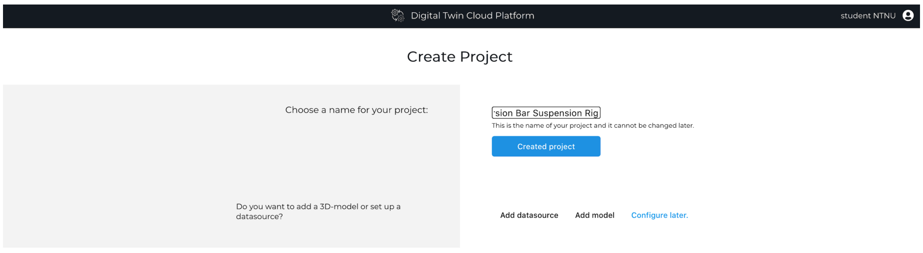


Your Projects

Project	Created	Users	Remove
Benchmarking	3 days ago	student@ntnu.no anne@ap.no	Delete
First Project	6 days ago	student@ntnu.no	Delete
Phone Tracking	6 days ago	student@ntnu.no	Delete
Test Project	6 days ago	student@ntnu.no	Delete

Figure E.4: Projects page: click the button framed in red to start configuration of a new project

You will be taken to the New Project page. The first and only required step is to give the project a name. Once ”Create Project” is clicked, the project is saved. However, it is possible add a datasource and/or a model before advancing to your new project. The buttons ”Add datasource”, ”Add model” and ”Configure later” as seen in Figure E.5 show the options, and clicking on them will render the configuration page. In Figure E.6, the ”Add datasource” button has been selected.

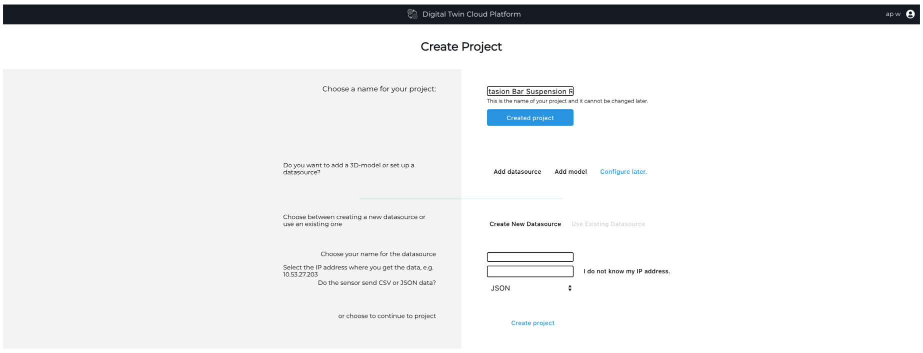


Choose a name for your project: sion Bar Suspension Rig

This is the name of your project and it cannot be changed later.

Do you want to add a 3D-model or set up a datasource?

Figure E.5: Options for further configuration after registering project name



Digital Twin Cloud Platform ap w

Create Project

Choose a name for your project: tasion Bar Suspension R
This is the name of your project and it cannot be changed later.
Created project

Do you want to add a 3D-model or set up a datasource? Add datasource | Add model | Configure later.

Choose between creating a new datasource or use an existing one: Create New Datasource | Use Existing Datasource

Choose your name for the datasource
Select the IP address where you get the data, e.g. 10.53.27.203 — I do not know my IP address.
Do the sensor send CSV or JSON data? JSON

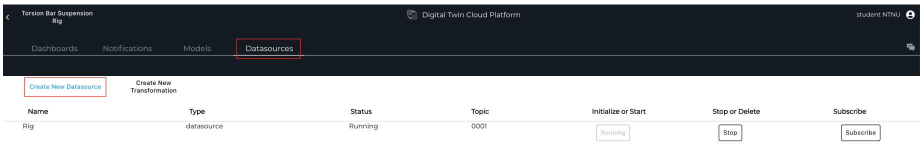
or choose to continue to project: Create project

Figure E.6: Adding a datasource to the new project

It is possible to add a new or existing datasource to the project. To configure a new datasource, please consult the user guide on how to configure a datasource (REF). If you want to add an existing datasource, a dropdown list shows the existing sources, and selecting one connects it to the project. The process of adding a model is described in REF.

E.4 Configure a datasource

A datasource can be created during configuration of a project as described in Appendix E.3 or from the Datasource tab in the Project page as seen in Figure E.7.



Torsion Bar Suspension Rig — Digital Twin Cloud Platform — student NTNU

Dashboards | Notifications | Models | Datasources

Create New Datasource | Create New Transformation

Name	Type	Status	Topic	Initialize or Start	Stop or Delete	Subscribe
Rig	datasource	Running	0001	Running	Stop	Subscribe

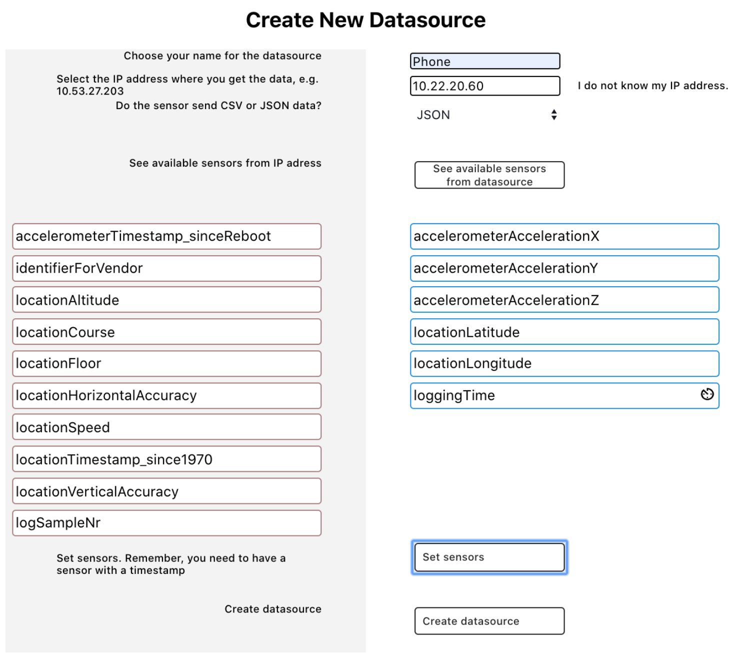
Figure E.7: To start the configuration process for a datasource, first navigate to the datasource page by clicking on Datasources in the navigation bar and then on the create new datasource on the top left

Now the create datasource page pops up. The platform accepts data on CSV or JSON format. Both configurations reruire IP address and definition of sensor names, but the process is not identical. The datasource name can only contain letters, numbers and underscore, and it has to start with a letter.

E.4.1 Configure JSON datasource

After selecting a name for the datasource, the IP address of the data stream must be filled in. If you do not know the IP address of your source, click on "I don't know my IP address", and a list of IP addresses sending JSON data to the server shows up. Copy the address and paste it in the IP address field.

Now, click on "See available sensors from datasource", and the sensors will appear in the gray column to the left. Drag and drop or click on sensors you want to use. NB! One of the sensors must contain a timestamp. Click on this sensor again when it is selected so that a clock icon shows up on the right end as in Figure E.8. Finally, click "Set sensors" and then "create datasource". The datasource is created and the configuration page is closed. Back in the Datasources tab, you should now see your new source as in Figure E.9.



Create New Datasource

Choose your name for the datasource	Phone
Select the IP address where you get the data, e.g. 10.53.27.203	10.22.20.60
Do the sensor send CSV or JSON data?	JSON

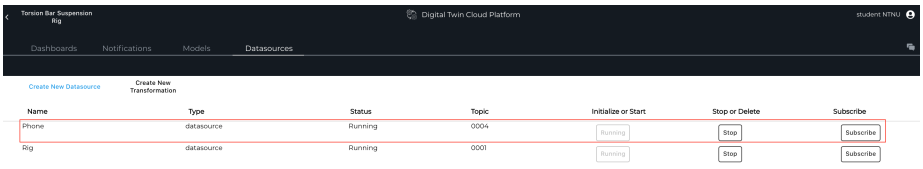
accelerometerTimestamp_sinceReboot
identifierForVendor
locationAltitude
locationCourse
locationFloor
locationHorizontalAccuracy
locationSpeed
locationTimestamp_since1970
locationVerticalAccuracy
logSampleNr

accelerometerAccelerationX
accelerometerAccelerationY
accelerometerAccelerationZ
locationLatitude
locationLongitude
loggingTime

Set sensors. Remember, you need to have a sensor with a timestamp

I do not know my IP address.
See available sensors from IP adress
See available sensors from datasource
Set sensors
Create datasource
Create datasource

Figure E.8: Completed configuration of a JSON formatted datasource



Name	Type	Status	Topic	Initialize or Start	Stop or Delete	Subscribe
Phone	datasource	Running	0004	Running	Stop	Subscribe
Rig	datasource	Running	0001	Running	Stop	Subscribe

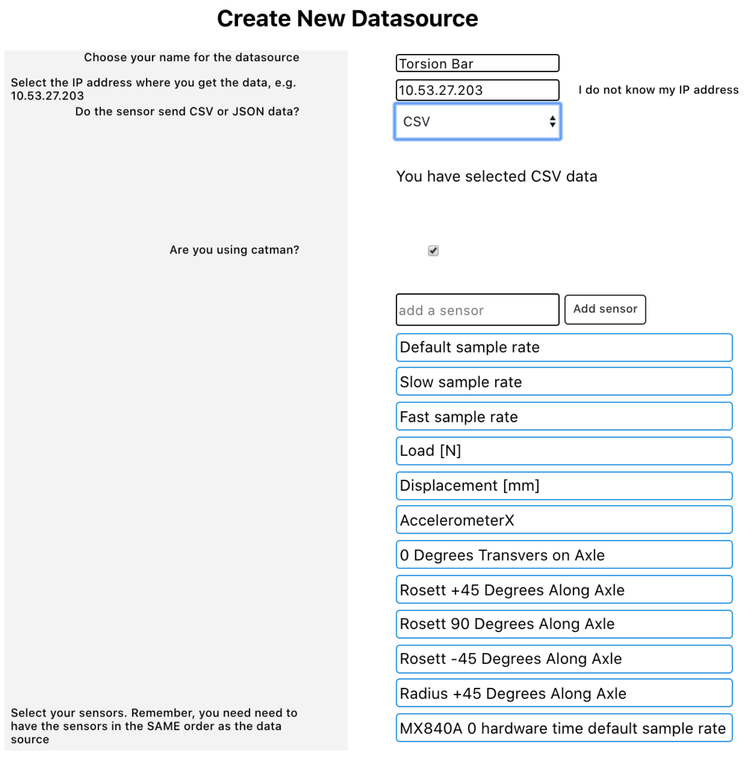
Torsion Bar Suspension Rig
Digital Twin Cloud Platform
student NTNU
Dashboards
Notifications
Models
Datasources
Create New Datasource
Create New Transformation

Figure E.9: When the configuration is finished and the create datasource window has closed, the new datasource should show up in the list of datasources, and it should be running

E.4.2 Configure CSV datasource

Fill in name and IP address for your source and select "CSV" from the dropdown menu. If you are not using catman, untick the checkbox and select the bytelength of each data point. The default is eight if catman is selected

Now, fill in names of all the sensor values in **correct order**. If the names don't match the format the data is sent on, the names will not correspond to the correct value. Figure E.10 shows configuration of the Torsion Bar Suspension Rig used for prototyping and development. After setting sources and clicking on "create datasource", the window should close and the new source should show up in the list of datasources as in Figure E.11.



Create New Datasource

Choose your name for the datasource: Torsion Bar

Select the IP address where you get the data, e.g. 10.53.27.203: 10.53.27.203

I do not know my IP address

Do the sensor send CSV or JSON data? CSV

You have selected CSV data

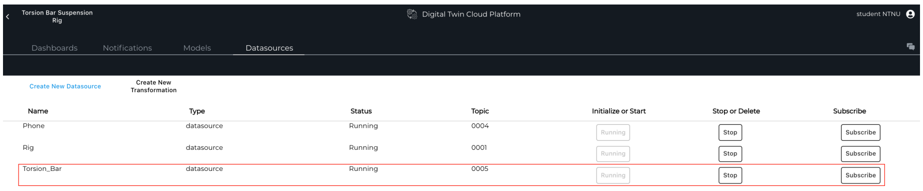
Are you using catman? ✓

add a sensor | Add sensor

Default sample rate
Slow sample rate
Fast sample rate
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

Select your sensors. Remember, you need need to have the sensors in the SAME order as the data source

Figure E.10: Configuration of a datasource on CSV format.

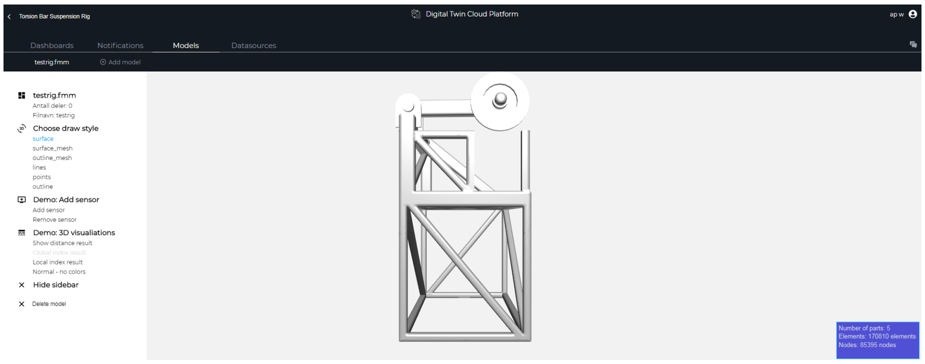

Torsion Bar Suspension Rig
Digital Twin Cloud Platform
student NTNU
Dashboards
Notifications
Models
Datasources
Create New Datasource
Create New Transformation

Name	Type	Status	Topic	Initialize or Start	Stop or Delete	Subscribe
Phone	datasource	Running	0004	Running	Stop	Subscribe
Rig	datasource	Running	0001	Running	Stop	Subscribe
Torsion_Bar	datasource	Running	0005	Running	Stop	Subscribe

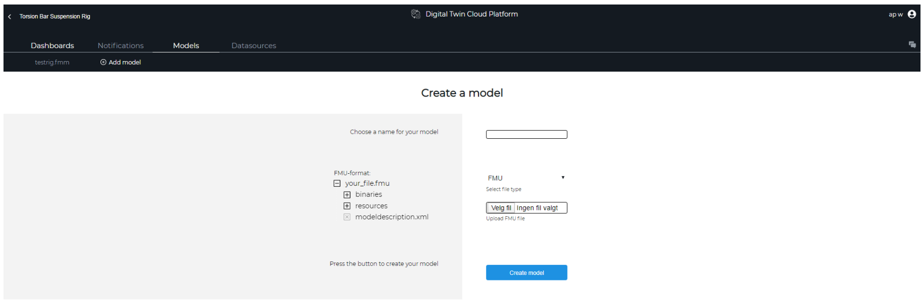
Figure E.11: List of datasources after creating a CSV formatted datasource

E.5 View and upload models

From the project page, click on the Models tab in the navigation bar. The page will look like Figure E.12a. To upload a new model, click on "Add Model" at the bottom of the navigation bar. Select file format, either "FMU" or"FMM". If FMU is the chosen format, select an FMU file from the file directory and wait until it has loaded. If you selected to upload an FMM file, first upload the FMM file. Then, upload corresponding FTL files. When all files have completed the upload, you will be redirected to the Models page.



(a) Models page



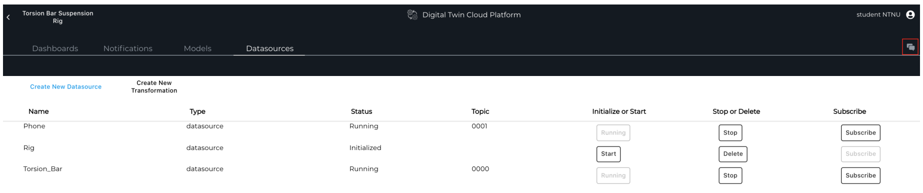
(b) New Model page

Figure E.12: The Models and New Model pages.

E.6 Invite User to Project and Chat

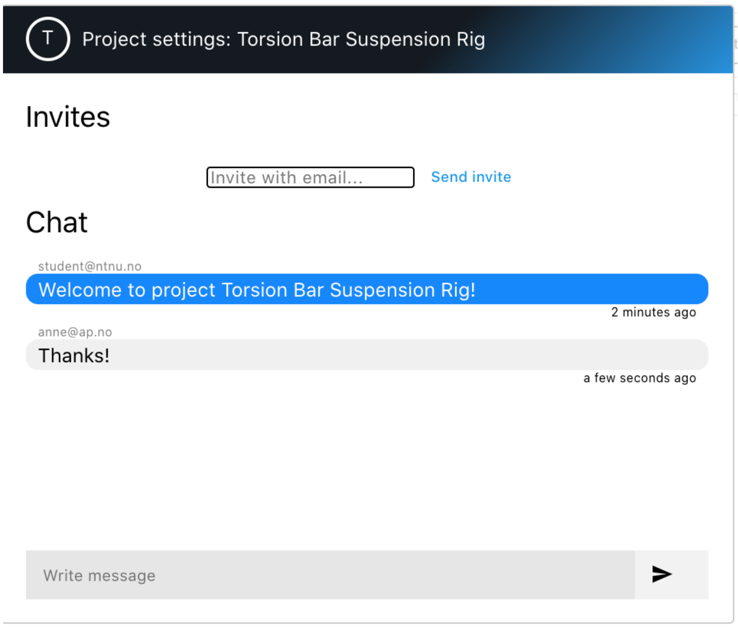
In the Project page, there is a speaking bubble icon to the right in the navigation bar, as seen in Figure E.13. Clicking on this symbol opens the window seen in Figure E.14.

To invite a user, fill in an e-mail address of an existing user and click "send invite". The user will receive a pop-up message as in Figure E.15. Accepting will add the project to the user's current projects, and "show invite" will take them to the projects page where the invitation is displayed. Ignore removes the pop-up, but the invitation is still available in the projects page.


Torsion Bar Suspension Rig
Digital Twin Cloud Platform
student NTNU
Dashboards
Notifications
Models
Datasources
Create New Datasource
Create New Transformation

Name	Type	Status	Topic	Initialize or Start	Stop or Delete	Subscribe
Phone	datasource	Running	0001	Running	Stop	Subscribe
Rig	datasource	Initialized		Start	Delete	Subscribe
Torsion_Bar	datasource	Running	0000	Running	Stop	Subscribe

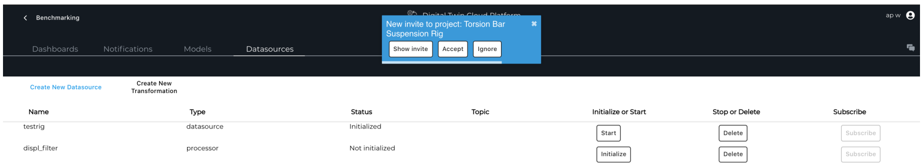
Figure E.13: Invite user



student@ntnu.no
Welcome to project Torsion Bar Suspension Rig!
2 minutes ago

anne@ap.no
Thanks!
a few seconds ago

Figure E.14: Invite and chat window



New invite to project: Torsion Bar Suspension Rig

Name	Type	Status	Topic	Initialize or Start	Stop or Delete	Subscribe
testrig	datasource	Initialized		Start	Delete	Subscribe
displ_filter	processor	Not initialized		Initialize	Delete	Subscribe

Figure E.15: A pop up is displayed when a user receives an invitation to a project.

E.7 Create a Tile

From the Dashboards tab in the navigation bar, navigate to the dashboard where you want to place the tile and click on "Add new", placed as shown in Figure E.16. A window will open showing different tile options, as in Figure E.17. By clicking on the main categories "Real-time monitoring", "Analytics and Statistics" and "3D models" and their respective sub categories, you can navigate between different tile types.

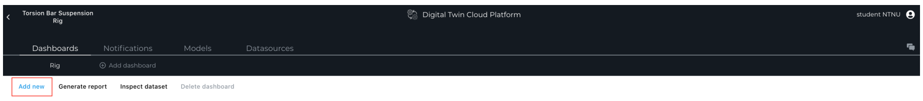


Figure E.16: Add new tile

All tiles require a name and some input data. The input data varies, but is generally either a datasource or a file (for Analytics and Statistics tiles and models). The files in the Analytics and Statistics category should be on a CSV

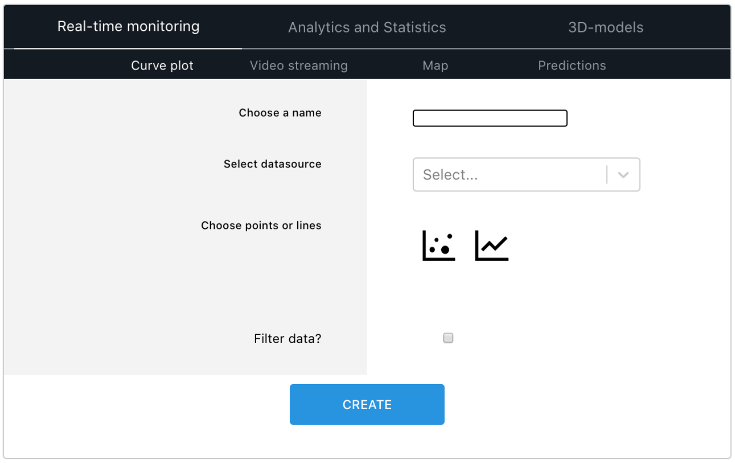
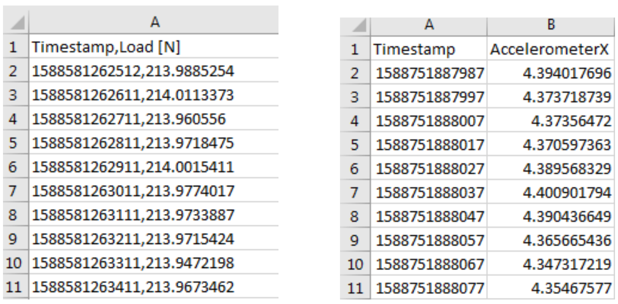


Figure E.17: The add new tile window: Showing the configuration of a Real-time curve plot

or XLSX format, and a timestamp is required as the first column as seen in figure E.18. The rest of the columns are interpreted as sensor data values. The first row should contain the column names.



	A
1	Timestamp,Load [N]
2	1588581262512,213.9885254
3	1588581262611,214.0113373
4	1588581262711,213.960556
5	1588581262811,213.9718475
6	1588581262911,214.0015411
7	1588581263011,213.9774017
8	1588581263111,213.9733887
9	1588581263211,213.9715424
10	1588581263311,213.9472198
11	1588581263411,213.9673462

(a) CSV file

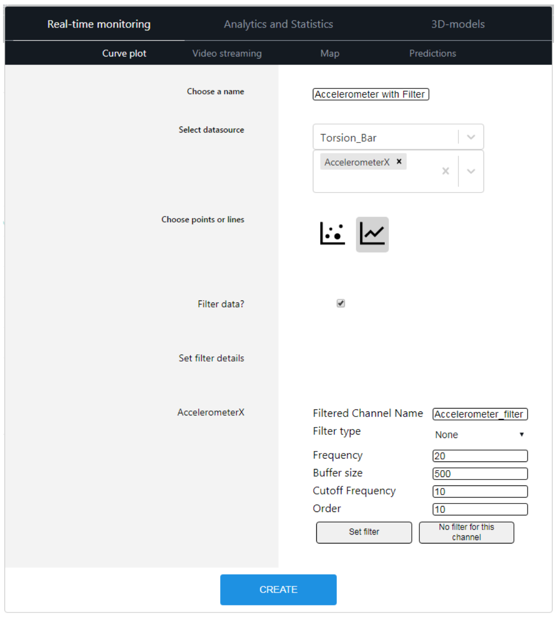
	A	B
1	Timestamp	AccelerometerX
2	1588751887987	4.394017696
3	1588751887997	4.373718739
4	1588751888007	4.37356472
5	1588751888017	4.370597363
6	1588751888027	4.389568329
7	1588751888037	4.400901794
8	1588751888047	4.390436649
9	1588751888057	4.365665436
10	1588751888067	4.347317219
11	1588751888077	4.35467577

(b) Excel file

Figure E.18: Expected format of CSV files and XLSX respectively

Real-Time Curve Plot The real-time curve plots take live data from a datasource as input and plots it in a graph. Figure E.17 shows its configuration. The user must first select a datasource and a sensor value from the selected datasource. Then, a plot type must be selected, either scatter or lines. If no processing of the signal is desired, click "CREATE", and the plot will be saved to the dashboard, the window closes and the new tile renders.

However, you want filtering, check the checkbox to the right of "Filter data?". Filtering options will appear for each sensor selected as in Figure E.19. If you don't want to filter one of the selected sensors, click "No filter for this channel" on the bottom right of the filtering options for that sensor. Otherwise, select filter type, frequency of the signal, buffer size, cutoff frequency and desired order of the filter. The possible filter types are lowpass, highpass, bandpass and bandstop. For bandpass and bandstop, two values must be submitted in the cutoff frequency separated by a comma. **NB!** The filtered sensor name must be different from the original sensor name, e.g. "sensorname_lowpass" for a lowpass filter. When values are selected, click on "set filter". A loading wheel will show up while the filter is generated. When it disappears, the filter is created and added to the plot.



Choose a name: Accelerometer with Filter

Select datasource: Torsion_Bar; AccelerometerX

Choose points or lines

Filter data? ✓

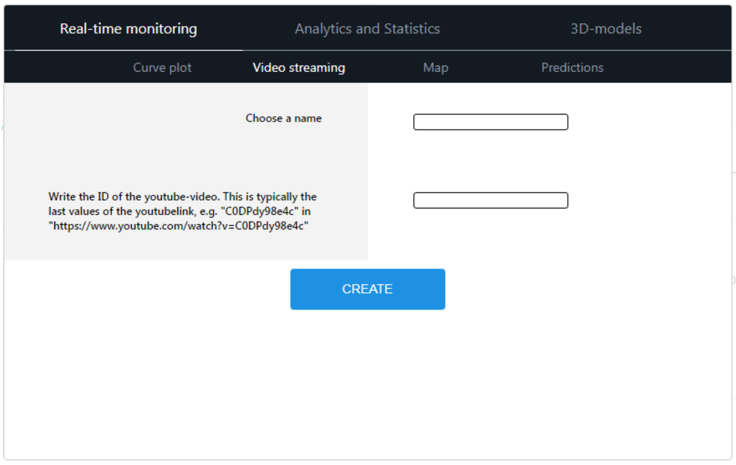
Set filter details

AccelerometerX

Filtered Channel Name	Accelerometer_filter
Filter type	None
Frequency	20
Buffer size	500
Cutoff Frequency	10
Order	10

Figure E.19: Filtering options

Video Streaming In addition to Tile name, configuration of a platform only requires ID of a youtube live stream. The ID of the stream is part of the url to the stream, as described in the left column in Figure E.20. A guide on how to create a live stream on youtube can be found on https://www.youtube.com/watch?v=Xka4OLgzW9Q.

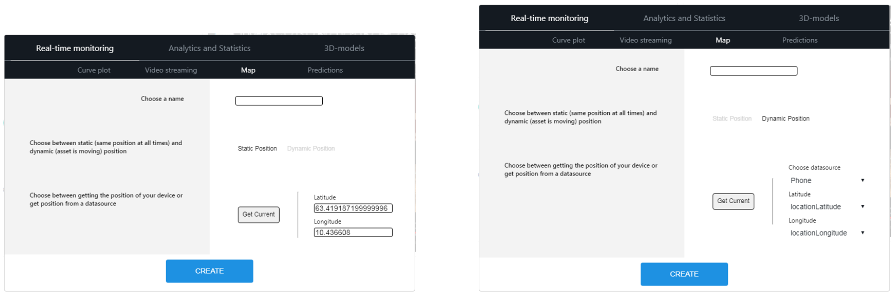


Choose a name

Write the ID of the youtube-video. This is typically the last values of the youtubelink, e.g. "C0DPdy98e4c" in "https://www.youtube.com/watch?v=C0DPdy98e4c"

Figure E.20: Configuration of a new video stream

Map Both static and dynamic maps can be configured. There are three possible input types: Both map types can use the location of the device that is logged into the system by selecting ”Get current”. The map will then place an icon of this location on the map and update it in the case of a dynamic map. A static map can also use a location specified by the user in the leftmost column seen in Figure E.21a. The last input type is to use a datasource's location sensors as seen in Figure E.21b.



Choose a name

Choose between static (same position at all times) and dynamic (asset is moving) position

Choose between getting the position of your device or get position from a datasource

Latitude: 63.419187199999996

Longitude: 10.436608

Choose a name

Choose between static (same position at all times) and dynamic (asset is moving) position

Choose between getting the position of your device or get position from a datasource

Choose datasource: Phone

Latitude: locationLatitude

Longitude: locationLongitude

(a) Static map configuration

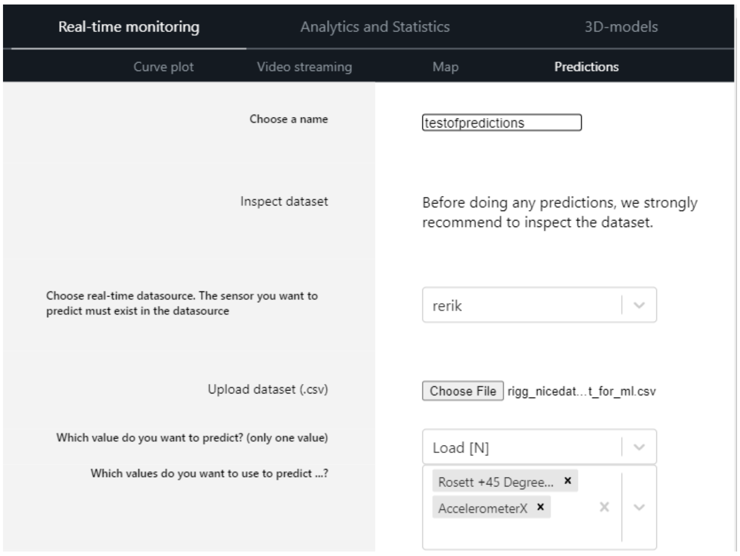
(b) Dynamic map configuration

Figure E.21: Adding new static and dynamic maps respectively

Predictions The user should always inspect the dataset for potential problems before it is used to train the machine learning. This can be achieved by using the *inspect dataset* functionality in the CBMS. The next step is selecting the datasource and the sensor the user wants to perform real-time predictions of as shown in Figure E.24. Then the user has to upload a data set to train the machine learning model. The data set must be a .csv file where decimal values are separated with "." as seen in Table E.1. The sensor names must correspond to the sensor names sent from the live data source. The user has to select the input sensors that should be used to predict the output. The user must have some knowledge about the data set in order to make predictions. However, the application in this project is developed for users that do not have any prior machine learning knowledge. The questions for describing the data set is therefore as general as possible.

Sensor 1	Sensor 2	...	Sensor n
4.54	7.54	...	787.12
8.14	2.56	...	987.16
5.92	9.81	...	12.32
...	...	...	...

Table E.1: Value format in CSV file



Choose a name: testofpredictions

Inspect dataset: Before doing any predictions, we strongly recommend to inspect the dataset.

Choose real-time datasource. The sensor you want to predict must exist in the datasource: rerik

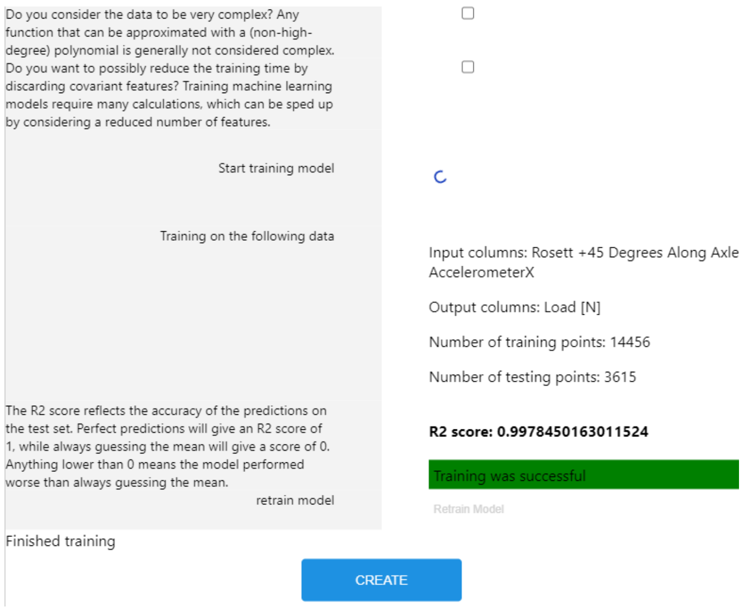
Upload dataset (.csv): Choose File rigg_nicedat...t_for_ml.csv

Which value do you want to predict? (only one value): Load [N]

Which values do you want to use to predict ...?: Rosett +45 Degree... AccelerometerX

Figure E.22: The user has to choose a datasource and sensor value for real-time predictions, upload a dataset and select the sensors that should be used for making the predictions.

The machine learning model is trained in the browser and the user can see information about input columns, output columns, number of training points and number of testing points to get a better overview of the training as shown in Figure E.22. Regularization is applied to the data set if the model does not converge after two iterations. If the model does not converge or has a high loss, the user will get an error message saying the training was not successful. If the training is successful but with limited accuracy, the user will be presented with a warning message. The accuracy is based on the R2 score which reflects the accuracy of the predictions on the test set. If the training is successful, the user will be shown a message with the R2 score.


Do you consider the data to be very complex? Any function that can be approximated with a (non-high-degree) polynomial is generally not considered complex. ☐

Do you want to possibly reduce the training time by discarding covariant features? Training machine learning models require many calculations, which can be sped up by considering a reduced number of features. ☐

Start training model

Training on the following data

Input columns: Rosett +45 Degrees Along Axle AccelerometerX

Output columns: Load [N]

Number of training points: 14456

Number of testing points: 3615

The R2 score reflects the accuracy of the predictions on the test set. Perfect predictions will give an R2 score of 1, while always guessing the mean will give a score of 0. Anything lower than 0 means the model performed worse than always guessing the mean.

R2 score: 0.9978450163011524

Training was successful

retrain model

Retrain Model

Finished training

CREATE

Figure E.23: The user has to answer two questions regarding the dataset for making better predictions.

After the model is trained, the user can create the tile and the real-time predictions will show up in the dashboard as shown in Figure E.23. The real-time data is shown for the sensor the user wanted to predict. Predictions are visualized together with the real-time data. The platform is doing predictions in real-time but it is also possible to do manual predictions. If the user wants to type in some values for the input sensors and see the predicted value, this can be done with by clicking the settings button on the tile making the predictions.

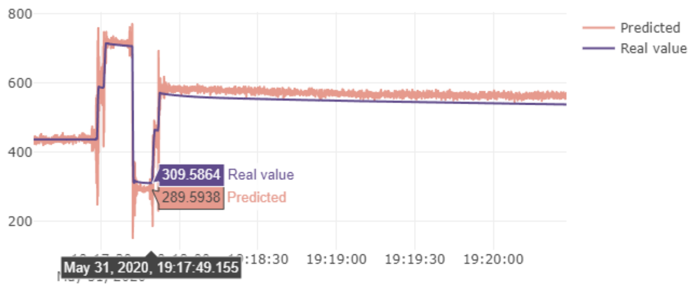


Figure E.24: Real-time predictions of the "load" sensor based on the "accelerometerX" value from the torsion bar suspension rig.

Historical Curve Plot Historical curve plots can be generated the same way as a real-time curve plot shown in Figure E.17. Historical values from this datasource's sensor(s) will be queried and displayed in the plot. It is also possible to plot values from a file, as Figure E.25 shows. The file is expected to have the same format as Figure E.18 illustrates. Switching between the file and datasource options is done by clicking on "Use Datasource" and "Upload File".

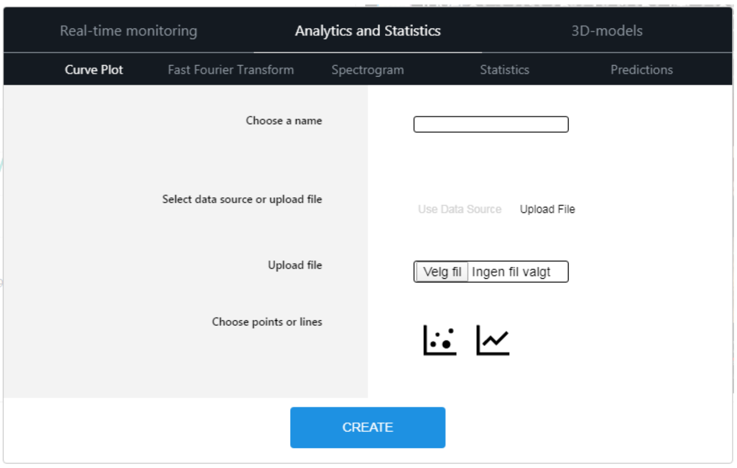
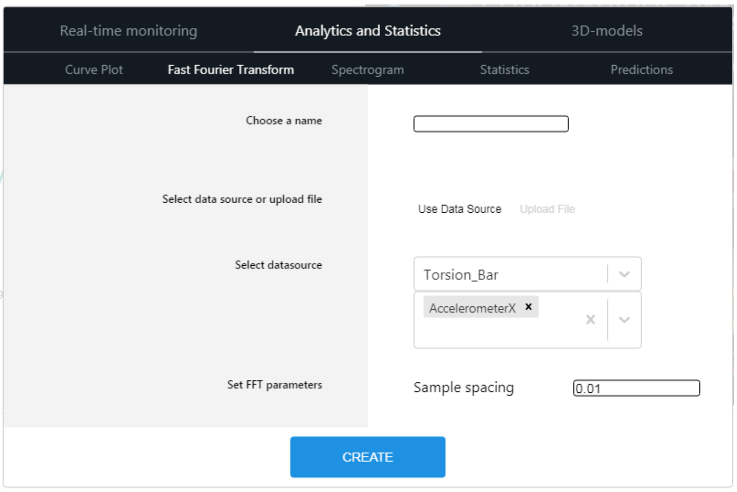


Figure E.25: Creating a historical plot from file

Fast Fourier Transform The FFTs can be generated using the same input types as the previous paragraph describes, and toggling between the two options is done the same way. Additionally, the sample spacing of the data must be provided. This is the time interval between data point or $\frac{1}{f}$, where f is the sampling frequency. The sample spacing variable must be provided for both input types. Figure E.26 shows FFT configuration for an acceleration sensor with sample spacing of 0.01, i.e. sample frequency of 100 Hz.



Choose a name

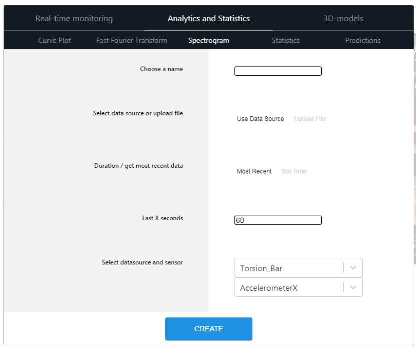
Select data source or upload file: Use Data Source Upload File

Select datasource: Torsion_Bar; AccelerometerX

Set FFT parameters: Sample spacing 0.01

Figure E.26: Add a fast Fourier transform

Spectrogram Spectrograms can be generated from a datasource or file as in Figure E.27. If a datasource is selected as input, select duration of the sensor data that should be used for spectrogram generation. The duration could either be provided in seconds, which generated a spectrogram of the most recent data for this interval. Otherwise, one can select an interval by clicking "Set time" instead of "Most recent" and select date and time for start and end times of the data to be used. If a file is selected as input, the sample frequency of the dataset must be provided.



Choose a name

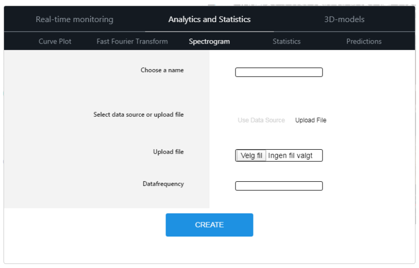
Select data source or upload file: Use Data Source

Duration / get most recent data: Most Recent

Last X seconds: 60

Select datasource and sensor: Torsion_Bar, AccelerometerX

(a) Create spectrogram from datasource



Choose a name

Select data source or upload file: Upload File

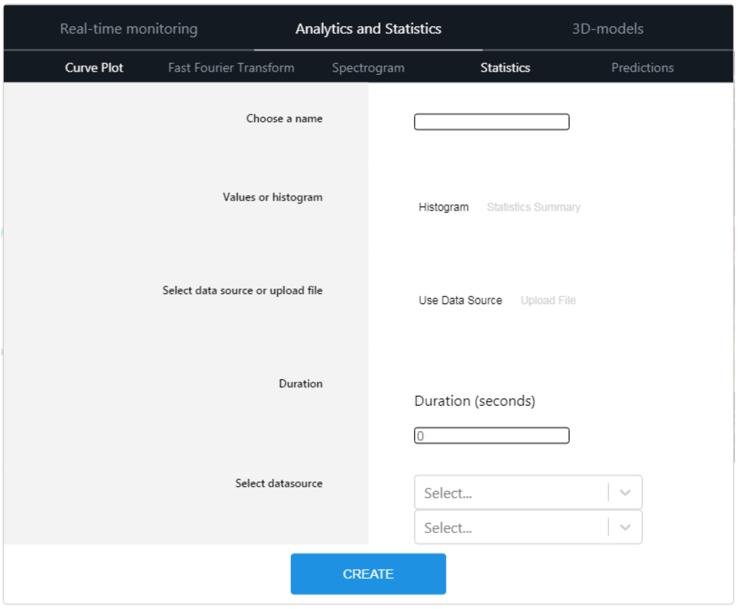
Upload file: Velg fil | Ingen fil valgt

Datafrequency

(b) Create spectrogram from file

Figure E.27: Adding new spectrogram from datasource and file respectively

Statistics Statistical information could be displayed as histogram(s) visualizing the variation of the data values or in a table with statistical information such as mean value, variance etc, and the buttons "Histogram" and "Statistical Summary" toggle the selection. Configuration of a histogram is identical to configuration of a statistics summary. Figure E.28 shows configuration of a histogram tile using a datasource. Fill inn desired sensor(s) and select a duration of the historgram/statistical summary. If you desire to retrieve statistical information from a file, there is no need to select duration, the statistical information from the whole time series of the file will be used.



Choose a name

Values or histogram: Histogram | Statistics Summary

Select data source or upload file: Use Data Source | Upload File

Duration: Duration (seconds) 0

Select datasource: Select... Select...

Figure E.28: Configuration of a statistics tile

3D models If you desire to add a model to the dashboard, Figure E.29 shows the configuration. Simply select format of the model file (FMU or FMM) and upload the required file(s). An FMU upload requires one file, an FMU file. The FMM format requires an FMM file and then corresponding FTL-files.

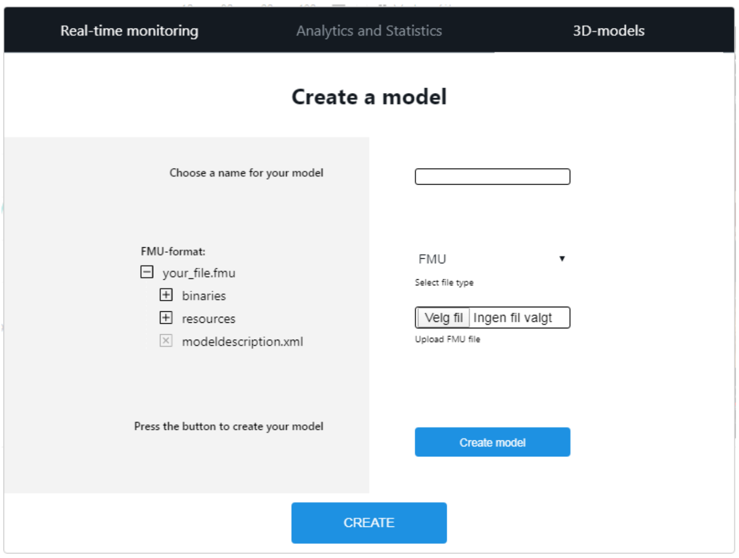


Figure E.29: Adding a model to the dashboard

E.8 Tile Settings

To open the Tile settings, click on the hamburger menu in the top right corner of the tile as seen in Figure E.30. The settings window will open, and it looks like in Figure E.31.

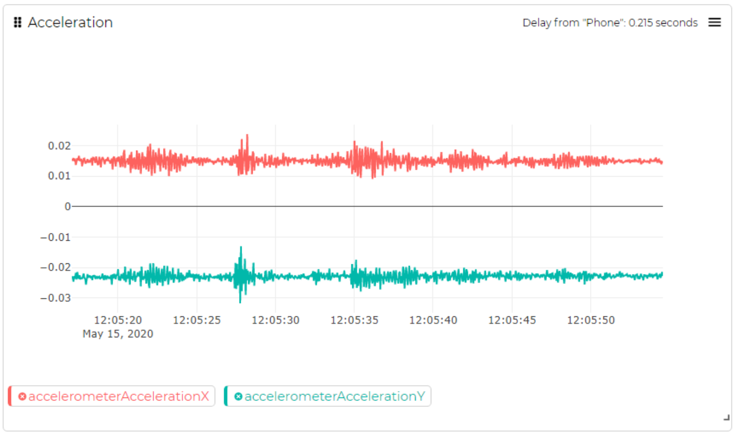
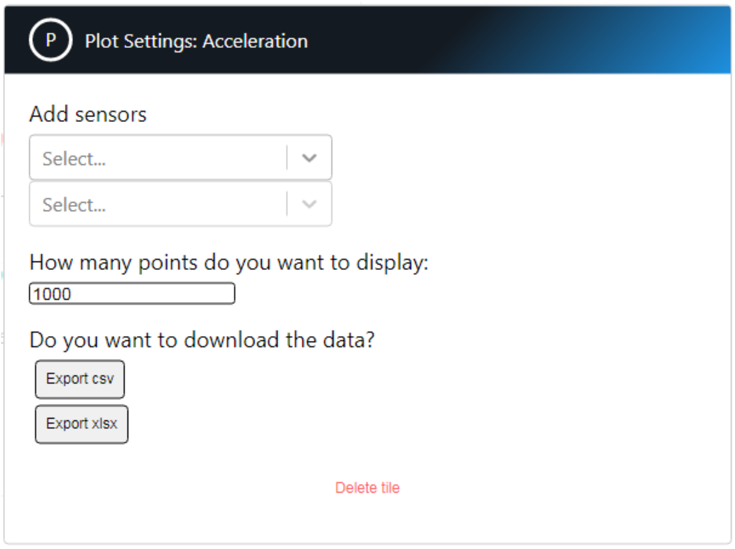


Figure E.30: Curve plot tile



Plot Settings: Acceleration

Add sensors

Select...

Select...

How many points do you want to display:

1000

Do you want to download the data?

Figure E.31: Settings for a real-time curve plot

E.8.1 Adding and removing sensor values from tiles

The selected sensors in a tile are displayed at the bottom in colors corresponding to the plotting colors. If you want to remove a sensor from the plot, simply click on it. Clicking on accelerationAccelerationX in Figure E.30 removes it from the plot. Only accelerationAccelerationY will remain. If you want to add it again, open the settings as described in the previous paragraph. Below "Add sensors", it is possible to add more sensors by selecting a datasource and one or more of its sensors.

NB! Adding and removing sensors is only possible for tiles made with datasources as input. Tile types that can add and remove sensors include real-time and historical plots and statistics.

E.8.2 Downloading data from Tiles

Open the tile settings. If the tile provides this functionality, there will be a "Do you want to download the data?" header. Click "Export csv" or "Export xlsx" to download the data on the format illustrated in Figure E.18. Video streams, maps and models do not provide this functionality.

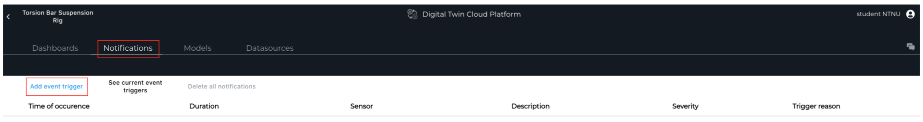
E.8.3 Adjust number of data points in plot

In the tile settings, adjust the number of points displayed in the plot at all times by changing the input field below the header "How many points do you want to display".

E.9 Event triggers and notifications

E.9.1 Create Event Trigger

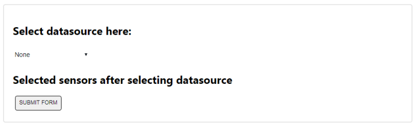
From the project page, click on the Notification tab framed in Figure E.32 and then the "Add event trigger" button.



Time of occurence	Duration	Sensor	Description	Severity	Trigger reason

Figure E.32: Notifications page

The event trigger window will show up looking like Figure E.33a. Select a datasource. The channels of the datasource will appear below the source. Select one or more sensors to add event trigger for these sensors. For each sensor, minimum and maximum values, description for each of them and severity must be filled in. A completed event trigger configuration can look like Figure E.33b. Click on "Submit form".

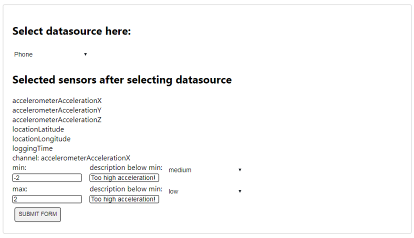


Select datasource here:

None

Selected sensors after selecting datasource

(a) Add event trigger window



Select datasource here:

Phone

Selected sensors after selecting datasource

accelerometerAccelerationX
accelerometerAccelerationY
accelerometerAccelerationZ
locationLatitude
locationLongitude
loggingTime
channel: accelerometerAccelerationX

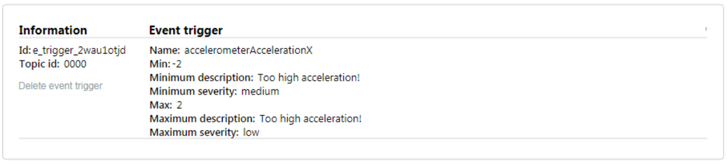
min: -2 | description below min: Too high acceleration! | medium
max: 2 | description below min: Too high acceleration! | low

(b) Event trigger Configured

Figure E.33: Initial add event trigger window and completed configuration of an event trigger

E.9.2 See Current Event Triggers

The event trigger will **not** appear in the list of the notifications page in Figure E.32, but if you click on "See current event triggers" to the right of "Add event trigger", the event trigger should be in the list. Figure E.34 shows the event trigger that was just made. Delete the event trigger by clicking on "Delete event trigger".

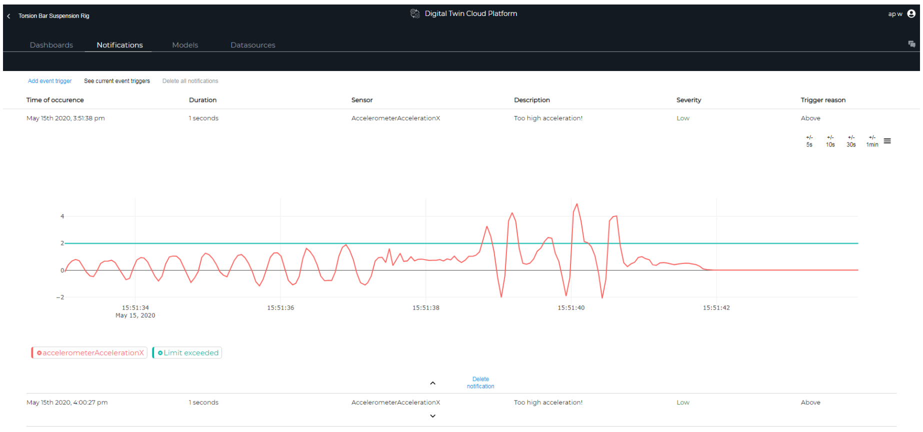


Information	Event trigger
Id: e_trigger_2wau1otjd Topic id: 0000 Delete event trigger	Name: accelerometerAccelerationX Min: -2 Minimum description: Too high acceleration! Minimum severity: medium Max: 2 Maximum description: Too high acceleration! Maximum severity: low

Figure E.34: List of current event triggers

E.9.3 See Notifications

If sensor values above maximum or below minimum are registered, a notification will be generated. The notification will appear in the list on the notifications page. To view a plot of the occurence, click on the down arrow. A graph like the one in Figure E.35 will appear with a plot of the sensor and the trigger value. To change the duration of the plot, use the "+/- Xs" buttons, where X is 5, 10, 30 and 60. Other sensors from the same datasource can be added the same way sensors are added to a tile by clicking on the hamburger menu on the top right. To download the data from the event, click on "Download data" at the bottom of the settings.



Time of occurence	Duration	Sensor	Description	Severity	Trigger reason
May 15th 2020, 3:51:38 pm	1 seconds	AccelerometerAccelerationX	Too high acceleration!	Low	Above
May 15th 2020, 4:00:27 pm	1 seconds	AccelerometerAccelerationX	Too high acceleration!	Low	Above

Figure E.35: Plot from an event in the notifications list

Appendix F

Installation Guide

This guide is contains information about required downloads, installations and steps to set up the front and back ends to facilitate further development of the project.

F.1 Downloads and Installations

In order to run the project, both back-end and front-end, there are some required installations. They are listed below along with a link to the download page. Zookeeper is also required, but it is included in the Kafka download. The Kafka download file should be unpacked.

Name	Link
node.js	https://nodejs.org/en/download/
java	https://www.java.com/en/download/
kafka	https://kafka.apache.org/quickstart
python	https://www.python.org/downloads/

F.2 Guide

Parts of the commands will be different on mac and windows operating systems, but they are included in the setup below.

F.2.1 Send data from the torsion bar rig to the back-end solution

To send data from the torsion bar suspension rig to the back-end solution, it is necessary to start Catman in the computer located next to the torsion bar suspension rig. Follow the steps in Jensen 2019 to set up Catman with the correct settings. The data is sent from Catman to the server's IP address by using UDP communication. The back-end solution receives the data and makes it available to the users of the platform.

F.2.2 Back-end

In order to run the back-end, do the following steps:

1. Download the back-end solution by running the following command in the terminal:
   git clone https://github.com/erikkjernlie/digtwin_backend

2. Run the command 'pip install –r requirements.txt' in the downloaded folder to install the necessary requirements.

3. Download and install Kafka. Kafka requires java, so download if it is not already installed.

4. Open a new terminal window, navigate to the Kafka folder and run the following command depending on the operating system:
   Mac: bin/zookeeper-server-start.sh config/zookeeper.properties
   Windows: bin\windows\zookeeper-server-start.bat config\zookeeper.properties
   This starts a Zookeeper server, which is required by Kafka.

5. Open a new terminal window, navigate to the Kafka folder and run the following command depending on the operating system:
   Mac: bin/kafka-server-start.sh config/server.properties
   Windows: bin\windows\ kafka-server-start.bat config\server.properties

   - Running Kafka might result in the following Java-error: "Please install or use the JRE or JDK that contains tehse missing componenets. Error: missing 'server' JVM at . . . "
   - If this error occurs, do the following:

  - Go into your “Java” folder. This is typically something like ’C:/Program files/java’
  - Go into jre7/8 directory and go into the bin folder
  - Create a folder called “Server”
  - Now go into the C:/Program files/java/jre8../bin/client folder
  - Copy all of the files in this folder into the Server folder

6. Navigate to the repo that was cloned in step 1. Set necessary ports in the settings.py file. The default port for the Kafka server should be set to “9092”.

7. Run the script “main.py” to run the back-end.

F.2.3 Front-end

When the back-end is running, the front-end can be started

1. Download the back-end solution by running the following command in the terminal:
   git clone https://github.com/erikkjernlie/digtwin_backend

2. Run the command ’npm install’ or ’yarn’ from the digtwin_backend folder in the terminal to install the required libraries.

3. Run the following command: ’npm start’ or ’yarn start’. The application will start, and generate a local address which can be pasted into the browser.

Appendix G

Front-end deployment

The client of the platform is hosting using Surge. Surge makes it easy to deploy projects for free through npm, by only writing a few lines of code in the command line. The process of deploying the front-end application to surge is shown in Listing G.1. The only prerequisite is the latest version of node.js, which can be downloaded from https://nodejs.org/en/download/. First, the application is built. In the build folder, the index.html file is duplicated and called 200.html. Then, surge is installed and run. The final step is to select a domain name and press enter. The project is hosted at "project-name.surge.sh".

```
npm run build
cd build
cp index.html 200.html
npm install --global surge
surge
```

Code lising G.1: Deployment from the terminal when the current directory is inside the front-end solution