Bachelor's thesis

NTNU
Norwegian University of Science and Technology

André Marthinsen, Øystein Qvigstad, Paul Schiager, Even Stetrud

Leveraging Visual Programming for Building Web Calculators

Bachelor's thesis in Programming
Supervisor: Peter Stefan Nussbaum
May 2024

NTNU
Norwegian University of Science and Technology

André Marthinsen, Øystein Qvigstad, Paul Schiager,
Even Stetrud

Leveraging Visual Programming for Building Web Calculators

Bachelor's thesis in Programming
Supervisor: Peter Stefan Nussbaum
May 2024

Norwegian University of Science and Technology

NTNU
Norwegian University of
Science and Technology

Abstract

Title: Leveraging Visual Programming for Building Web Calculators		**Date:** 21.05.2024
Participants: André Marthinsen, Øystein Qvigstad, Paul Schiager, Even Stetrud		
Supervisor(s): Peter Stefan Nussbaum		
Employer: Skogkurs		
Keywords (3-5): PWA, Visual Programming, Web Development, Graphs		
# of pages: **112**	**# of appendixes:** **16**	**Availability:** **Open**

Short description of the bachelor thesis:

In an age seeing an increasing shift towards mobile computing, Skogkurs had a goal of moving a previously Excel based cost estimation tool to the web in a format usable on mobile devices as well as stationary devices.

This thesis explores Progressive Web Apps, structuring of web content through JSON, and the use of node based visual programming for the purpose of editing web content, and in particular how useful it is for non-programmers in managing said content.

The project has been carried out in close cooperation with Skogkurs and their partners on their premises in Brumunddal through weekly meetings.

While visual programming provides intuitive visualization of flow from input to output in a calculation, our experience shows it does not lend itself to building up large mathematical formulas, though providing sub-graphs does help in reducing visual clutter in such cases.

Sammendrag

<table>
<tr><td colspan="2">Tittel:
Leveraging Visual Programming for Building Web Calculators</td><td>Dato:
21.05.2024</td></tr>
<tr><td colspan="3">Deltakere:
André Marthinsen, Øystein Qvigstad, Paul Schiager, Even Stetrud</td></tr>
<tr><td colspan="3">Veileder(e):
Peter Stefan Nussbaum</td></tr>
<tr><td colspan="3">Oppdragsgiver:
Skogkurs</td></tr>
<tr><td colspan="3">Nøkkelord (3-5):
PWA, Visuell Programmering, Web-utvikling, Grafer</td></tr>
<tr><td>Antall sider:
112</td><td>Antall vedlegg:
16</td><td>Publiseringsavtale:
Åpen</td></tr>
<tr><td colspan="3">Kort beskrivelse av bacheloroppgaven:
I en tid med en økende overgang til mobil databehandling hadde Skogkurs som mål å flytte et tidligere Excel-basert kostnadsestimeringsverktøy til nettet i et format som kunne brukes på mobile enheter så vel som stasjonære enheter. Denne oppgaven utforsker Progressive Web Apps, strukturering av webinnhold gjennom JSON, og bruk av nodebasert visuell programmering for redigering av webinnhold, og også hvor brukervennlig en slik løsning er for ikke-programmerere i håndtering av nevnte innhold.
Prosjektet har blitt gjennomført i nært samarbeid med Skogkurs og deres partnere på deres lokaler i Brumunddal med ukentlige møter.
Selv om visuell programmering gir en intuitiv visualisering av flyten fra input til output i en beregning, viser vår erfaring fra prosjektet at det ikke egner seg for å bygge opp store matematiske formler, selv om bruk av undergrafer hjelper med å redusere visuell overbelastning.</td></tr>
</table>

Preface

This bachelor's thesis is written at Department of Computer Science in NTNU Gjøvik. We are pleased with the project result after working for several months, and have gained insights and experience valuable for future projects.

We want to express our gratitude to everyone involved in this project.
We had an excellent cooperation with our employer, Skogkurs. We would like to thank everyone involved, especially Mikael Fønhus and Brede Lauten for their role as product owners, Merete Nyheim for feedback on UX design, and Camilla Glomsås for support with sustainability in a forestry context.
We would also like to thank the external stakeholders Erling Perstølen (Allskog), Jon Sigurd Leine (Glommen-Mjøsen Skog), Bjørn Lauritzen (MEF), and Helmer Belbo (NIBIO) for valuable feedback and insights into the forestry industry throughout the project.
Finally, we would like to thank our supervisor Peter Stefan Nussbaum for support and feedback from start to finish of the project.

Contents

Figures

Tables

Code Listings

Acronyms

API Application Programming Interface.

CI/CD Continuous Integration/Continuous Deployment.

CIA Confidentiality, Integrity, Availability.

CRA Create React App.

IaaS Infrastructure as a Service.

IDaaS Identity as a Service.

NPM Node Package Manager.

PaaS Platform as a Service.

SaaS Software as a Service.

SDK Software Development Kit.

TLS Transport Layer Security.

Chapter 1

Introduction

Skogkurs is a Norwegian organization dedicated to strengthening the financial development and stewardship of Norwegian forests and other land resources. Their primary activity in this regard, is to disseminate knowledge from the scientific community to the participants in the forestry industry, as well as to the Norwegian educational system and the broader public.

Among the software tools Skogkurs offer to the industry is a downloadable Excel spreadsheet that can calculate cost and productivity of logging operations. The task commissioned by Skogkurs was to offer the same functionality the existing Excel document offer available on the web, ideally expanded with the ability to calculate additional scenarios/forms of logging. After much consideration, which will be expanded upon later in the thesis, the bachelor group decided to develop the following solution:

1. **A web application** capable of reading a given data structure representing a mathematical formula, which must also dictate how input fields and the results should be presented.
2. **A web editor** capable of creating and modifying existing formulas, supporting a broad range of mathematical operations and creating visualizations such as graphs, tables, pie charts, etc. to be displayed in the web application.
3. **A scalable backend** that stores formulas generated by the web editor, and performs calculations on these formulas for API users partnered with Skogkurs, as an alternative to using the web application.

Instead of merely converting the existing Excel document into a website, the project aimed to create a holistic solution that can be maintained by non-technical staff and remains cost-effective for the future development of additional calculators.

1.1 Background and Problem Statement

The calculation tool is used to estimate cost ($\frac{kr}{m^3}$) and productivity ($\frac{m^3}{G_{15}}$) of a logging operation, categorized by machine. A harvester machine cuts and delimb trees that are large enough for harvesting, while also clearing smaller trees to facilitate access to larger ones as needed. The processed logs are subsequently collected by a forwarder and transported to a loading area. The cost and productivity of the operation are influenced by various factors, including the composition and volume of the forest stand, the terrain type, travel distances, and other elements.

The motivation for migrating the Excel spreadsheet to a web-based platform is attributed to several factors, including but not limited to:

- The spreadsheet relies on macros for its calculations, which are frequently disabled or flagged as security risks on end users devices.
- A significant number of end users are on mobile devices, particularly when assessing a forest in person. As a result, offering the tool as a spreadsheet is impractical for many.
- Providing the tool as a downloadable spreadsheet presents challenges in notifying users of new versions and distributing updates. Additionally, this approach makes it difficult to track tool usage.
- The existing spreadsheet is limited to a single harvesting method, *clear-cutting*. Skogkurs partners are requesting the ability to calculate additional harvesting methods, such as *selective/patch cutting* and *shelterwood* harvesting, along with the inclusion of other cost factors.
- The maintainer of the spreadsheet is no longer available, making it difficult to extend functionality.

1.2 Scope and Objectives

The goals of the project are divided into three categories: outcome goals, impact goals and learning goals. The outcome goals in particular pertain to the design of the software, and as such they are elaborated upon in 3.2

1.2.1 Outcome Goals

The application must support:

- Creating new formulas with different cost drivers.
- Saving and retrieving versioned formulas and cost calculations.
- Usage on a variety of device types, like PCs, tablets, and smartphones.
- Usage on multiple operating systems, like Windows, iOS and Android.

- Usage in the field, even without internet access.

1.2.2 Impact Goals

The application must:

- Ensure that all users at any given time have access to the latest version of any particular formula.
- Work as a tool to plan logging operations, as used by various actors in the forestry industry, like machine operators, forest owners and forestry operations managers.
- Serve as a tool in Skogkurs' teaching courses, like "Bedriftsøkonomi for entreprenører".

1.2.3 Learning Goals

- Gain experience with processing and analyzing a real product's owner problem, formulate a solution for that problem area and finally implement that solution.
- Expand and develop our experience in the field of web development.
- Apply and expand upon knowledge and skills acquired throughout the three year study program.

1.3 Target Audience

This bachelor's thesis is primarily intended for the thesis evaluator but may also be of interest to anyone engaged in software development, especially web development. The report assumes a certain level of familiarity with concepts in these fields, which may pose challenges for readers without this background, especially the technical chapters.

The target audience for the product itself differs from that of the report. The intended users of the product are individuals in the education sector and stakeholders in the forestry industry, including contractors, machine operators, and forest owners.

1.4 Background and Competence

All four members of the project group are students from the Programming Bachelor's Degree Program, and as such possess the same basic educational background, with minor variations depending on which elective courses have been taken.

From the Bachelor's Degree Program we have applied acquired knowledge about the broader field of software development in general, and more specialized disciplines from amongst others the following courses:

- PROG2053 - Web Technologies: web development in general, Express and Node frameworks in particular.
- IDATG2102 - Algorithmic Methods: datastructures, particularly graphs and trees.
- PROG2005 - Cloud Technologies: REST APIs and deployment.
- PROG1004 - Software Development: Scrum and agile development, UML and modelling, project managment.
- IDG1362 - Introduction to User-Centered Design: Iterative design, prototyping, user testing methodologies.

1.5 Project Framework

Skogkurs had no particular technological or methodological requirements for the process or the software. They however did specify that:

- The calculator presented in the web app must recreate the formulas from the original Excel spreadsheet.

Because the software should be delivered to Skogkurs, the project group identified one central guideline:

- The software should be maintainable without experience in the field of web development.

Lastly, the final deadline for delivery of the project was **May 21st**.

1.6 Role Descriptions

The distribution of responsibilities in the project group was organized as follows:

- Øystein Qvigstad: Scrum master (until April 3rd), developer and deployment lead
- André Marthinsen: Developer and testing lead
- Paul Schiager: Developer, scrum master (from April 3rd), design lead and joint security lead
- Even Stetrud: Developer, documentation lead and joint security lead

The group's thesis supervisor was Peter Nussbaum. Meeting weekly with the group he gave feedback on the general status of the development process, as well as input on the academic aspects of the report and the thesis in general.

Skogkurs has been represented by Brede Lauten and Mikael Fønhus in weekly meetings with the project group, providing instrumental input on specifying the requirements for the application. Irregular meetings were also held with other employees of Skogkurs,

primarily to discuss complying with their guidelines for graphic design, and for discussing the sustainability of the application.

In addition to Skogkurs, several other stakeholders in the development of a web based calculation tool were represented throughout the majority of the project period. These stakeholders were a combination of forest owners associations, namely Glommen-Mjøsen Skog and Allskog, the Machine Owners Union, MEF, and the research organization NIBIO. Their representatives contributed with feedback on the application and aided in the process of gathering requirement specifications.

1.7 Report Overview

This report consists of 11 main chapters.

- Chapter 1, Introduction: Introduces the project owner Skogkurs, the project and the project group.
- Chapter 2, Theoretical Framework: Provides a survey of some concepts that will prove beneficial to understanding the details of the software implementation, more specifically Progressive Web Apps and some of the data structures used in the dynamic version of the calculator.
- Chapter 3, Overview and Requirements: Gives an overview of the application and its components, and details their respective requirements.
- Chapter 4, Development Process: Details and reflects on the software development model, Scrum, utilized in the project, and our interaction with Skogkurs.
- Chapter 5, Technical Design: Describes the application's system architecture, and details the decisions and alternatives in the application's design from a technical standpoint.
- Chapter 6, Graphic Design: Gives insight into the process used in designing the interface of the web app and web editor, and decisions that were made as a result of this process.
- Chapter 7, Implementation Details: Provides a detailed description of the choices made and challenges faced in implementing the software for the application.
- Chapter 8, Testing and Quality Assurance: Details the methodologies used for testing the application's code, the formal user testing of the web app and the accuracy of the web-based calculation.
- Chapter 9, Deployment: Gives a survey of the organization and deployment of the application, including branching strategies and CI/CD pipeline.
- Chapter 10, Security: Describes the steps taken to secure the application, and the frameworks with which we have identified and analyzed security risks.
- Chapter 11, Discussion and Conclusion: Critiques the degree to which we achieved our

goals, the development process in general, and ethical and sustainability concerns.

Chapter 2

Theoretical Framework

In this chapter we go over a few concepts that form the theoretical basis for some of the topics we discuss in section 3, Implementation and Evaluation. Summarized we will go over the basics of binary expression trees and other related data structures that form the basis of the data representation used for persistence of calculator tools in our application, node graph architecture and its relevancy to the GUI design of the editor, as well as Progressive Web Apps in relation to the web app and offline use.

2.1 Data-structures

A challenge faced in creating an editor capable of generating mathematical functions for use in the web app was to find an appropriate format for storage and transmission. There were a few considerations we had to keep in mind; As the data structure would in effect represent some sequence of code to be run on an end-user's machine, we had to make sure that there were well defined restraints on possible outcomes of executing the function represented by the data. For purposes of further development and maintenance, the data structure could not be too esoteric, and if practical, human readable.

Rather than re-invent the wheel, we looked to a couple of reasonably well known data structures suitable for our purposes. While not a one-to-one mapping with what's in use in the editor, they are similar enough that understanding them will provide a solid foundation when we go into more detail about the editor later.

2.1.1 Graphs and Trees

We assume most readers of this paper will have some fundamental understanding of graphs and trees, but for the sake of accessibility we will briefly touch on the very basics of these data structures.

```
function evaluate(node):
    if(node contains number)
        return number
    if(node contains unary operator)
        leftValue = evaluate left child
        return result of unary operation on leftValue
    if(node contains binary operator)
        leftValue = evaluate left child
        rightValue = evaluate right child
        return result of binary operation on leftValue and rightValue
```

Figure 2.1: Pseudo code for evaluating a node in a binary expression tree

Graphs A graph is a data structure made up of nodes and edges. Nodes are objects containing data, such as a number, a function or a string. These nodes can then be connected, most often through a reference to another node. Such a connection is referred to as an edge, and it may be directional in the sense that traversal should only happen in one direction. By starting out on some node in the graph one can traverse it in any fashion allowed by the direction of the connections.

Trees Trees are graphs with a few additional properties, so named because they start in one node, known as the root node, before branching out from the root to so called *child nodes*. These children can in turn have children of their own. Another rule is that for two arbitrary nodes n and m, they may only be connected through one path, where a path is a traversal along connected nodes. Nodes that have children are referred to as *internal* nodes, whereas nodes with no children are known as *leaf* nodes.

2.1.2 Binary Expression Trees

A binary tree is a tree where each node may have zero to two children. A binary expression tree is a tree where internal nodes represent some operation to be performed on its children, which can be an unary or binary operation, and the leaf nodes represent some value. In short, it is a way to represent a mathematical expression composed of binary and unary operations as a data structure.

If we consider the expression $5+1/5$, it can be represented as seen in figure 2.2. An advantage of such a representation is that it is far easier to work with when writing algorithms. In figure 2.1 we see pseudo code laying out the basic principle of evaluating the value of a node in such a tree. The function takes any node, assuming the node contains either a numeric value or an operation on numeric values.

If the the contents of the node is a number it simply returns the number. If the content of the node is an operation, such as + or *, then the operation is applied to its children. If the child is an operation and not a concrete value, *evaluate* is called on the child before the operation is applied, and so it continues all the way to the leaf nodes until the root

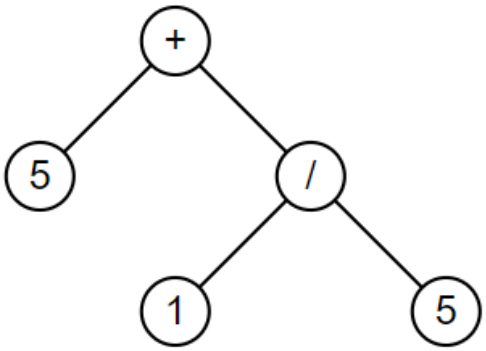


Figure 2.2: A binary expression tree representing 5 + 1/5

of the tree has been evaluated, in this case 5,2. A consequence of this behaviour is that precedence of operation can be encoded through the 'depth' of a node, meaning the count of edges between the node and the root node. In the example, 1/5 is further from the root. As 1/5 has to be evaluated before the + operator can be applied to its children, this ensures the operations are carried out in the right order.

2.2 Node Graph Architecture

There is a category of software design where the software is wholly or partially built around the concept of graphs, where a *node* represents some function with a set of inputs and outputs. This goes both for the underlying source code and the graphical representation provided to the user of the software, each node available to the user having some underlying implementation, often as a class or dedicated function. This allows users to design flow of control and data through linking these nodes visually through dragging edges around on a canvas, allowing visual programming of software behaviour.

A big caveat here is that this definition is largely based on a wikipedia article[1] with few references, but as there seems to be no other formal definition, we've opted to include it as it works well as a framing device for continued discussion.

2.2.1 Strengths

Such a design can be a powerful abstraction of the underlying implementation, allowing users to focus on the outcome without needing the ability to code or knowledge about the mechanics of the behaviour provided by the nodes. This reduction of exposed complexity

[1]https://en.wikipedia.org/wiki/Node_graph_architecture

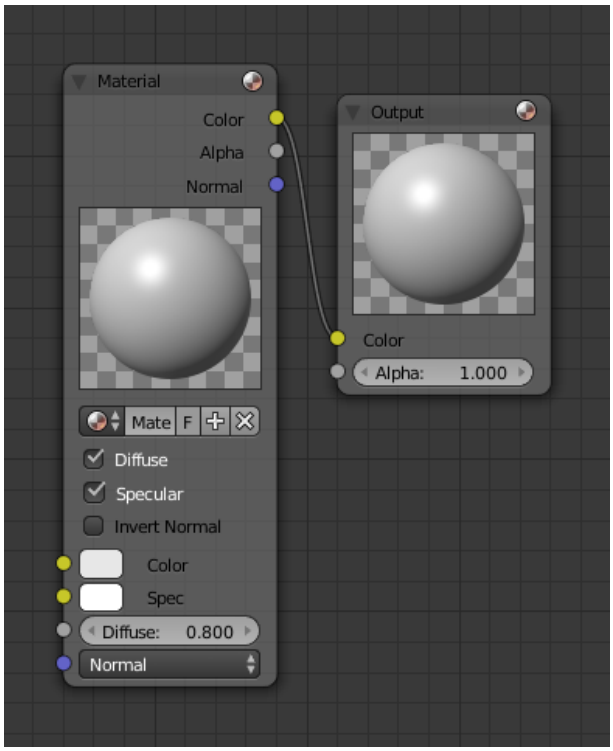


Figure 2.3: Nodes representing a surface material in Blender

can lead to increased learnability and usability, such as in the case of visual programming used within education. Visual programming tools such as Snap![2] or Scratch[3] provide the users with simple blocks of functionality that can be composed into program behaviour. Where these provide function input and output as geometrical shapes that can interlock as shown in figure 2.4, node based visual programming provide input and output as *sockets* on the node, usually with color coding and textual hints of the type of the socket, allowing only sockets of the same type to be connected, as shown in figure 2.3.

Another advantage is how the style of visualization may make it easier to understand flow of data and control. Cause and effect is visually encoded as edges from outputs to inputs on nodes, allowing the user to follow an input through the graph and see where and how it is used, a process which may be far more complex in code or some other visual representations. For this reason the node graph design has seen increasing use within the graphics, game and machine learning domains for the purpose of data processing, such as setting up inputs, intermediary data manipulation and properties of the final output.

2.2.2 Weaknesses

The level of abstraction provided by a node graph may come at the cost of flexibility, as one can only program with what's provided by the nodes available. For example, if there is

[2]https://snap.berkeley.edu/
[3]https://scratch.mit.edu/

no node encoding looping behaviour, it may be impossible for the user to program looping, while it may be perfectly doable in the language implementing the node system. A consideration then is tedium of use versus flexibility, a topic we will return to in section 5.3, but in short, the more granular the functionality provided by the nodes, the more nodes are needed to achieve a particular task. The consequence is that the same behaviour at a lower level of abstraction will involve more visual clutter due to the increased node count, and in turn a reduction of usability. A node graph design can be said to be located on a spectrum between flexibility and usability, where the more flexible systems will be able to output a greater variety of programs, but be harder to learn and use.

Any desired behaviour in the form of nodes also comes with the added cost of implementing a graphical representation and underlying code. As the implementation of node systems often involves some form of inheritance of a base node class, it can quickly incur heavy costs in the form of cumbersome inheritance hierarchies, a topic which will be discussed in section 7.3, editor implementation.

Finally, one has to consider what should be some controller on a node rather than a node input. Will it be more convenient to provide the user with a file picker on a node where an image is needed, a separate image node and corresponding input/output, or a hybrid solution? Such design choices will greatly impact the usability of the design, and it can be hard to predict ahead of time what will ultimately be the best solution, leading to redesigns.

2.2.3 Relevance

So why is this relevant for our project? The project presented us with the need for a tool that would allow a user with limited preexisting knowledge to re-configure or create a new calculator tool in our app. If one considers the input fields in the app the inputs of a function and the ways in which the results of calculations is presented as its outputs, it may not be far fetched to envision the web app as a platform where some multi-input/output function gets visualized, possibly with the need for conditional behaviour. Combining the capability of node graph architectures to simplify the understanding of programmed behaviour, and the ease of processing graphs as discussed in section 2.1.2, it becomes an interesting proposition to explore the usefulness of node graphs for the purpose of visually programming calculation tools due to the close link between data representation and visualization of node graph architectures.

The advantages and disadvantages of the design, implementation and use of our node based system will be discussed in sections 5.3 and 7.3, along with a discussion on more traditional GUI solutions compared to a node based one.

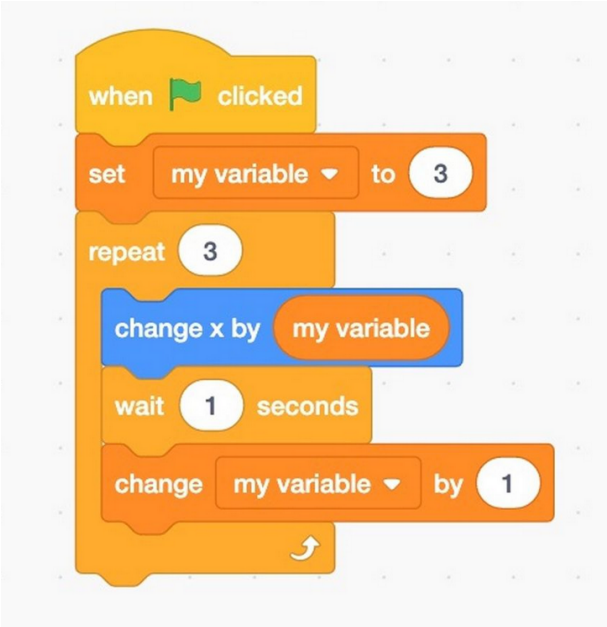


Figure 2.4: Connected blocks encoding program behaviour in Scratch

2.3 Progressive Web Apps

Progressive Web Apps (PWAs) are typically described as applications built using web platform technologies while providing a user experience similar to that of native apps [1]. This broad description likely reflects the ongoing evolution of PWAs, which change over time. At it's heart lies the design principle of *progressive enhancement*, which emphasizes high availability of essential content, and only secondly extends additional features depending on the given capabilities of the system [2]. In the context of PWAs, this means enhancing a website with additional functionality while ensuring compatibility with older web browsers.

Table 2.1: PWA attributes proposed by Russell and Berriman [3]

Attribute	**Description**
Responsive	to fit any form factor
Connectivity independent	Progressively-enhanced with Service Workers to let them work offline
App-like-interactions	Adopt a Shell + Content application model to create appy navigations & interactions
Fresh	Transparently always up-to-date thanks to the Service Worker update process
Safe	Served via TLS (a Service Worker requirement) to prevent snooping
Discoverable	Are identifiable as "applications" thanks to W3C Manifests and Service Worker registration scope allowing search engines to find them
Re-engageable	Can access the re-engagement UIs of the OS; e.g. Push Notifications
Installable	to the home screen through browser-provided prompts, allowing users to "keep" apps they find most useful without the hassle of an app store
Linkable	meaning they're zero-friction, zero-install, and easy to share. The social power of URLs matters.

To gain a deeper understanding of what PWAs are, we can refer to Frances Berriman and Alex Russell, who are widely credited with coining the term *progressive web app* [4]. Table 2.1 outlines the core attributes they proposed. While these attributes remain relevant today, the general consensus is that the essential requirement for a PWA is its ability to be installable [5]. However, it should be noted that web sites aren't classified as PWAs based on their installation status, but rather the ability to offer a user experience akin to that of native apps, which can also be offered directly from the web-browser. To be installable, the web app must meet the following requirements:

1st requirement A manifest file: This is a JSON[4]-formatted text file that instructs the browser on how the PWA should behave once installed. At a minimum, it must include a name, icons for various sizes, and the URL to open when the app launches [6].

[4]JavaScript Object Notation; A text based open source data exchange format

2nd requirement Served using HTTPS: This ensures that communication is encrypted with Transport Layer Security (TLS). It is also a prerequisite for using service workers.

3rd requirement Registering a service worker: This critical aspect will be detailed further in the next section, where we explore its functionality and importance in the architecture of PWAs.

2.3.1 Service Worker

Despite their capabilities, less than 2% of all websites utilize service workers [7]. Given their low prevalence, we cannot assume familiarity among readers. Furthermore, since service workers are a core component of our solution, a dedicated theoretical section is warranted to explain their function and importance.

Before delving into the details, it is important to understand what a service worker is primarily used for. Typically, service workers are utilized to enable offline functionality, benefiting not only Progressive Web Apps (PWAs) but websites in general. This is accomplished by registering another program (often referred to as a "background service" in other contexts) that can manage the behavior of the website, particularly in terms of how it handles network requests and data caching.

Two types of workers

By default, browsers use a single thread to run all JavaScript code [8]. This leads to poor user experience when excessive computation might block the main thread. Workers on the other hand, run in secondary threads, and are capable of offloading work, hence the name *worker*. There exists two types of workers, each with their own use cases, but we are only concerned with service workers in this project.

Table 2.2: Key differences in worker types [9]

Feature	**Web Worker**	**Service Worker**
Lifespan	Coupled to session	Independent of session
Can spawn multiple instances	Yes	No
Can intercept HTTP requests	No	Yes
Use Cases	Heavy computation[5]	Networking and caching[6]

[5]Examples include AI, games, image encoding, compression, etc.
[6]Examples include proxy network requests, caching, provide offline support, etc.

Worker Lifecycle

As detailed in Table 2.2, a service worker operates independently of the user session, allowing tasks to persist and caches to be maintained across sessions. Its lifecycle is entirely managed by the browser. The process begins when the website checks if registration is possible, adhering to the 'progressive enhancement' principle (see Section 2.3). Assuming successful registration and activation, the browser will automatically initiate the service worker whenever the web app is accessed subsequently. Updates to the service worker adhere to the same registration mechanism, with the browser checking for technical conditions to determine if re-registration is necessary or not.

Offline Support

In order to implement offline functionality, the service worker must specifically identify and cache a predetermined set of web files (assets) that constitute the application. This is managed through a file manifest embedded within the service worker (distinct from the PWA manifest). Whenever the source files are modified, this manifest is updated, triggering the browser's byte-by-byte comparison algorithm to refresh the service worker and clear the old cache [10]. This setup ensures that the so-called skeleton files are consistently stored, allowing the app to launch reliably without internet connectivity.

Content that is subject to frequent changes is typically not included in the file manifest; but rather managed at runtime. This is commonly addressed by instructing the service worker to act as a proxy, intercepting HTTP requests and caching their response. Given the varied requirements of different applications, several strategic approaches have been devised to define the interaction between HTTP requests and the cache [11]; see table 2.3 for details.

Table 2.3: Strategies for service worker caching [11]

Strategy	**Description**
Cache Only	Services respond from the cache only, ignoring the network
Network Only	Sends all requests to the network, bypassing the cache
Cache first	Prioritizes cached response; fetches from network if not in cache
Network first	Prioritizes network; uses cache if network fails
Stale-while-revalidate	Delivers cached (stale) data, while updating cache in the background with network request

Complex applications might also combine strategies (which we will return to in section

7.2.2). As shown in Figure 2.5, the service worker sets up a minimum set of files, or *skeleton*, to be stored in cache so they're always ready when the PWA starts. This is called *prefetching*. As the PWA requests content, these requests are subsequently intercepted by the service worker if the browser supports this feature (otherwise they are passed on as normal). The service worker then forwards these requests and saves the responses in the cache for future use, when there might be no internet connection. This setup is a good example of the design principle called *separation of concerns* (SoC), which means "partitioning the software system so that each partition is responsible for a separate concern" [12]. In this context, the PWA is completely unaware that the network requests are intercepted and cached, allowing the business logic and caching rules to be managed separately.

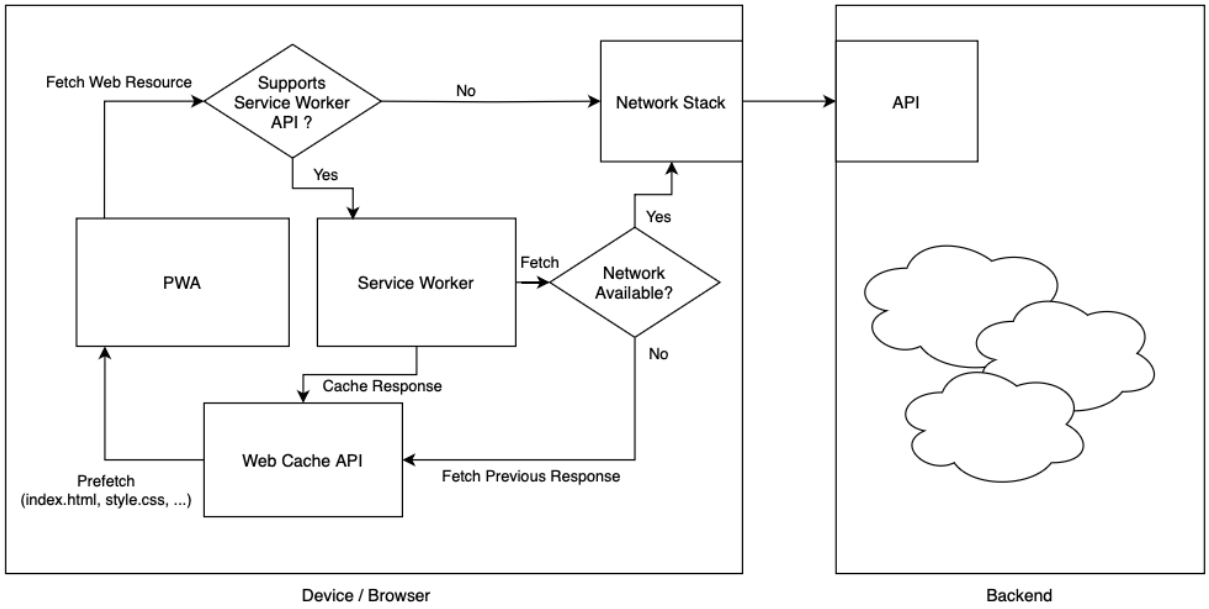


Figure 2.5: *Network-first* caching strategy (see table 2.3)

Chapter 3

Overview and Requirements

This chapter will give an overview of the system with its entities, the respective requirements for each entity with respect to the problem statement from Skogkurs, as well as some discussion on our reasoning for expanding the solution with tooling for content creation.

3.1 Overview

This section aims to provide a basic understanding of the system. Detailed explanations will follow in later sections, so a full understanding is not expected at this stage.

The system comprises two web applications, informally referred to as the *web app* and the *web editor*. The former can be considered as the consumer of the system, while the latter functions as the producer. The primary "*commodity*" of these applications is the data structures that encapsulate both the mathematical formulas and the graphical layout consisting of inputs and visualisations. Together, these elements compose what is known as a single calculator. These calculators are accessible through the web app, as a list of calculators, and can also be integrated with other systems via an API. Furthermore, these calculators are equipped with version control, ensuring that any updates do not disrupt shared or saved results, nor affect existing API queries. In simpler terms, the API acts as the "gatekeeper" to the backend services that manage, store, and deliver these calculators. Supporting the API are two critical components: the authentication service and the database. The authentication service verifies that only authorized users can modify existing calculators, while the database provides a secure repository for storing them.

3.1.1 Rationale for a Dynamic Solution

Early on, the project group noted that Skogkurs didn't have a development team to take over maintenance of the web app post-project. This raised concerns about the practicality of outsourcing, especially in regard to effectively communicating mathematical revisions between

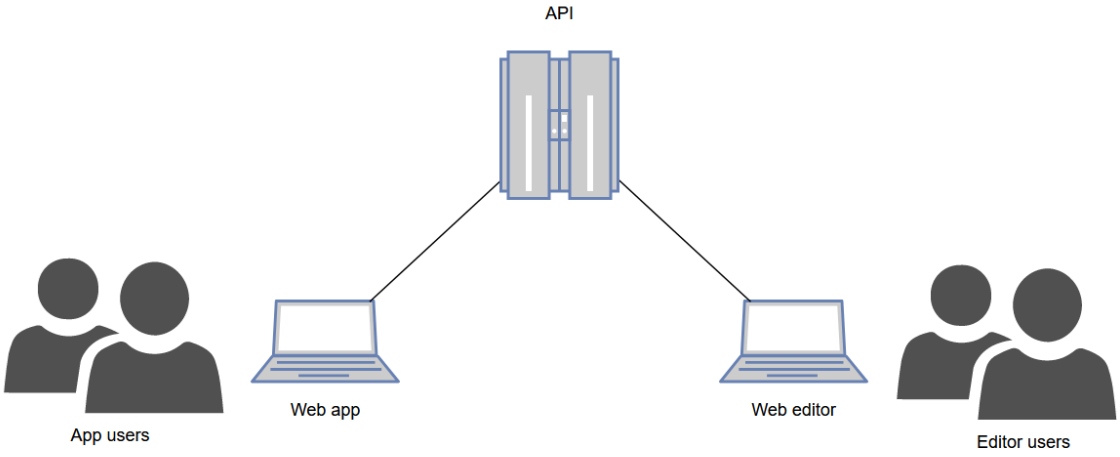


Figure 3.1: High level abstraction of system solution

Skogkurs's science-focused staff and external developers. The idea of building a graphical user interface, enabling Skogkurs employees to update the calculator themselves, emerged as a potential solution. This approach would not only facilitate updates to the mathematical calculations but also to the explanatory texts associated with inputs and results. Additionally, this solution presents a more intriguing subject for a bachelor thesis. As mentioned in the introduction, Skogkurs expressed a desire to extend the mathematical functionality beyond those provided by the current spreadsheet. While analyzing which changes would be necessary, it became clear that merging the various mathematical models was impractical due to their reliance on different explanatory variables. This realization strengthened the team's view that the app's maintainability and scalability would be greatly restricted if it were to be hard-coded with a static formula at the project's end.

3.2 Requirements

The project's requirements are based on Skogkurs' goal of offering a web based cost calculator to the forestry industry. A detailed specification from Skogkurs was not available. Through close cooperation over the course of the project, a understanding of Skogkurs' and other stakeholders' wishes have emerged, which in turn were added as backlog items. Functionality not originally asked for was also added, offering Skogkurs a way to implement calculators for other forms of logging operations without further software development. This enhances the product, as well as adding to the academic interest of the project.

The requirements are grouped by the main components (web app, web editor, API), as well as common requirements for the system as a whole (deployment and security requirements). Each requirement is tagged with a code, consisting of initial letter of the component,

followed by requirement category and a number. The categories are as follows:

- F: Functional requirement
- NF: Non-functional requirement
- O: Operational requirement

3.2.1 Web App Requirements

The web app should:

W.F1	Let users calculate results while on- and offline.
W.F2	Let users store and retrieve results while on- and offline.
W.F3	Retrieve new or updated formulas for calculation automatically.
W.F4	Let users share, save and retrieve calculations.
W.F5	Let advanced users do calculations via an API.
W.F6	Keep calculation results visible at all times.
W.NF1	Have UI complying with Skogkurs' color scheme.
W.NF2	Be available on desktop, tab and mobile.
W.NF3	Support availability tools, such as screen readers.
W.NF4	Have Norwegian as UI language.
W.O1	Let the user perform a calculation without noticeable delay.

Table 3.1: Web app requirements

3.2.2 Web editor requirements

The web editor should:

E.F1	Provide a mechanism for declaration of dropdown inputs and number inputs
E.F2	Provide declaration of visualizations of results, such as tables, graphs and pie charts.
E.F3	Provide a manner of designing math functions that connect inputs to visualizations
E.F4	Provide a manner of structuring the order and grouping of inputs and displays on different screen sizes
E.NF1	Be easy to learn for Skogkurs employees

Table 3.2: Major requirements of the editor

3.2.3 API requirements

The API should:

A.NF1	Use HTTPS for data transmission
A.NF2	Provide responses in JSON format

Table 3.3: API requirements

3.2.4 Security requirements

The software should:

S.NF1	Have security built in from the beginning.
S.NF2	Validate all user input data before processing.
S.NF3	Offer a web app open and available without authentication to all users.
S.NF4	Handle secrets/keys separately from the source code.
S.NF5	Use monitoring to detect emerging vulnerabilities during development and in production.
S.NF6	Enforce authentication for editor users.
S.NF7	Use password-less authentication for login to editor.
S.NF8	Check authentication before every privileged operation.
S.NF9	Offer minimum necessary privileges for users for all operations.

Table 3.4: Security requirements

3.2.5 Deployment requirements

The software should:

D.O1	Be cloud hosted.
D.O2	Be reachable from Skogkurs' existing web site.
D.O3	Maintain 99% uptime.
D.O4	Scale dynamically according to number of users.

Table 3.5: Deployment requirements

3.2.6 Cloud Provider

This project was given complete freedom to choose a cloud provider. As part of the requirements gathering, we looked into whether there were any constraints and partnerships

we had to be aware of, so as to not cause unnecessary administration for Skogkurs. As it turns out, all their existing websites use platform-specific hosting solutions (Wordpress, Shinyapps, Sharepoint, etc.), which meant that Skogkurs did not have a preferred partnership, and left the decision to us. The main criteria for choosing a vendor was:

Server-less architecture: The solution should not impose any server maintenance on Skogkurs. Furthermore it must be highly scalable. Either PaaS or FaaS are considered acceptable options, but PaaS is slightly favored during development, and the requirements are subjected to change.

Node.js 20.x runtime environment: While javascript isn't the most ideal language for an API, it's justified by the need for sharing code across the SPAs and backend. This allows the web app to perform calculations, even when users are offline; and the API to be integrated to partnering systems.

Integration of database: The service must have persistent storage of previous calculator versions, in order to avoid breaking existing API calls whenever there is an update to the current calculator. A document-oriented database is preferred, especially during development. The main benefits include the ease of expanding types/features and the ability to work with plain JavaScript objects without serialization and/or deal with composition queries.

Authentication Engine: Publishing new calculator versions will naturally require authorization. In order to reduce the burden of maintenance and security risk, integrating a 3rd party authentication authority and session manager, such as Identity-as-a-Service (IDaaS) would be ideal.

Automatic Certificate Renewal: Each TLS certificate should be set for automatic renewal to reduce the risk of service interruptions that can occur if not renewed every two years.

Single Provider: For ease of billing and administration, it would be best to centralize all components to a single provider. This also simplifies the use of service accounts when dealing with authentication across platforms.

3.3 Use Case Diagram

3.3.1 Actors

- **App user:** The intended end user of the calculator web app, for example a forestry manager performing a cost calculation to decide whether or not to harvest.

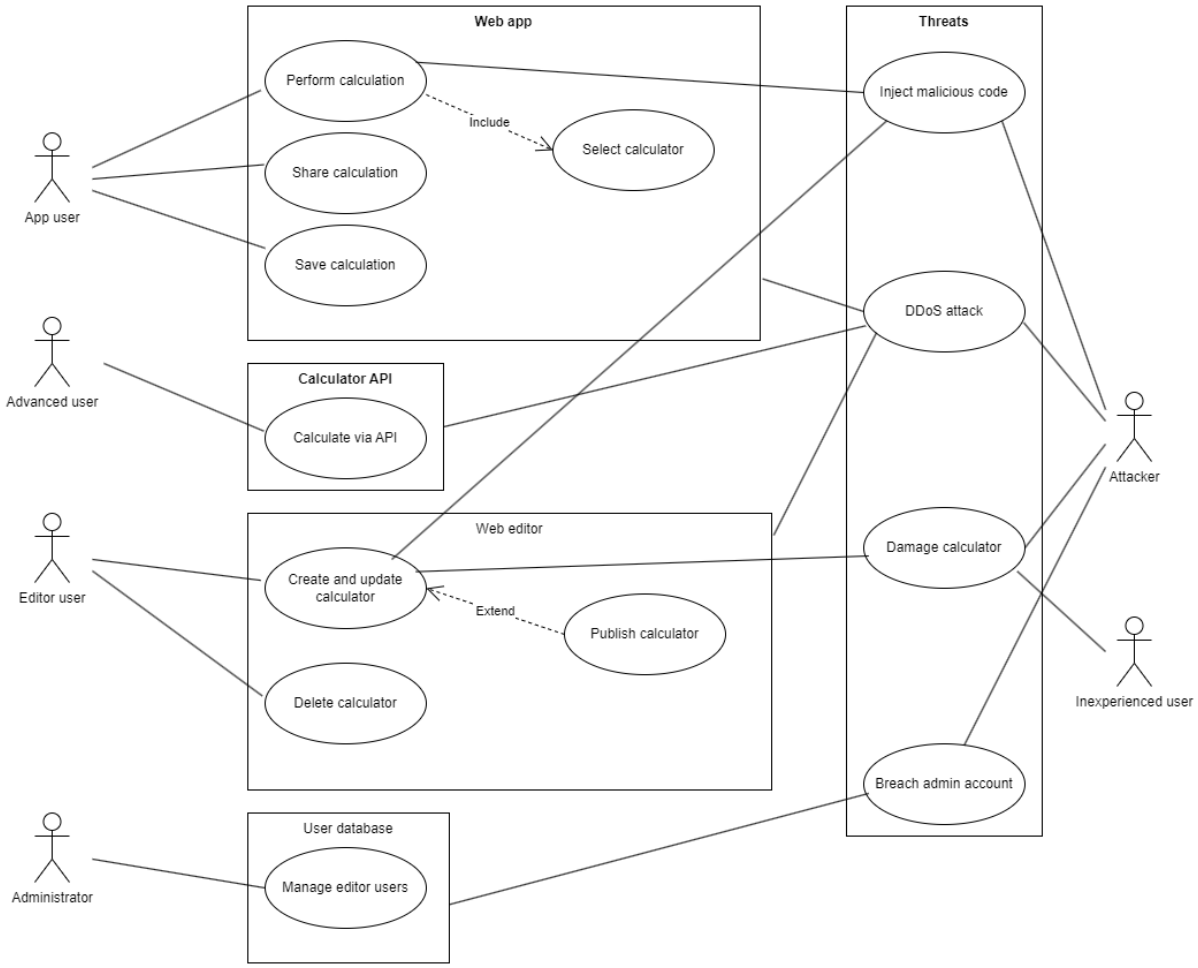


Figure 3.2: Use/abuse case diagram

- **Advanced user**: A user that utilizes the API through an external system, rather than in the web app. Typically an IT professional in a larger forestry organization.
- **Editor user:** The intended user of the web editor. The editor user is responsible for creating, updating and publishing calculators. In practice a Skogkurs employee.
- **Inexperienced user:** A new editor user unfamiliar with the web editor's workflow.
- **Attacker:** Anyone who wants to harm the application and its assets.

3.3.2 Use case examples

Use case	Perform calculation
Actor	App user
Goal	Calculate cost of forestry operations
Description	App user fills in own values in some or all input fields. Main results are viewed in real time. More comprehensive results are viewed on user's request.

Table 3.6: App user, example

Use case	Perform calculationCalculate via API
Actor	Advanced user
Goal	Perform calculation using an external system
Description	User utilizes https requests containing calculator version and all necessary input data to perform calculation. Results are returned in a machine readable format.

Table 3.7: Advanced user, example

Use case	Create and update calculator
Actor	Editor user
Goal	Create new and update existing calculator
Description	Via a calculator editor, the Editor user can create a new calculator. A set of math operations, input types and result displays are available. Editor user may choose to save calculator, or save and publish. For previously saved calculators, Editor user may apply changes and updates. At update, the calculator should be given a new version number. All Editor user operations require that the user is logged in.

Table 3.8: Editor user, example

Abuse case	Damage calculator
Actor	Attacker, inexperienced user
Goal	Overwrite stored calculators
Description	Attacker may get write access in calculator editor client, and change existing calculators on purpose to make them erroneous or unusable. An inexperienced user may overwrite a previously stored calculator by accident.

Table 3.9: Abuse case, example

These examples serve as an illustration of how the different use cases are described. Main tasks for the different user groups are described to help us get an overview of overall functionality of the software. The complete use case diagram with use case descriptions is appended in appendix F.

3.4 Domain Model

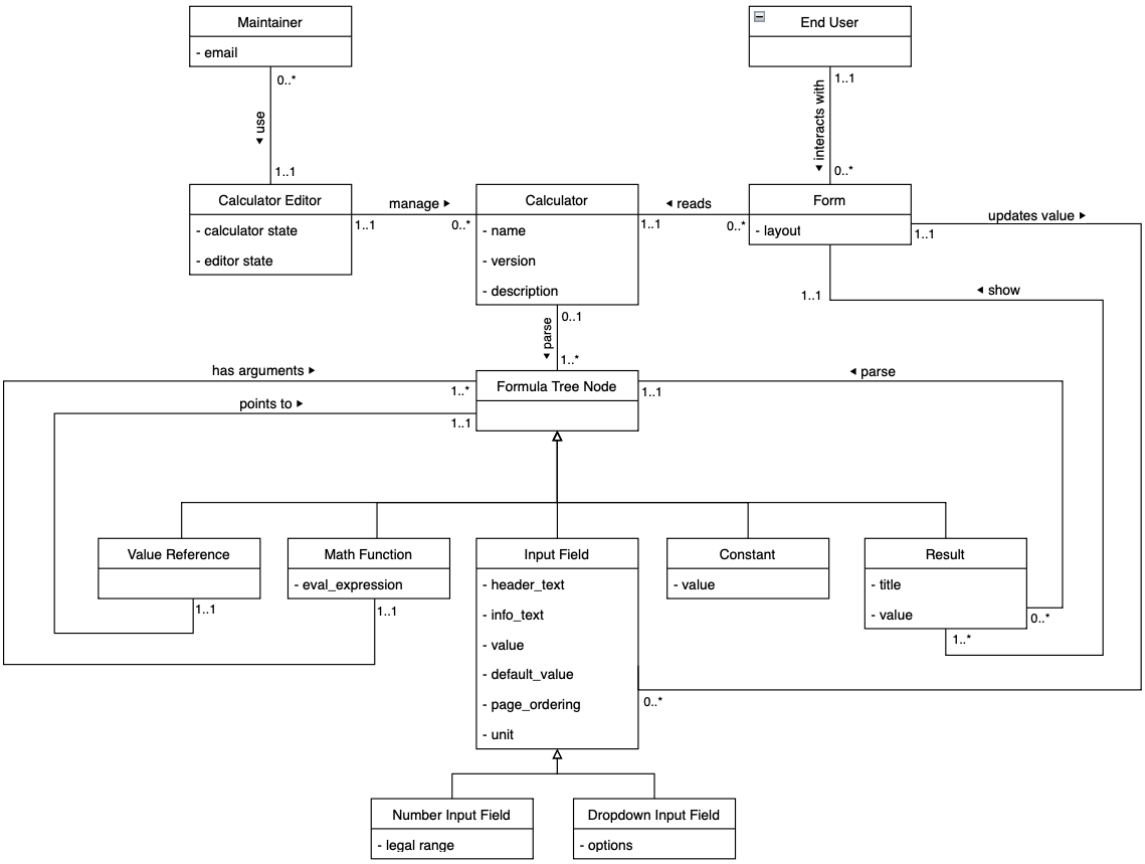


Figure 3.3: Domain model

Although this domain model shares some similarities with the actual implementation, it is purely conceptual and has been intentionally simplified to facilitate a quick understanding of the concepts. Consequently, this model should not be considered comprehensive and may be misleading if interpreted as reflective of the implemented code. For instance, while the tree structure (derived from a *Calculator* entity) is mostly accurate, complexities such as modules and additional node types like displays have been omitted.

The end-users interacting with the *Form* entity remain anonymous, ensuring that no user information is collected. However, editors (or *Maintainers*) use pre-approved email addresses to access the system through a passwordless mechanism commonly referred to as email links.

The *Form* entity represents the user interface, with which users interact. This *Form* gathers instructions on how to render the UI from the *Input Field* and *Result* entities. As tree nodes are updated via the *Form*, the results are fed back into it. While the *Form* can accommodate multiple calculators, it can only process one at a time. These calculators, predefined by the *Calculator Editor*, include versioning that is crucial for historical tracking and particularly vital for users accessing through the API.

Chapter 4

Development Process

This chapter will briefly recapitulate our software development model as outlined in the project plan, and offer insights and critiques on how its actual utilization differed from its conception in the planning phase, as well as critiquing the development process more broadly.

4.1 Scrum

As described in the project plan we used Scrum [13] as our development model, adapting and tweaking it to suit the particular needs of our case.

4.1.1 Choice of Development Model

From the inception of the development process it was clear that Skogkurs' requirements for the software was defined as overall functionality on a conceptual level (see Appendix D). We foresaw that refining this into actionable work items would require an iterative approach, responding to new challenges and requirements as they were uncovered. This made an agile development model ideal for our purposes, and given that we had experience with adapting the Scrum model to suit our needs as a group, Scrum was the obvious choice.

Another advantage is that the structured process of sprints made coordinating the group easier. While the meeting time required for sprint planning, review and retrospectives could be considered a distraction from the development work itself, our previous experiences with Scrum impressed upon us the importance of coordinating the team's effort across the branching fields of development, testing and documentation that this project would require.

Alternatives

While we did not conduct a robust evaluation of other development models to see whether they would be a good fit for our use case, we will briefly address alternatives to Scrum that

may have been utilized in its place, primarily **Kanban**[14].

The iterative, agile approach of Kanban would perhaps have been suitable for our needs. However, given our inexperience with practical usage of the model, and the emphasis Kanban places on planning only those work items which are currently in progress, we felt that this would leave too large a possibility for mismanagement of the team's resources.

4.1.2 Sprint structure

The structure of the sprints corresponded for the most part with the model as outlined in the project plan (see Appendix E). The following gives a brief retrospective of the Scrum process as actually utilized throughout the brief structure, focusing primarily on the sprint activities, touching briefly on sprint lengths. See table 4.1 for a typical sprint schedule.

Sprint length

To more frequently receive, process and formulate work items based on feedback, the first five sprints were one week long, consisting of 18 work hours and 10 hours of meetings.

Once a clearer picture of what development would entail, and to facilitate spending more time on implementation and less time spent in meetings, sprints 6 through 10 were redefined at two weeks, consisting of 43 work hours and 13 hours of meetings.

The 11^{th} sprint was defined at three weeks, with a less rigid distinction between work hours and meetings. The last sprint was spent primarily in writing the actual thesis paper, and meetings between the team members were held as needed to coordinate that effort. The number of hours was also deliberately kept undefined, being decided by how many hours were required to the finish the paper.

Sprint activities

- **Standup:** The daily standup, ostensibly intended to serve as a short meeting where the team members updated each other on their progress and plans for the day. Very frequently they far exceeded their 15-minute time slot, and were often used as ad-hoc mini-seminars, where technical insights were shared. Standups were also frequently used to refine product requirements and discuss the next steps in the development process.
- **Backlog grooming:** weekly meeting where we sat down with our primary contact from Skogkurs, the POs, to refine the contents of the product backlog. The intent of these meetings were to prioritize the items in the product backlog and define a goal for the coming sprint, though product backlog items (PBIs) were also generated and reworded as necessary.

Table 4.1: Typical two week sprint condensed into one for brevity

Time-slot	Monday	Tuesday	Wednesday	Thursday	Friday
		Sprint-End	Sprint-start		
09.00	Backlog grooming	Standup	Sprint-planning	Other courses	Standup
10.00					
11.00					
12.00	Lunch	Lunch	Lunch		Lunch
13.00		External review	Supervisor meeting		
14.00		Sprint retrospective			
15.00		Internal review			
16.00					

- **Sprint planning**: once-per-sprint meetings where PBIs were selected from the backlog to satisfy the coming sprint's business goals. The team-members then estimated their capacity for the sprint, and PBIs were estimated for time and complexity (see 4.1.3 for elaboration). The estimation and selection of PBIs repeated until the sprint's estimated capacity for work was reached. These meetings deviated very little, if at all, from their description in the project plan.
- **External review:** a once-per-sprint meeting where progress on the application was demonstrated for the external stakeholders and Skogkurs. Meant as an avenue for the stakeholders to give feedback and suggest improvements, but were just as frequently used as an arena for the stakeholders and Skogkurs to discuss forestry-related concerns beyond our scope and level of knowledge.
- **Internal review**: once-per-sprint meetings where the team members reviewed the status of the given sprint's PBIs, to determine which PBIs can be closed, and which ones must be transferred to the upcoming sprint. Also intended to be used for knowledge sharing within the group, but this was often made obsolete by the daily standups.
- **Sprint retrospective:** once-per-sprint meetings where the group would reflect on how the development process could be improved based on experiences from a given sprint. Group members were intended to prepare suggestions beforehand, but this happened only infrequently, and thus these meetings often ended up with only a very short list of improvements.

The most notable divergence from the project plan is in the roles of the standup and

the internal review, where the standup absorbed some of the purpose of the internal review (see listing of sprint activities above).

The purpose of the internal review was to avoid any single team member becoming too unfamiliar with the code base, or broader status of the project. Based on our experiences from earlier projects, the consequences of falling out of touch with the development progress could hamper a team member's ability to contribute to development. Avoiding this through sharing knowledge and insights between members was part of the motivation for the internal review. Because the standups fulfilled this purpose, we do not consider this divergence from the project plan to be particularly problematic.

4.1.3 Selection of work items and sprint capacity

As stated in the project plan, work items in the form of product backlog items (PBIs) were distributed among the team member on the basis of priority in the backlog and the motivation of the individual team members. I.e., if someone wanted to work on the web application UI, and a PBI related to that was placed sufficiently high in the product backlog, then they were free to do so.

As previously mentioned we defined a sprint's capacity as 18 hours of working time (the time not dedicated to meetings) per person, which we divided into 6 storypoints for the sake of estimating the workload of a given PBI, each story point roughly equating to three hours or half a days work. PBIs were then assigned a number of storypoints linearly, according to its estimated time to complete. I.e., a PBI estimated at 1 one story point would require half a workday, a 2 story point PBI would require a full day, 4 story point PBIs would require 2 days and so.

This is somewhat unusual in Scrum [15], where story points are often defined exponentially in a Fibonacci sequence.

After some discussion we found this approach to be quite time-consuming for estimating work items, and therefore settled on our linear approach. This choice was also influenced by our experiences from an earlier project where rigidly estimating the time required to complete work items required a great deal of time with little payoff in terms of increased productivity.

4.2 Co-operation with Skogkurs

During the entire project period we maintained close co-operation with the project owners at Skogkurs, and were given the opportunity of using their offices in Brumunddal as our primary physical location.

This physical closeness, as well as the relatively frequent meetings (weekly backlog-grooming and bi-weekly external sprint reviews), meant that both receiving feedback on the product, as well as discussing the scope and domain-specific considerations outside of our area of expertise, was easily facilitated. These interactions, it should be noted, also happened outside of the regularly scheduled meetings, and therefore some of the important discussions and decisions that occurred there have gone without documenting in meeting minutes and the like.

4.3 Project management

At the start of the project period we made a Gantt chart to roughly track when and in what order the various stages of planning, development and documentation should occur (see fig. 4.1).

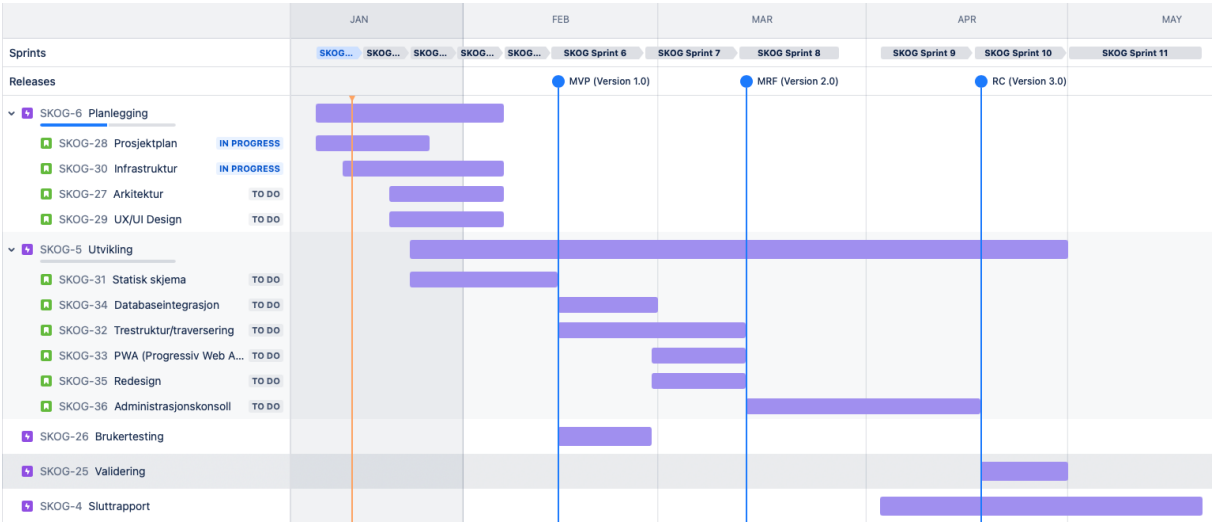


Figure 4.1: Gantt-chart, from the project plan

To allow for iterative specification of requirements that was deemed beneficial to the development process (see 4.1.1), the outline as described in the chart was never meant to be anything other than a rough guideline to help structure the work throughout the project period. Consequently, we have not performed a thorough evaluation of whether or not it proved a realistic estimation of progress, but it proved a valuable tool for broadly defining a path towards project completion.

In addition to a broad framework of the project itself we also set three deadlines to track major milestones of the development of the application:

- **MVP**: the minimally viable product, a functional version of the calculator, migrated from Excel spreadsheet to a web application.

- **MRF**: minimum releasable features, an updated version of the MVP, with a redesigned interface integrating feedback from the stakeholders, including the ability to see results change in real times as users update input parameters.
- **RC**: release candidate, the dynamic version of the software, where formulas are generated in the web editor, and displayed and calculated in the web app.

These milestone characterizations, apart from the MVP, were defined mostly post hoc. While the dates were decided upon at an early stage, the actual requirements for each major iteration was kept purposefully open-ended. This would allow us to adjust the direction the development process as requirements were discovered in collaboration with the PO.

Chapter 5

Technical Design

This chapter handles the system architecture as a whole, design alternatives and choices, as well as other details of the design process not tied to a particular framework or library.

5.1 System Architecture

The most predominant constraint in the system design derives from the requirement for offline support; which means that the web app must be capable of performing calculations without internet access. Furthermore, Skogkurs must be able to make changes to the calculator with minimal effort, thus ensuring that all end users have up-to-date versions. This makes packaging and distribution a challenge.

To complicate things further; because users should be able to link and recover past calculations, the application must ensure that changes to the underlying mathematical formula or input fields do not alter the result of prior calculations, thus making them unreliable and untrustworthy. Generally, there are two solutions to this problem: either export the result as a read-only report, which means the calculation cannot be adjusted in the future, or create a data structure that describes the entire calculation process, from input to output, and append the versioning number to the result link as metadata.

We went with the latter approach. By extracting most of the business logic, the application is reduced to a "skeleton" that seldom require maintenance nor updates, which reduces the frequency for repackaging and distribution. Instead, updates and requests to calculator versioning can be fetched through API calls and cached on the end user device.

5.2 Web App

The web application is developed as a small, single-page application (SPA), which despite the name, doesn't mean that web app consist of a single page. Rather, it means the entire

skeleton code (including HTML, CSS, JavaScript, images, etc.) for the application is loaded into the browser upon first visit. Subsequent navigation/interaction in the app changes the content dynamically in the browser's DOM without requiring the page to reload.

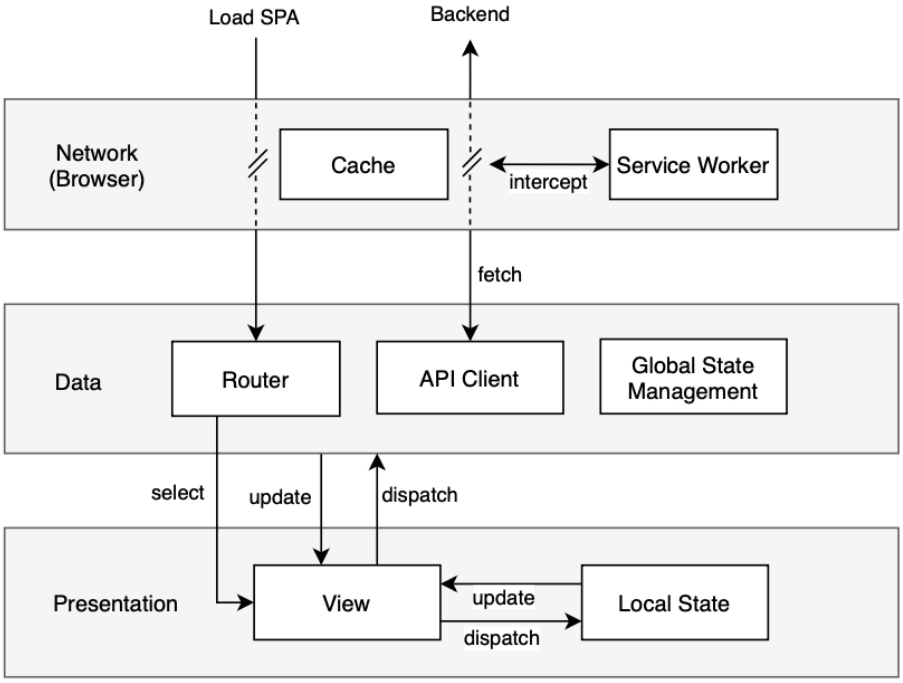


Figure 5.1: Abstract Web App Architecture

Figure 5.1 describes the architecture of the web application at a level of abstraction that leaves out many of the mechanisms related to implementation. Most apparent is the cache component in the network layer, which is intentionally left ambiguous. Caching strategies are very complex, and the mechanisms depend on the libraries and technologies used, so caching is described in greater detail the implementation chapter, section 7.2.2.

This layered model is conceptual and does not build on any specific reference model. The presentation layer contains the views, which define the composable UI components that make up what is visible on the screen. A DOM-router in the data layer act as a navigation controller and determines which view is currently active, based on the URL or dispatched navigation calls. Each UI component also have a non-persistent state store associated with their life cycle, which is discarded when the component is unloaded. This simplifies global state management and prevents the emergence of stale data.

The data flow between the presentation layer and the data layer is unidirectional, making it easier to reason about state updates. The data layer does not only organize state, it is also capable of performing HTTP requests to the backend, so that specific calculator versions can be retrieved periodically or on-demand. Furthermore, a service worker (see 2.3.1 running in a separate thread, is registered to intercept incoming and outgoing requests, effectively acting as proxy server. This enables the web app to use the browser's caching capabilities to

prefetch data and store earlier HTTP responses, to achieve higher level of performance and availability.

In essence, the web app's objective is simply to parse and present calculators (which describe inputs and outputs) from the database. Once downloaded, the calculators are stored persistently on the end-users device, and can be brought up with the web app, regardless of service/network availability. It may be observed that the business logic for mathematical operations, on which the calculator relies, is absent in figure 5.1. This functionality has been isolated into a shared package called 'common', which is then imported as a dependency to the global state management module.

The skeleton code, service worker, manifest file, and app icons are seamlessly integrated to create a Progressive Web App (PWA) by the build process, that can be installed to the home screen of any modern device. While some browsers may alert users to the installation option, others may not. Nevertheless, the offline functionality works consistently in the browser regardless of whether the user has installed the app or not.

5.3 Editor

Where the web *app* has the responsibility of presenting the underlying data structure to the user as a calculation tool, the *editor* is responsible for the creation of the actual data structure, allowing Skogkurs to edit and create new calculator tools that can be accessed in the web app.

In this section we will go over the editor's design goals, alternatives and choices made throughout the project. This section will focus on design irrespective of actual implementation such as libraries and languages, while section 7.3 will go into more detail on how the design detailed in this section has been implemented.

5.3.1 Goal

The primary goal in designing the editor was to present the user with a tool where they could alter or create new calculators for publishing on the end-user client, ideally with none or minimal knowledge of how web pages work. The only prerequisite for producing new content would be knowledge of basic mathematics and the basic layout of the app itself. Lowering the requirements of creating new content in this way let us separate the concern of *what* is to be visualized from *how* the presentation is implemented. Developers can focus on developing tooling, infrastructure and presentation while others can focus on the content to be displayed.

With this in mind we deemed it important that the user is provided with a way of previewing the presentation of content they are working on, with the idea that structuring of

the content, such as order of form fields or order and width of result displays in a grid, would have a low learning curve through short feedback loops. Even if the user doesn't immediately understand what a setting is for, they can see the effect of changing it by observing the preview in the GUI.

5.3.2 Possible Solutions

When one thinks of an editor, one usually thinks of the classic layout seen in software such as Microsoft Word, Adobe Photoshop and others with a centered canvas, be it with a drawing, text or a 3D model, and various tool panels around the central view. It's a commonplace pattern, and below editors adhering to this pattern will be referred to as *traditional editors*.

Another solution, as discussed in 2.2, is a node based UI. Such editors usually have niche applications due to being practical for some tasks, like designing a machine learning model with its layers, while being nearly useless for other tasks such as editing text. While there's not a black and white separation of the two concepts, editors that mainly rely on a node system to edit some data will be referred to as *node based editors* for the sake of simplicity.

A third option that was never truly considered was simply forgoing a graphical editor altogether in favor of implementing a proprietary script language, or modifying an existing one, that would let the user declare objects for inputs and outputs, along with basic math functions. The reason for its exclusion is the fact that we desired to allow non-programmers to edit the content.

Traditional editors

The problem of separating content creation and code implementation is a solved one with respect to forms and calculators. Altinn Studio[1] is a tool used by Altinn to let non-developers create new forms, whereas ConvertCalculator[2] is a Software as a Service solution allowing users to create calculators, questionnaires and other solutions for their needs. They both work on a somewhat similar principle. The user is provided a blank canvas and sets of drag and drop elements, such as input fields, visualizations such as graphs or histograms and other components like headers and paragraphs.

Upon dragging an element into the canvas they are able to apply some styling through settings in a side bar, as shown in figure 5.2. There are separate tool windows for math, conditional logic for display behaviour and other aspects of the element. Upon publishing, it is hosted by Convertcalculator and made available for embedding in your web site.

This type of GUI can have a good deal of flexibility, but the learning curve can be said to be steeper due to the added complexity. In the case of ConvertCalculator you're also

[1] https://github.com/Altinn/altinn-studio
[2] https://www.convertcalculator.com/

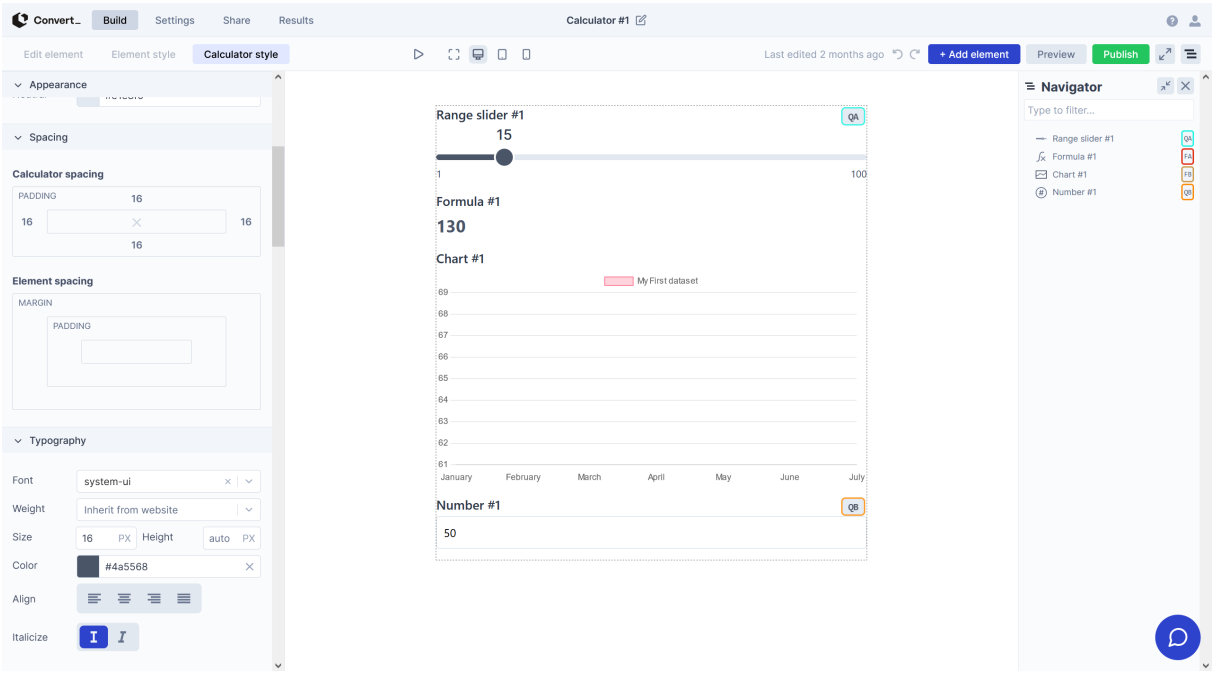


Figure 5.2: ConvertCalculator: Styling of an element

expected to know basics of CSS for styling purposes.

Node Based Editors

Another pattern used in applications that aim to make scripting or otherwise complex tasks simple is to give some sort of graphical representation of the underlying data flow or functionality. Applications such as Blender or Unity utilize node based visualization to let the user define flow of information in an intuitive fashion. Nodes demand some input for some task, then provide an output to let the user pass on data to another node, like shown in 2.3.

An advantage is that the relation between an input, where it is used, and where it ends up is usually immediately obvious just from looking at the visualized graph as flow of information is represented by the edges going from node to node, and the appropriate type of input and output is indicated by color and text on any particular connection point.

Node vs traditional

The traditional GUI works. It's a true and tested solution with commercial use within the domain we were working in, though there were worries that such a solution would involve a lot of custom components, while a handful of us were not very UI savvy. A primary concern was the mathematical representation. In ConvertCalculator an Excel like syntax was used to represent functions, and while off the shelf general purpose math parsing libraries are available, in all likelihood we would have to write something custom to fit our needs in terms of math to data representation, and properly implementing such a thing with proper

highlighting of errors might be no small feat.

5.3.3 Design Choice

Regardless of whether we had chosen a more traditional approach or a node based design at this point, there would be several uncertainties in play. For the traditional approach: Would we have enough UX know how to make it work well, and would implementing a basic math parser with syntax highlighting be feasible? For the node based approach: Would the node based layout actually be beneficial for the user, and how complex would it be to implement?

After considering these aspects we settled on a node based approach, as:

1. UX design was assumed to be easier as node based systems often have a very utilitarian and easily composable design, usually a column with elements.
2. Nodes could provide flexibility in adding new features. Adding a new feature could require as little as adding a new node that could connect to appropriate nodes.
3. A node graph could likely easily be serialized and converted into a data format appropriate for display in the app.
4. A node based solution struck us as the most interesting option of the two.

5.3.4 Architecture

As the architecture ended up being highly specific to the implementation, it is detailed in the implementation chapter. See 7.3.2

5.3.5 The Editor and Nodes

One of the first things we did in the project after moving towards the idea of representing the web app as a graph was to perform an analysis of the Skogkurs formula, constructing a binary expression tree diagram as can be seen in appendix O. Based on the analysis, as well as the design work done in the web app up to that point, we identified a set of nodes we would need to construct the data representation.

Input

Dropdown fields with text would be needed. Behind the scenes the selected field would resolve to a number, such as "Very poor" resolving to the number 5 before being passed on to the formula.

The other input type we would need would be number inputs that would let the user write any number, optionally within a set legal range, either natural numbers or decimal numbers.

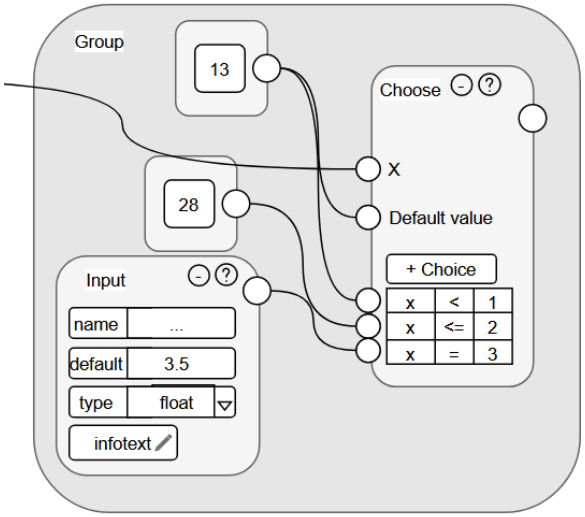


Figure 5.3: An early draft of a "choose" node.

Output and Display

Display nodes would be needed; Nodes containing graphical visualizations of results and configuration options. Additionally we needed labeling of results with name and color. These labeling nodes, or Outputs as we called them, would go between numeric nodes and displays.

Math

In addition to the inputs and outputs, math nodes were needed to build the function that would tie them together. For the most part this aspect was fairly straight forward, only requiring basic operations like arithmetic and some quality of life nodes like SUM and PROD to avoid long chains of addition or multiplication.

Conditionals

To be able to select forest type as in the original formula, there needed to be some form of conditional node. We landed on the design shown in figure 5.3 where the user could define any number of conditional statements, where the input of the first one to evaluate to true would be passed on by the node, more or less an "if else" chain where each branch evaluates to a value. Due to the function used in Excel being named Choose, we opted to use the same name for familiarity.

Modules

To combat the problem of increasingly complex graphs we opted to integrate what the rete.js documentation referred to as modules. Module nodes are nodes with inputs and outputs

that represent a concealed nested graph that can be edited and viewed on its own. This way, segments can be named and re-used where appropriate.

5.4 Backend

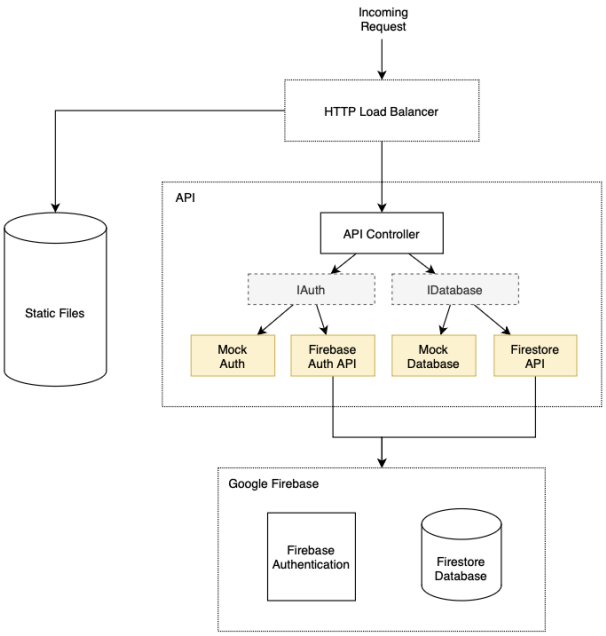


Figure 5.4: Backend Architecture

As per the requirements listed in section 3.2.6, the backend of the service is designed to be cloud-hosted in a serverless environment. The main components consist of:

1. API
2. Load balancer
3. Database
4. Authentication Service

5.4.1 API

For the sake of flexibility, the API is designed to be completely stateless, and can therefore effortlessly scale horizontally by starting additional instances. Furthermore, the system is loosely coupled to its dependencies (the database and the authentication service) through the use of interface types and dependency injection, which is illustrated in figure 5.4 with gray and yellow boxes, respectively.

The effect of dependency injection means that the API is resilient against environmental changes. Such as changing service vendor simply means writing a new implementation for

the affected interface, without the need to change any of the business logic. Each dependency is injected into the application through a configuration object at initialization. This also benefits testability, because mock implementations mimic actual implementations, so that the business logic can be tested in isolation from their underlying dependencies.

5.4.2 Load Balancer

The purpose of the load balancer is two fold: First and foremost, it accommodates for scalability and can direct traffic towards certain API builds. Secondly, the SPA web files are served directly from handlers defined at the load balancer, which lessens the load and yields better performance compared to routing requests for static files through the API.

5.4.3 Database

As of now, the database is only used for storing and retrieving calculators, including the web editor configuration for each respective calculator. From the perspective of the service, the database is entirely accessed through the API as a single point of entry (SPE), which means that the actual data structure is only relevant to the API consuming it. In this case it's a document database, which does not enforce a schema for maintaining consistency.

5.4.4 Authentication Service

Due to uncertainty in the amount of effort that can be spent on future maintenance, the project must be extra cautious with respect to risk appetite on identity and access management (IAM). Rather then carefully designing every security mechanism in the system, the best approach is to adopt a Identity-as-a-Service (IDaaS) solution. Figure 5.5 illustrates the division of responsibilities in such a system, where the IDaaS provider acts as a dependable intermediary between the web app and the API. The web app communicates directly with the IDaaS provider to manage its authentication token, which is then verified by the same IDaaS provider for privileged API calls. Additionally, SDKs cover the entire lifecycle of the tokens, from secure storage to updates; effectively reducing the number of security mechanisms for the system.

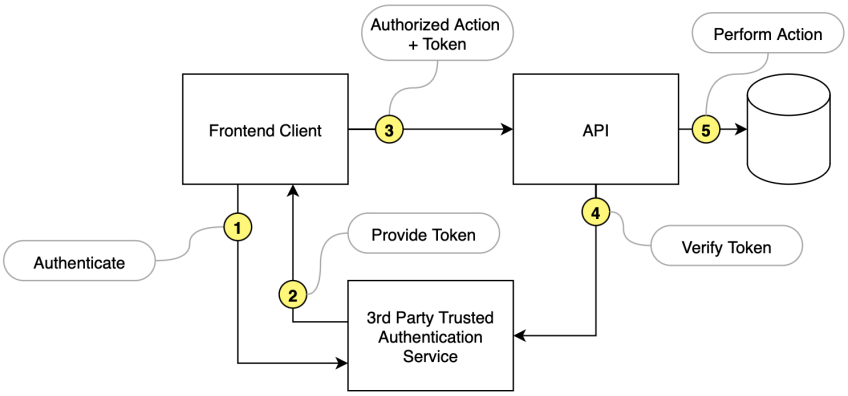


Figure 5.5: Abstract use of "Identity as a Service" (IDaaS)

Chapter 6

Graphic Design

In this section we will discuss and reflect on the methodology utilized in designing the UI of the web app, changes done based on findings from user testing, and how interacting with stakeholders, users and Skogkurs employees informed design decisions made throughout the project.

6.1 Process description

6.1.1 Overall guidelines from Skogkurs

Few direct guidelines were given, apart the transition from excel sheet into a web app. This meant a high degree of freedom with regard to design. Some high level design requirements, as well as a few details, emerged over the course of the project. UI requirements are listed in table 3.1.

6.1.2 Sources of feedback

Throughout the project, design feedback has been gathered through multiple sources. A significant amount of feedback was gathered by talking to Skogkurs employees at the office, in addition to the more formal sources listed:

- **Skogkurs Project manager:** Main source of feedback through project owner meetings. Weekly meetings gave insights into what was most important from Skogkurs' perspective, with regard to features and general progress. This was balanced against our onw progress and motivations at each stage of the development process.
- **External stakeholders:** A group of forestry professionals representing the intended users were given the opportunity to influence the application's development to better suit their needs. Discussions with users with first hand knowledge from the forestry

domain helped us better understand the needs and challenges of developing the application.

- **Skogkurs UX designer:** A design professional employed at Skogkurs was consulted in an early design stage for feedback on navigation in the app. Later some guidance was given for compliance with Skogkurs' design template. The designer also created the app logo. Dicussing with a design professional was helpful because we as programmers tend to focus more on the technical aspects, rather than the human centered.
- **User testing:** For a more systematic evaluation of the application, usability testing was performed as described in section 8.2.

6.1.3 Design decisions

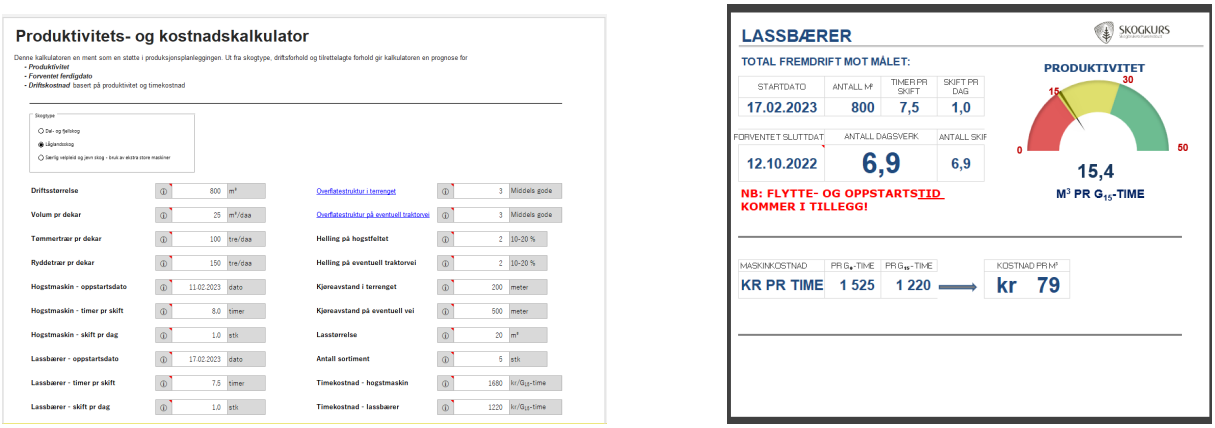
Decisions of changes to the design has been dynamic. By that we mean that small changes requested has been implemented continuously. The process of deciding what feedback to implement and what to ignore has largely been informal. Decisions have been based on a combination of feedback and our own views of what would make a better user experience. Ideally, we should have made a design document where requests and decisions could be logged. This would have helped us better document the changes done and the reasoning behind each design choice.

Furthermore, the visual design is somewhat restricted by the chosen CSS framework (Bootstrap). Many components are ready made out of the box, but we have had to create several components ourselves to get the layout and design the way we want. Especially, color schemes proved challenging to implement in Bootstrap. Alternative frameworks are available, such as "Material Design" [1]. Much time and effort has been put into creating reusable components in Bootstrap.

6.2 Web App

6.2.1 Design iterations

The sections below intend to describe the evolution from spread sheet into a web application from a design point of view. A full overview of all design iterations and changes would be too comprehensive. The application's design at select important stages is described, along with examples of details that have been explored, added or evolved throughout the project.



(a) Input page **(b)** Result visualisation

Figure 6.1: Original excel sheet (excerpt)

Original solution

An excerpt of the original calculator can be seen in fig. 6.1. The presentation of information on the input page (6.1a) is clean and tidy, but the number of user inputs simultaneously shown may overwhelm the user. The result page (6.1b) has information in multiple formats, making it difficult to understand what is most important. This indicated that when already doing the transition from spread sheet to web app, a redesign of the user interface was also a necessity. Some of the original inputs and results were deemed unnecessary by Skogkurs, rendering especially the result page more readable.

Organization of information The initial plan for organizing the information was as follows:

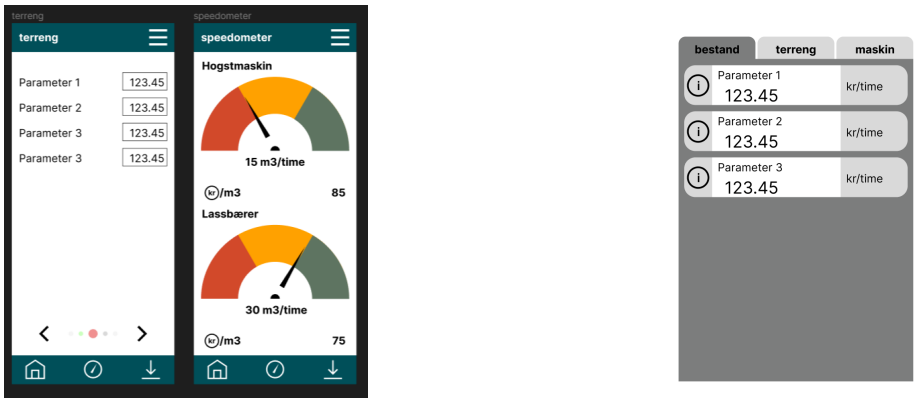
- Input pages: Group inputs belonging together in the same page
- Result page: An own dedicated page to view results, both numerically and visually
- Landing page: A starting page to select calculator version
- Info page: Information about the calculator and its use
- "Hamburger" menu: For links to additional information, like the scientific basis for the calculator

Prototypes

To maintain familiarity, the layout of each input field was kept unchanged; an info button, a value and a unit. The "speedometer" visualisation from results page was also kept.

As the finished product was required to be available on multiple devices, a "mobile first" approach was used. This is sensible because presenting enough information on a small screen can be more difficult than on larger screens. Moreover, mobile units are often the

[1]https://m3.material.io/



(a) Early prototype **(b)** Prototype with tabs

Figure 6.2: Design prototypes

preferred device and we expected that the majority of users would use the mobile version. The specific challenge of too many inputs was solved by splitting the inputs into three groups (forest, terrain, machines).

An interactive prototype was created using Figma [2], a web based tool for designing user interfaces. The first prototype, shown in fig. 6.2a, used a drop down menu to navigate between input pages, using the "hamburger" button shown to open said menu. The reasoning behind using a drop down was that the navigation would occupy minimal space unless actively in use. To clearly signify a different purpose from the inputs, a separate button in the bottom toolbar was used for navigating to the results.

Arrow navigation with page indicators was also briefly explored. This combined with the drop down menu was considered to allow a flexible navigation, letting the user follow a linear workflow, while also offering the possibility of going back to a specific page. At the prototyping stage, exploration of other navigation modes was also done. Most notably, navigation tabs were considered. A low fidelity prototype was made (fig. 6.2b), to test the idea internally. An obvious advantage of using tabs is that all navigation alternatives are visible at all times. A drawback is that, especially on a small screen, the number of tabs must be limited. In this case, more than four tabs seemed problematic, and therefore this idea was dropped for the time being.

MVP

Partially in parallel with the prototype created in Figma, an early version was implemented in TypeScript/React, using React-Bootstrap[3] for styling. React-Bootstrap offers several UI components which can be used and modified to build a UI with a consistent look, without the need to write complicated css. Complex css-styling from scratch would be very time

[2]https://www.figma.com
[3]https://react-bootstrap.netlify.app/

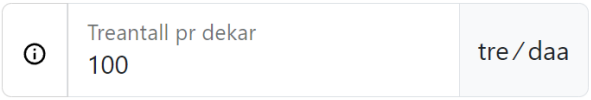
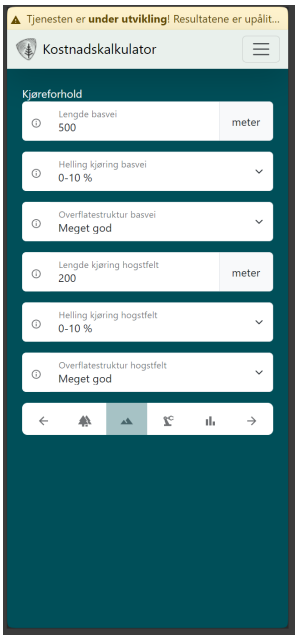
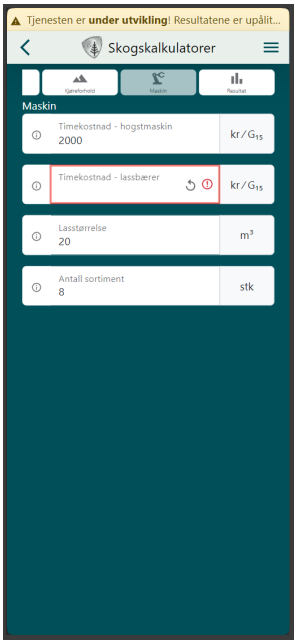


Figure 6.3: Example input



(a) Button bar



(b) Scrollable button bar

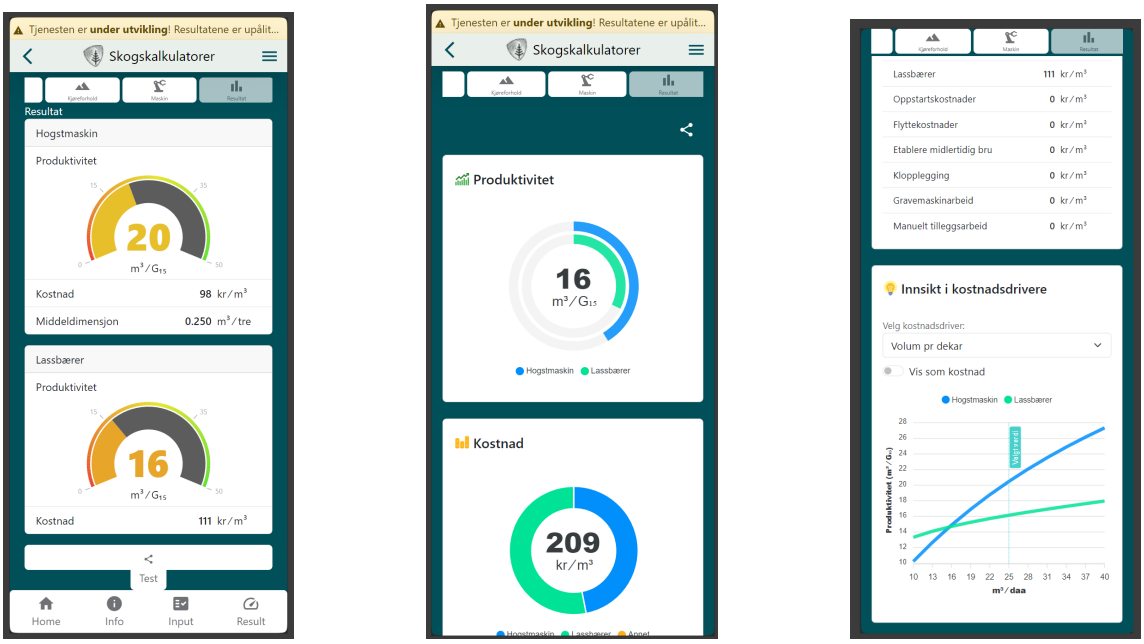
Figure 6.4: Navigation modes

consuming, and we decided to put the effort elsewhere.

Number and drop down inputs were created in the same style as earlier, see fig. 6.3. An info button is available for each input, triggering a text box with additional information. Additionally, the inputs are validated, turning red to signal the user if invalid values are typed. The input fields remained mainly unchanged throughout the project.

As the work progressed, new navigation modes were tried out. Bootstrap offered a pagination bar component, with numbered buttons for each individual page, as well as arrows for incremental navigation. To better communicate to the user which content to expect on each page, the number labels were replaced by icons, shown in figure 6.4a. A challenge when working in such a specific domain as forestry is that universal icons for the different categories do not exist. In a meeting with Skogkurs' own UX designer, we were advised either use text only, or as a supplement to the icons chosen. Introducing text in the buttons meant that each button increased in size. It was therefore decided to make the navigation bar scrollable, see fig. 6.4b. This way, an arbitrary number of buttons could be added, while leaving enough room for descriptive texts.

It should also be noted that the navigation bar was moved to the top of the screen, after conferring with the UX designer at Skogkurs. A main problem was that the bar followed the



(a) Result visualization, early **(b)** Result visualization **(c)** Graph result

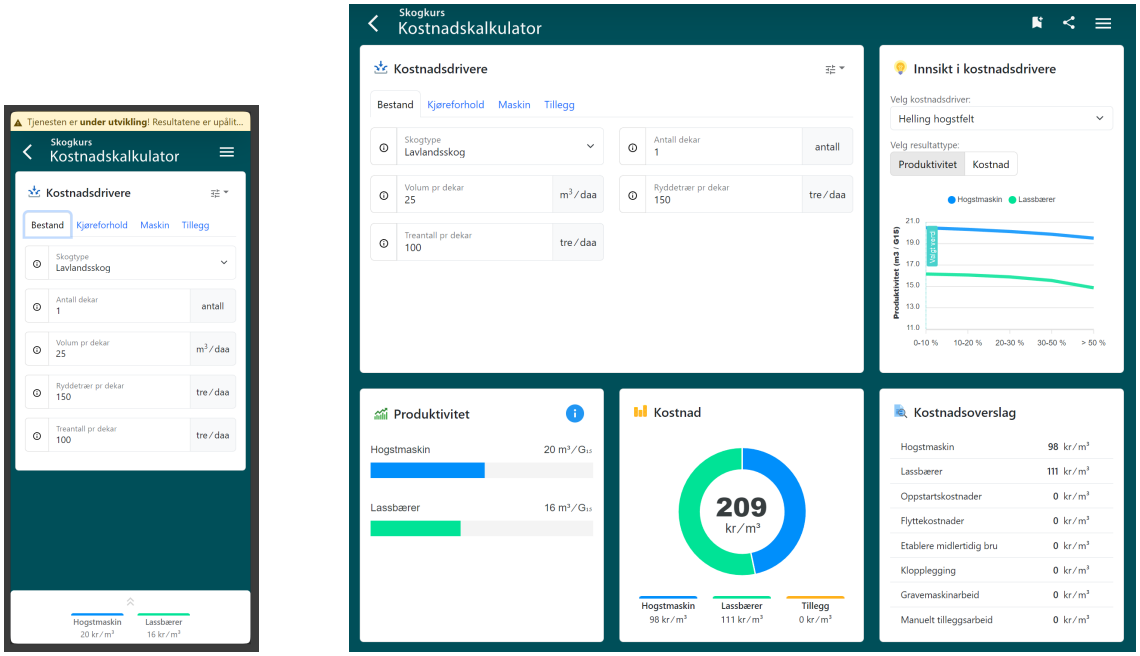
Figure 6.5: Result visualizations

size of the content as it changed. Therefore, the decision was made to make it stick to the top of the screen instead. User testing was performed on this version, and most of the users seemed to like the navigation. Some users however had trouble initially with finding the buttons hidden by scrolling.

The result page also evolved in this version. From a more or less direct copy of the visualisation from the spreadsheet with graphic representation of productivity and numerical value for cost, both of these were now visualized separately, as illustrated in fig. 6.5b. Visualizations of results were a wish from Skogkurs and the stakeholders. User testing indicated that the results were mainly clear and easy to understand, although no A/B testing was done on this.

Especially the external stakeholders wanted to see how each input affected the calculation of cost and productivity. Due to little specificity in how they wanted to see this information, the group decided to implement a graph view, as can be seen in fig. 6.5c. The user may select to show either productivity or cost, and must select a specific cost driver (input) to analyze. The X axis of the graph shows the user input value in the middle, and values above and below. The Y axis shows productivity or cost. The two graphs represent the two machines used.

As the user selects an input to analyze, the cost (or productivity) is calculated for all values in a set interval. The user may trace the graph and see how the results are affected as the input value changes. When demonstrating this functionality to the external stakeholders, its reception was mixed. It was noted that the visualization might be too complicated to



(a) Mobile version **(b)** Desktop version

Figure 6.6: Final design

understand. However, user tests indicated that the graph was a valuable addition, and most users understood its purpose and content.

Final version

After testing the MVP, some problems were detected. With regard to navigation, the test uncovered that it was not immediately apparent that the line of navigation buttons was scrollable. One user also pointed out that a scroll bar potentially could be a problem in the forest in wet weather conditions.

Another issue from the user testing was that the circular graph used was somewhat confusing. The reason for this is that while the graph shows both machines' productivity, only the value for the least productive was shown numerically.

There was also a desire among the stakeholders that the result should be visible at all times. Navigating to another page to see what the input value did to the result was considered tedious, especially when experimenting with different input values. A larger redesign of the application was done to solve these issues. As depicted in fig. 6.6a, a minimized result is always visible at the bottom of the screen. The user may click the result bar to view the full result page. Navigation between input pages is back at using tabs, as dropped in an early version. In case many buttons are needed, the tabs are simply divided into more rows, which solves the problem of horizontal space outlined in the prototype section. The result page is mainly kept as in the MVP, but the circular graph is replaced by a bar graph,

showing values for both machines numerically and visually.

From largely focusing on the mobile version, the application was adapted for larger screens. As can be seen in fig. 6.6b, the layout allows all information to be presented in a single page. To complete the application, a front page was added, where users can select which calculator to use.

A toolbar is also present at all times. The tool bar lets the user navigate back to the front page. It also contains buttons to store or share calculations, as well as a menu linking to additional app information such as API documentation.

No user testing has been done with the later iterations, but periodic feedback from Skogkurs and the external stakeholders indicates that the design has evolved in a positive direction.

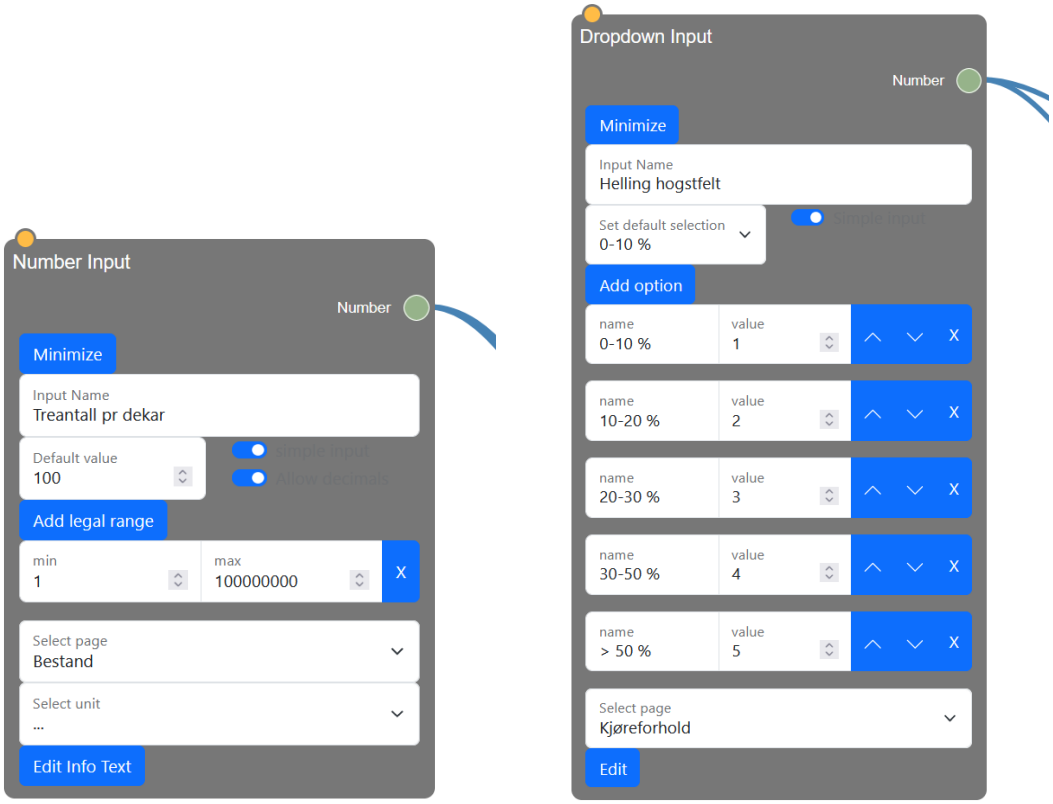
6.2.2 Responsive UI

A key requirement from Skogkurs was that the application should be available on multiple types of devices. Whether the user wishes to use their phone, tablet or desktop computer, the user experience should be similar. Designing an app that functions equally well across devices is a challenge. The individual components have to be designed to fit and provide the needed information in a small screen. Our approach was to focus on the mobile representation from the beginning of the design process. This is a sensible place to start because challenges of fitting all necessary components into a small screen is more difficult than on larger screens.

Using a grid based layout system, rows and columns of content are added or rearranged at set screen size breakpoints. For tablet and desktop, the layout is quite similar. The mobile version differs in more ways. The most apparent difference is that there is only one column in the mobile version, with different UI components stacked on top of each other. If the page contains much content, the user will have to scroll to see all of it. The wish of seeing calculated results in real time when inputting values dictated a compact result presentation to be implemented specifically for the mobile version. A small result bar was added at the bottom of the screen, with the option to maximize the bar to see the full results.

6.3 Editor

Compared to the web app, the process for iterating on a graphical design for the editor has been limited. This is owed to the conscious decision to begin development on an interface for the web client in the earliest stages of the project period, to facilitate multiple rounds of feedback from both the project owner and stakeholders, in addition to performing a round of formal user testing.



(a) A number input node **(b)** A dropdown input node

Figure 6.7: Input nodes

Owing in part to the shorter development time allocated to the editor, the primary requirements for the editor was functionality. Consequently, accessibility and usability would not be prioritized.

That being said, we have made some considerations for usability, namely the node-based display, the module system for containing distinct mathematical functions, and the side panel for managing meta-data.

6.3.1 Nodes

To avoid cluttering the interface, each node displays and allows manipulation of information that affects that node exclusively. I.e., each input node has a name field, rather than necessitating a separate "name"-node type that would give other nodes their name (fig. 6.7). This means that any given node may potentially require a lot of screen space, to display all the unique details of that node. To alleviate, nodes can be minimized in the editor display (fig. 6.8).

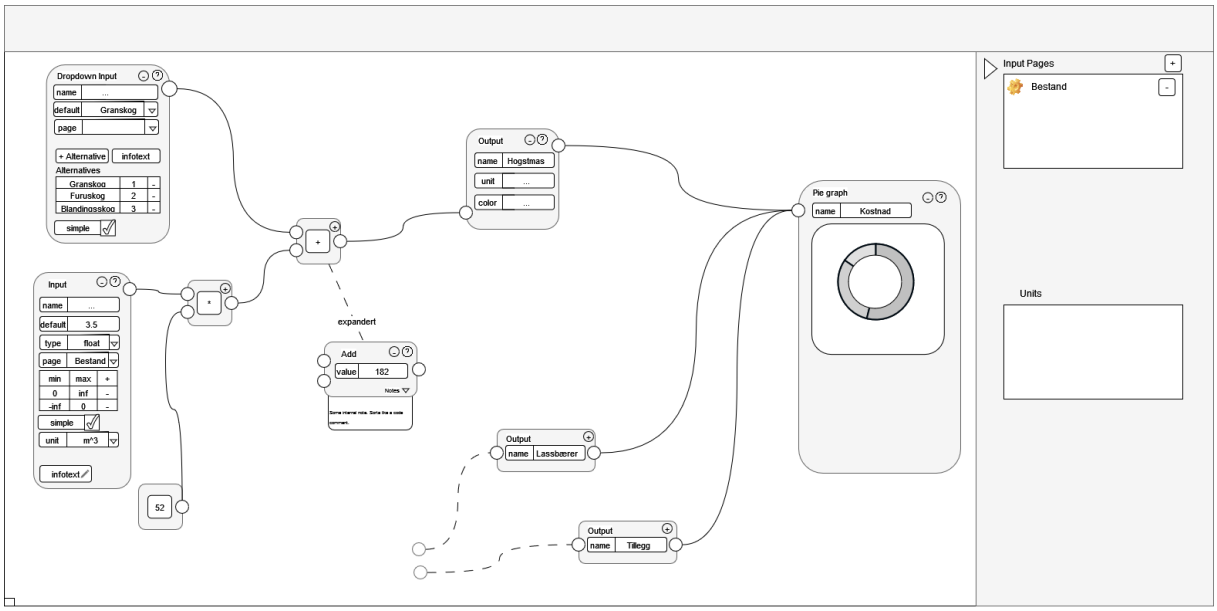


Figure 6.8: Second wireframe of editor with buttons for minimizing nodes.

6.3.2 Modules

Grouping nodes associated with a distinct mathematical function, like calculating the effects upon logging costs of terrain conditions, may aid in understanding a given graph. Displaying these nodes directly in the main view, however, might quickly clutter the display, particularly once the number of nodes and distinct calculation steps start increasing (fig. 6.9). To mitigate this, we have implemented functionality for containing the nodes associated with a particular mathematical functional within a separate and distinct module (fig. 6.10). In this way, those nodes are displayed and manipulated in a separate view from the main formula, with the benefits of de-cluttering the primary view of the editor. Apart from hopefully improving the navigability of the editor, this allows users to reuse modules, by exporting and importing them in the form of JSON-files. Should a module be usable in multiple formulas, the user is then saved time in re-implementing identical sub-calculations.

6.3.3 Side-panel

Keeping track of node meta data, in a strictly node-based editor, would mean connecting meta data nodes to every single node that required it. For large graphs, this might generate a great deal of visual noise.

To avoid this, we created a side panel in the editor view, to encapsulate the management of data like the unit of a nodes value, which page an input node belongs to, or how the node displays should be organized in the web app (figs. 6.12, 6.11b).

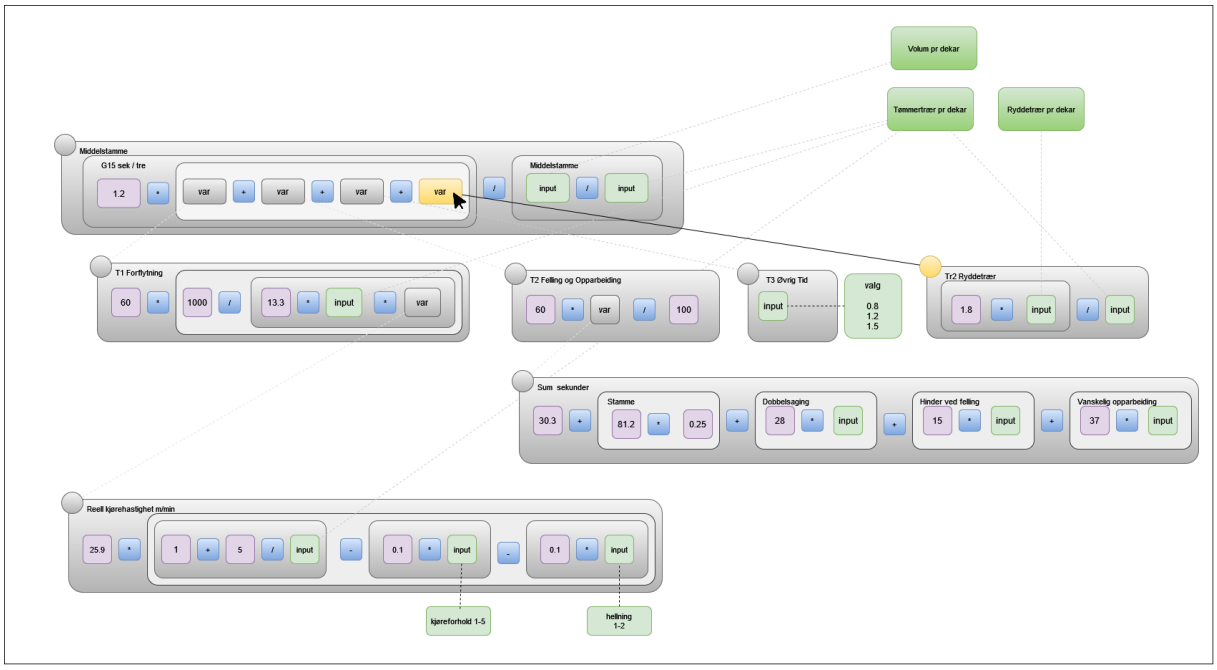


Figure 6.9: An early concept for what we thought the editor may look like, with nested nodes.

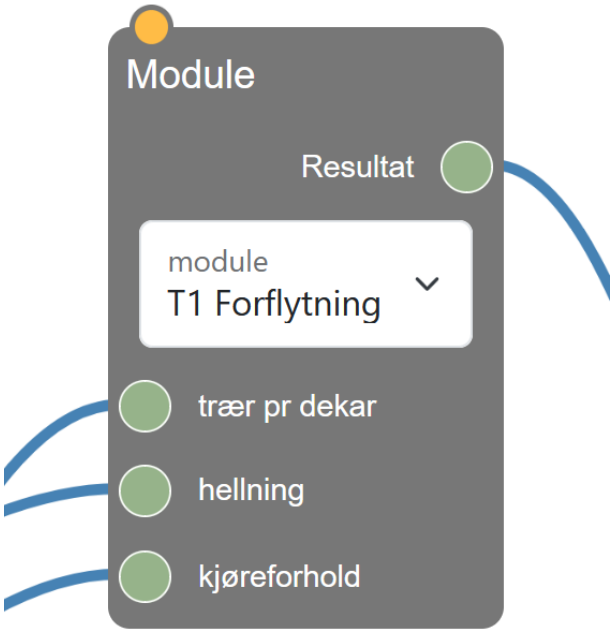
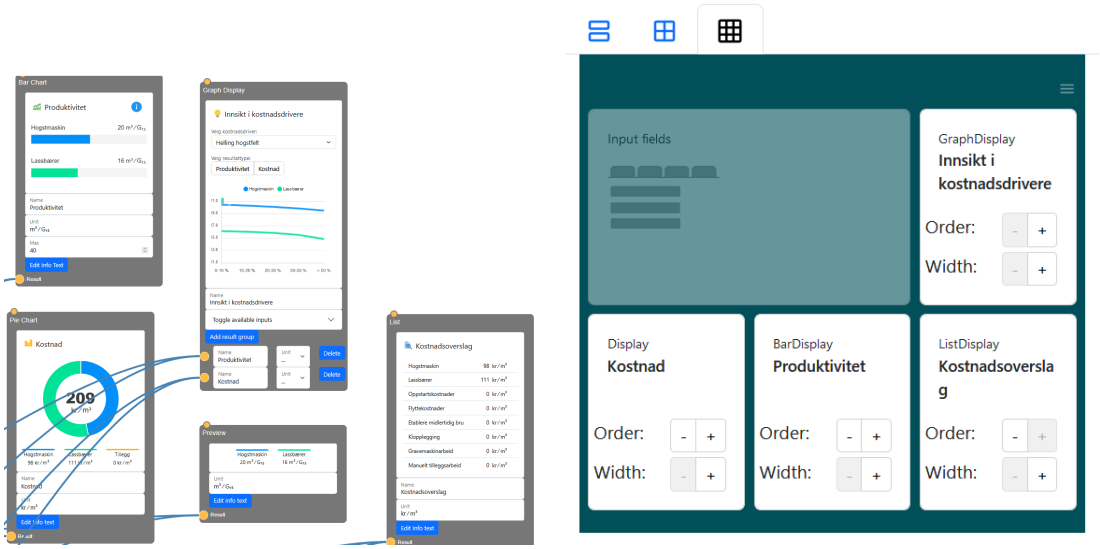


Figure 6.10: A module node with inputs and outputs connected



(a) A grouping of display nodes **(b)** Preview of ordering and span

Figure 6.11: Display nodes

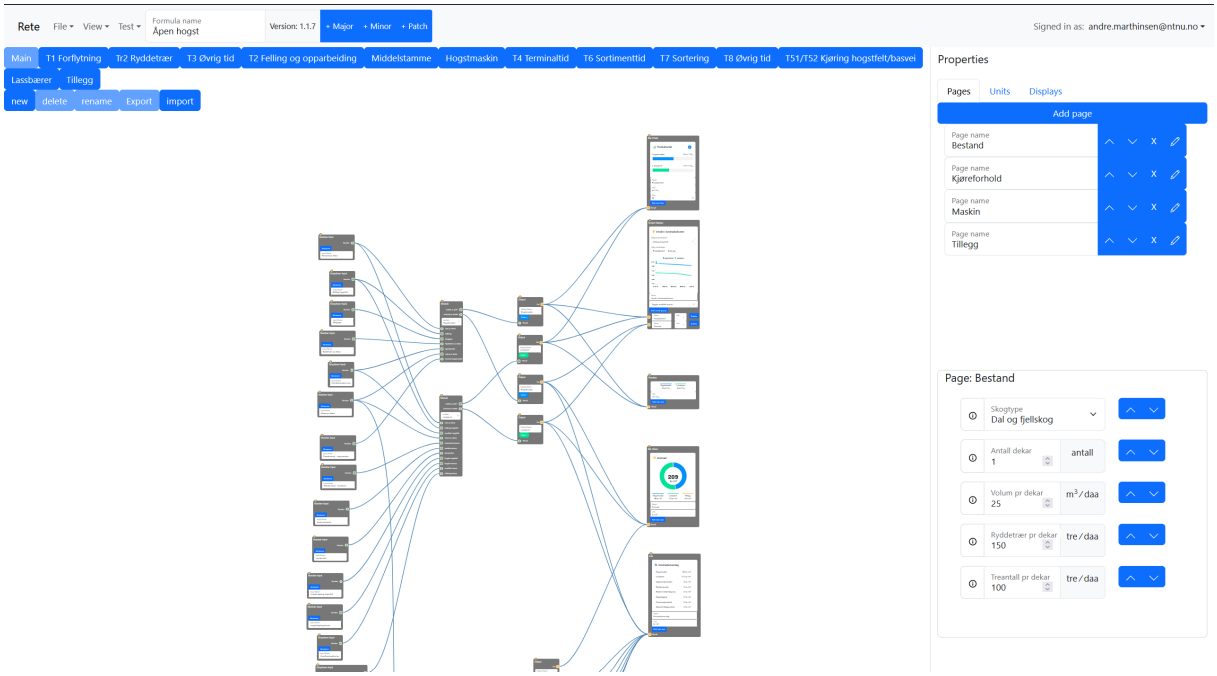


Figure 6.12: Main graph of the current implementation of Skogkurs' calculation tool.

6.4 UX design reflections

Designing good user interfaces can be a challenge. The original requirements may be open to interpretation. Often the product owner is unsure of what they really want. The end user often have opinions differing from the product owner of what makes a good user experience. Taking on the role of UX designers, we must decide how to design the application. The decisions should be supported by best practices in combination with input from PO, users and other stakeholders.

In our case, the starting point was the already existing excel sheet. This was used as a basis for our further design iterations. Transition from a desktop friendly spreadsheet into a mobile friendly layout implies that several challenges must be overcome. One example is the organization of user input fields. The spreadsheet had all inputs on the same page, which for a user may seem like information overload, making the product difficult to use. By grouping inputs according to the real world application, the user is presented with a few inputs at a time to fill in, letting the user focus on what is important at that time.

6.4.1 Feedback

During the process, we got feedback on the design through multiple channels. Much of the feedback was given to us in informal settings at the Skogkurs office. This type of feedback is valuable because it allows us to quickly correct problems that the product owner or potential user sees during the development process. A problem with such feedback is that it often comes from one single source, and that makes it challenging to decide its importance. Organized feedback in the form of usability testing, where we investigated specific details of the app was a valuable addition. Although a qualitative user test is no exact science, we were able to see tendencies, for example multiple users pointing out the same problems with the application. Getting similar feedback from more sources is helpful when deciding on what to implement. Combining our own expertise with the stakeholders' and users' point of view has helped us take informed design choices that the application has benefited from. Talking to a UX professional during the early stages of the design process also helped us.

As programming students, we may tend to focus more on the technical aspects of creating an app, rather than the design and navigation. The design we had implemented at the time made sense to us as developers. Knowing the process and reasoning behind choices made, we may understand the navigation and symbols, but we should not take for granted that people without that background information also understand. This is a key challenge in design. Putting your own bias aside and seeing the application from a user's point of view is a difficult task, but we believe it is required to create a concise and discoverable UI allowing a good user experience.

6.4.2 Navigation

Navigation in an app is highly important for the user experience. If the navigation is difficult, or feels off in any way, the user may get frustrated. Trying out different navigation modes has been interesting. For the developer, one form of navigation may seem like a good choice, while the actual users may find it confusing or difficult. Demonstrating the application for Skogkurs and other stakeholders has been valuable because we have got a significant amount of feedback that way. The user testing also contributed to insights that we would not have gotten elsewhere. A good user interface should be intuitive to the user. The user should not need any training to navigate the application. With the help of our own opinions and experiences, and, more notably, feedback through user testing and less formal channels, the app design has iteratively evolved from mostly making sense to the developer team, into being intuitive for most users.

Chapter 7

Implementation Details

This chapter details technology alternatives and choices made, code implementation and challenges faced and how these challenges were dealt with in implementing the design of the project.

7.1 Overview

This section aim to introduce the technology stacks employed in the project. Table 7.1 lists the various dependencies used by the front-end clients. While many of these dependencies are shared between the web app and the web editor, some are unique to one or the other; notably, Workbox is used only in the web app, while Rete is specific to the web editor.

Table 7.1: Core Frontend Dependencies

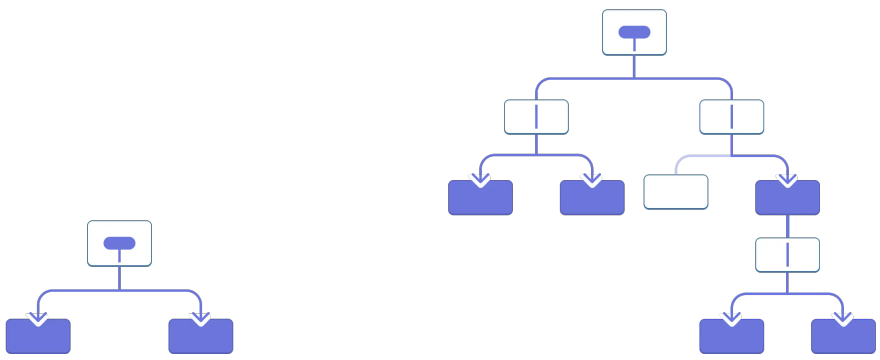
Dependency	**Version**	**Description**
Vite[1]	5.2.10	Frontend bundler and development tool
React	18.2.0	Declarative component-based UI library
ReactDOM	18.2.0	React DOM integration library
Redux	5.0.1	State management library
Bootstrap	5.3.2	Responsive design framework
Rete	2.0.0	Node-based visual editor framework
Apex Charts	3.46.0	Charting and visualization library
Workbox	7.0.0	Service worker toolkit

The team already has prior experience with React and Bootstrap, both of which continue to be among the most utilized libraries/frameworks within web development. According to the annual *Stack Overflow Survey*, React is the most used JavaScript library that developers reported using extensively over the past year [16]. Similarly, Bootstrap holds a dominant position as the most used CSS framework, capturing a market share of 77 % among websites where CSS frameworks are identified [17]. The team briefly explored other libraries and frameworks, but ultimately concluded that there was no appealing reason to switch, given the project's use cases, the advantages of leveraging prior knowledge, and the libraries continuing industry relevance. The codebase is primarily written in Typescript, ensuring compile-time type-safety. More details on the build system and code organization is discussed in section 9.1

For further discussions and details on the comparisons of different frameworks and technologies, refer to our research document in Appendix J.

State Management

A common challenge with unidirectional data flows in declarative UI frameworks like React is the need to lift shared properties and callbacks to a common ancestor in the component hierarchy (see Figure 7.1a). Since UI components are encouraged to be small and reusable, this often results in *prop drilling* (abbreviated from *property drilling*). Prop drilling occurs when excessive data is passed through multiple nested components that do not directly need it (see Figure 7.1b). An alternative to prop drilling is to use context, a React feature that injects props into the component tree, eliminating the need to pass them at each level. However, from a performance and maintenance perspective, context isn't ideal for managing individual props. Instead, using a centralized state management library helps maintain a unidirectional flow of data through context more efficiently.



(a) Lifting props to common ancestor **(b)** Prop drilling

Figure 7.1: The problem with passing props. Image source: [18]

[1]Before version 2.0-alpha2, releases were bundled by create-react-app, which is now deprecated. See Section 10.2.1 for more details

Redux is a popular state management library commonly used with React. However, its complexity and extensive boilerplate led the maintainers to develop Redux Toolkit, which simplifies common Redux patterns and is now the recommended approach [19]. This is the primary state management library in our web applications, but the web editor is much more involved, and go beyond the scope of centralized state management, see section 7.3.2. One of the most state-intensive tasks in our web apps is handling API operations. The user interface must respond to various loading stages and manage data transformations and errors. Fortunately, Redux Toolkit includes a query pattern that combines Redux's capabilities with network request handling, greatly simplifying API calls and making the UI code easier to maintain.

API

As stated in the cloud provider requirements (section 3.2.6), the system must facilitate code sharing between the frontend and backend. Specifically, the API must be able to import JavaScript modules (pre-compiled from TypeScript), which imposes significant constraints on how the API can be implemented. As such, the API is also built with Typescript, uniforming the project to use the same language across the entire project.

While Node.js, the modern JavaScript runtime system, allows developer to write web servers directly through its http module, there are popular frameworks that simplify the process, with more robust routing mechanisms. Express is the most popular backend framework for Node.js [20], built for this exact purpose, with very good support for third party middleware[2], such as rate limiting and JSON parsing. Firebase SDK is used to connect to the rest of the backend services as a whole and will be covered later in section 7.5. Finally, there's the bundler *Rollup*, which is also integrated into Vite in the web applications, not only aggregate code, but also transcribes ES6 export statements (where file-extensions are missing) that is not caught by the Typescript compiler, thus reducing the likelihood of run-time exceptions.

7.2 Web App

7.2.1 Two major releases

Unlike many software projects where each release builds directly on the previous one, this isn't entirely the case for this project. Due to uncertainty about the technical state that could be achieved by the project's deadline, we've developed both releases mostly in parallel. This

[2]"Middlewares are codes that execute before an HTTP request reaches the route handler or before a client receives a response, giving the framework the ability to run a typical script before or after a client's request." [20]

Table 7.2: Core API Dependencies

Dependency	Version	Description
Express	4.17.21	Web server framework
Firebase Admin SDK	12.0.0	Connects to database and IaaS
Rollup	4.16.0	Bundler

strategy ensures the possibility to fall back to a fully functional release 1.x in case we run out of time to complete release 2.x. Both versions fundamentally share the same design, functionality, and thus look and feel identical from an end-user perspective. However, release 2.x is designed with maintainability and flexibility in mind.

Release 1.x, referred to as 'Static Version' Given the extensive time required to construct the tree structure representing the entire computation, all input attributes, mathematical functions, and presentation layouts are specified directly in the source code and integrated into the single-page application for the first major release. These releases were initially developed out of the necessity to gather early feedback from stakeholders, primarily on the graphical design; but also to test the mathematical formulas derived from Excel.

Embedding all the business logic directly into the source code requires recompilation and redeployment for even minor modifications, which is not ideal if future adjustments, such as when changing default values are desired by the customer. Nonetheless, this approach enabled rapid prototyping, resulting in useful feedback nearly every week. This continual input was instrumental in refining the user interface, which ultimately defined the requirements for each tree node.

Release 2.x, referred to as 'Dynamic Version' The latest major release use tree structures as blueprints to configure the layout, how to present the results, and the parameters for each input field displayed across different pages. This is accomplished through API requests, originating from the web app, to the backend in order to fetch each calculator's configuration. When used in conjunction with the web editor, this system greatly simplifies the process for Skogkurs to upkeep their existing calculators and develop new ones. Content updates and design improvements were continuously backported to release 1.x

7.2.2 Service Worker

As stated in section 3.2, the web app needs offline support to function in network-less environments like forests. This is achieved by using a service worker to prefetch static files

and provide cached responses to HTTP calls when the network is down. For a deeper understanding, refer to the section on service workers, 2.3.1.

Managing service worker lifecycles, cache management, versioning, and maintaining browser compatibility are specialized skills that require deep knowledge of low-level APIs. Consequently, numerous tools and libraries have been developed to simplify these complexities. See table 7.3 for a comparison between the most popular tools.

Table 7.3: Comparison of Service Worker Tools

Feature/Tool	**Workbox**	**UpUp**	**PWA Builder**
Primary Focus	Service Worker	Offline support	PWA
Ease of Use	Harder, high-level abstractions	Very easy, minimal setup	Easy, graphical setup
HTTP cache	Yes, through strategies	No	Yes, through strategies
Customization	Moderate	Limited	Limited
Maintainer	Chrome Aurora[3], initiated by Google[21]	Community	Community, initiated by Microsoft
Community	Large	Small	Good

SW-Toolbox is another popular tool, but left out since it's been deprecated by Google in favor of *Workbox* [23]. Given the project's need for HTTP response caching, *UpUp* is not a viable option. While *PWA Builder* presents an attractive alternative, it requires more manual intervention whenever project file structure change, as the service worker must be manually regenerated through their website.

Workbox Emerging as the best option, Workbox not only offers granular control and seamless integration into the build process but also boasts widespread adoption. According to HTTP Archive, 54% of all PWAs on the web used workbox in 2022, marking an increase of 24 percentage points from the previous year[24]. Furthermore, Google's expertise in browser technologies and web standards lends significant credibility to the library.

Prefetching files Given the application's small size, particularly in release 2.x, Workbox has been configured to prefetch the entire website, encompassing images, stylesheets, and

[3]Chrome Aurora is a collaboration between Chrome and open-source web frameworks & tools[22]

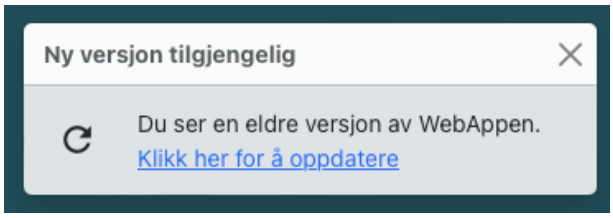


Figure 7.2: Refresh Dialogue

additional JavaScript code. Notably, the service worker file itself is excluded from prefetching to facilitate the registration of a new service worker in case of manifest updates. For further insights, refer to Section 2.3.1.

More importantly, is how service worker updates are handled. Activating a new service worker immediately terminates the previous one, risking data inconsistencies. Furthermore, changes to prefetched files won't apply without a refresh. While forcing a page refresh upon service worker updates is a solution, it can lead to unexpected user experiences, especially if activation takes several seconds. A more user-friendly solution, which is what Google recommends[25], is to notify the user that a newer version is available, whilst providing a button to activate and refresh. Figure 7.2 depicts how we implemented a toast notification during development to inform users that a new release is ready.

HTTP Caching As described in Section 2.3.1, Workbox utilizes caching strategies upon intercepting HTTP requests. Among the five available strategies, only two are relevant to this project:

1. **Network-first:** Prioritizes fetching data from the database (through API) but can fallback to the cache if needed.
2. **Stale while revalidate:** Serves content directly from the cache while simultaneously updating it for future requests.

Both of these approaches come with their own set of considerations. The *Network-first* strategy prioritizes displaying the most recent version of a calculator, albeit at the expense of responsiveness, as it necessitates the SPA to connect to the API, resulting in potential delays for the end-user. Conversely, *Stale while revalidate* presents content from the cache at any given moment, requiring an additional refresh to reflect updates to the calculators.

As a precautionary measure, we've initially chose the *Network-first* strategy. However, as updates became less frequent towards the project's conclusion, we transitioning to *Stale while revalidate*. Additionally, a limit on how long the stale data can remain in cache has been set to one year, so it cannot be used offline indefinitely. While ensuring offline functionality, this approach also serve as a control mechanism, enforcing users to occasionally check for updates.

Since the web editor is hosted on the same subdomain and port, the service worker will

Figure 7.3: PWA installed on Android 14

also intercept the HTTP requests. This can lead to undesired outcomes because cached data may not reflect the most recent updates, potentially causing data loss. To address this issue, a *Network-Only* strategy with higher precedence is implemented whenever the HTTP header *X-No-Cache* is present. This header is globally defined in the web editor API client.

7.2.3 PWA

While the browser provides direct offline support, certain users may like to save the web app onto their device. This is facilitated by a manifest file (not to be confused with the prefetch-manifest), which outlines the app's details, including links to app icons in various resolutions. Initially, our team began designing the app icon but later delegated this task to Skogkurs' communication advisor and UX designer due to branding concerns. The final app icon is illustrated in Figure 7.3. Some browsers automatically detect the manifest file upon visit, and subsequently ask the users if they like to install the PWA, while others, such as Safari, do not offer this functionality (without publishing to the App Store), and require the user to save it manually.

7.2.4 Calculator storage

The web app allows users to store calculations for later retrieval using the browser's local storage. This means the stored calculations are only accessible in the specific browser and device used for storage. The duration for which calculations remain in local storage may vary between browsers, and deleting browser data will almost certainly remove the stored calculations. While allowing users to store calculations on their file system would provide greater persistence and portability, it was deemed too complex to implement within the project period

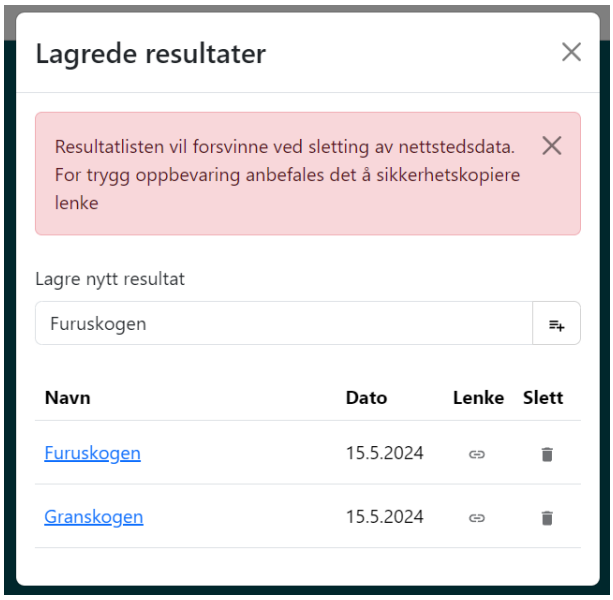


Figure 7.4: Storage menu

Calculations are saved as URLs containing metadata such as the calculator version and all input values. The calculation depends on the existence of the specific calculator version in the database. If a calculator is deleted or undergoes breaking changes (e.g., added inputs), the stored calculation will no longer function.

7.3 Editor

The most important decision early on in the project with respect to the editor was selection of a visual programming framework. Some primary considerations were ease of integration with React, level of documentation, feature richness and whether a potential library was under maintenance still or not.

While the options were many[4], when looking up visual programming, Rete.js frequently. With a good documentation and a large selection of examples demonstrating how one can achieve various functionality and support for React, rete seemed like a good choice.

In retrospect, React Flow could have been a better choice for its smoother integration with React and Redux, but determining the level of friction between frameworks ahead of time can be quite challenging, even more so if one doesn't fully understand React when starting out.

[4]A wide selection can be found here https://github.com/xyflow/awesome-node-based-uis

7.3.1 Rete.js

Rete is set up to be highly modular. NodeEditor, a container class for nodes and connections with methods for manipulating, adding or removing these primitives, serves as the core of rete.js. The NodeEditor lets you define a graph, but to do anything beyond this one has to add one of the many plugins available or extend it with logic manually. DataflowEngine, AreaPlugin and ReactPlugin are of particular interest for this project, and will be discussed briefly in the following sections. Other plugins have also been used, but provide minor functionality compared to those mentioned.

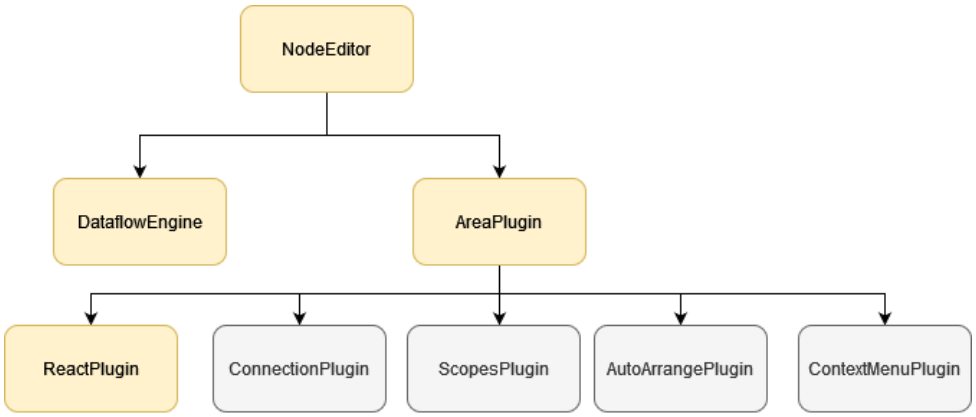


Figure 7.5: Signal direction and ownership hierarchy

AreaPlugin and ReactPlugin

AreaPlugin adds the notion of a 2D surface. Where the NodeEditor alone has no visualization, AreaPlugin combined with a plugin for React, Vue, Svelte or Angular adds a predefined visual to the editor, allowing nodes to actually be displayed and moved around. For this project React was the framework of choice for rendering, and so ReactPlugin was added for visualization. Some custom components were written, but for the most part the defaults were utilized due to time constraints and having to prioritize making the core logic function.

Dataflow

The Dataflow plugin is responsible for piping data from node to node and invoking .data() throughout the graph in a recursive manner.

Pipes

Rete structures itself by allowing the build-up of a tree where each plugin can be considered a node, the editor itself acting as the root. This is achieved by calling .use() on a parent with the child as the argument. Each plugin, or scope, can emit payloads in response to events such as adding or removing connections, which are sent down the tree towards the leaf

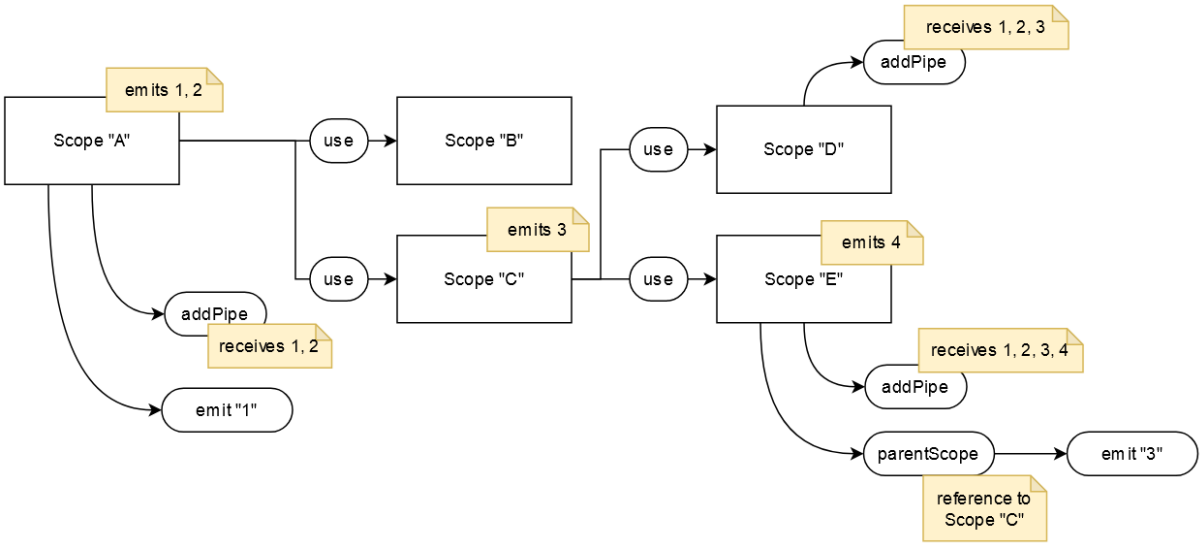


Figure 7.6: Diagram detailing plugin architecture sourced from rete.js documentation [26].

nodes, as shown in figure 7.6. Rete allows for the registration of *pipes*, essentially middleware functions that receive a payload as their input. These functions can perform anything from logging, enhancement of the payload, or preventing the payload from passing down to the children of a plugin by returning null.

Figure 7.5 shows which plugins are in use in the GraphEditor and their hierarchy. Pipes are set to verify the validity of a connection to prevent connection of incompatible sockets, as well as signaling subscribers of the GraphEditor. Additionally, .data() is invoked on all nodes on connection or node creation or deletion.

7.3.2 Architecture and State Management

As the project progressed, it became apparent that some of the UI for manipulation of the data structure would have to be moved out of the node graph for the sake of usability. As an example, we needed a way to encode a generic information about *pages* that input fields would belong to. Initially we attempted creating a simple label node with string information that could be used as a page, connecting to an input on and input field node. The problem arose when we needed to let the user set the order in which they would appear within a page, as there was no obvious practical way of encoding order through nodes and edges alone which wouldn't be cumbersome to keep track of.

A challenge was that this state had to be stored somewhere, and we were faced with two realistic options and a third impractical one.

1. Keeping all node and graph related state in the Node Editor class, providing methods for changing pages and ordering and then injecting the contextual information into the Nodes from the Node Editor.

2. Offload some state management to one or more Redux store slices.
3. Put all state management in the Redux store, using the Node Editor only for manipulating the graph structure.

Option 1. would be impractical for a few reasons. First off, React is designed to work with its own state system, declaring stateful variables through useState(). Working with state encapsulated in classes involves manual updating of react state variables through the use of getters on the class whenever calling methods that may update the state of the class. If the class state changes in one React component, another component may rely on the same object without being able to easily observe that change. Additionally, the graph state may change through canvas interaction, which the external React context has no knowledge of. While doable, it is prone to bugs due to relying on constant vigilance on part of the developer. Secondly, this would require we expanded an already overwhelming interface with yet more methods.

Option 2. would not be without its problems. While more practical with respect to writing React components as Redux integrates well with React, we would now have two sources of state for graph. Having two sources of state is problematic for a couple of reasons. Both a Node and the Redux store holds information about that particular node. That state now has to be held in sync to ensure the user sees what is ultimately exported to the database.

Option 3. would be likely be highly impractical as it would require a lot of re-writing with minimal benefit compared to resources spent, while also tying processing of node data to React rendering which could have negative side-effects on performance. To accommodate some of the underlying algorithms, the full state could likely never be fully put into the store, as some data would still have to be mirrored in the node object itself.

Neither of the options were great. We opted to go for option 2, while keeping state manipulation and management outside of the graph to an absolute minimum, being limited to pages with input ordering, unit declaration and display ordering. The following section will detail how the store and GraphEditor co-operate.

Architecture

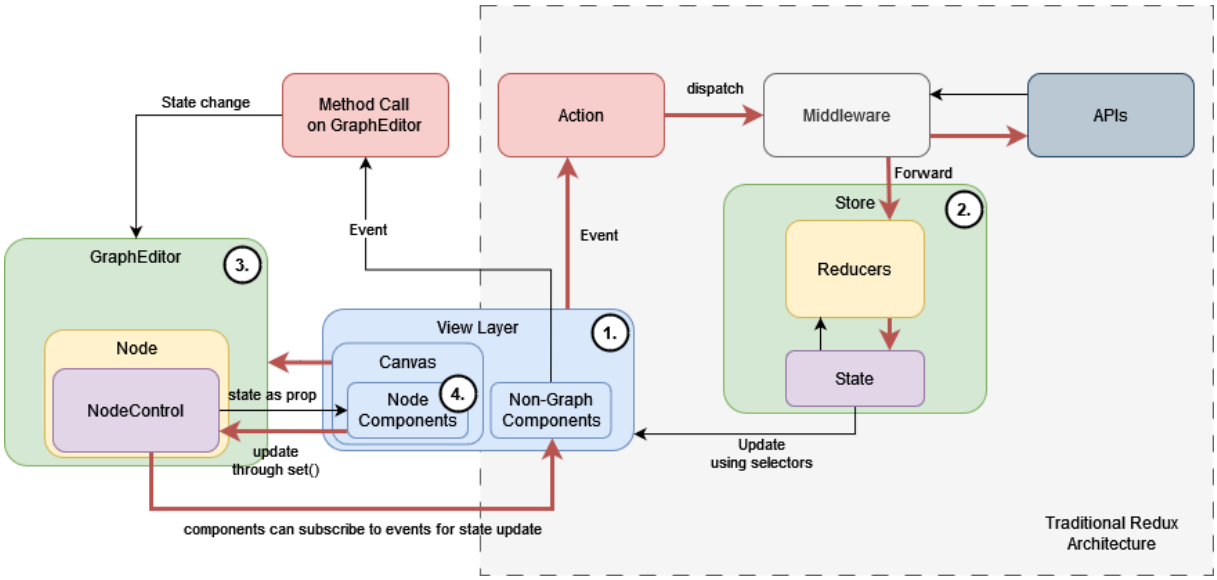


Figure 7.7: Editor Redux interaction. GraphEditor is simplified for clarity.

Figure 7.7 shows an overview of the editor architecture. The arrows highlighted in red indicate flow of data from the user to the Redux store, and ultimately the database. The section outlined in a stippled box represents the traditional Redux architecture, whereas the elements outside of it demonstrate how the GraphEditor integrates with Redux.

1. The **view layer** consists of two parts. The canvas is an interactive surface where the user can create and delete nodes, also serving as the container of the nodes. The remaining components make up the surrounding menus, tools and navigation.
2. The **Redux store** provides selectors for accessing predefined data structures, which can be updated through dispatching actions from components.
3. The **GraphEditor** class acts as the container for all graph related logic. It contains the nodes and the connections between them. The nodes in turn contain NodeControls that serve as the singular prop for the respective React component of a node, providing methods for retrieving state for rendering and setting new values.
4. Both the **canvas and node components** act as interfaces for a user to manipulate the graph structure, and as a consequence *they make up the main source of data*. The canvas lets the user create and delete nodes and connections, as well as repositioning nodes and manipulating the viewport. The nodes themselves allow the user to manipulate state such as numeric values and input names.

As the user manipulates the graph, two data structures are continuously updated and kept in memory. One is the serialization of the rete.js graph, while the other is the data structure meant for use in the web *app*. The app data structure is kept in the Redux store and used for visualization in display nodes, allowing users to verify that the data structure

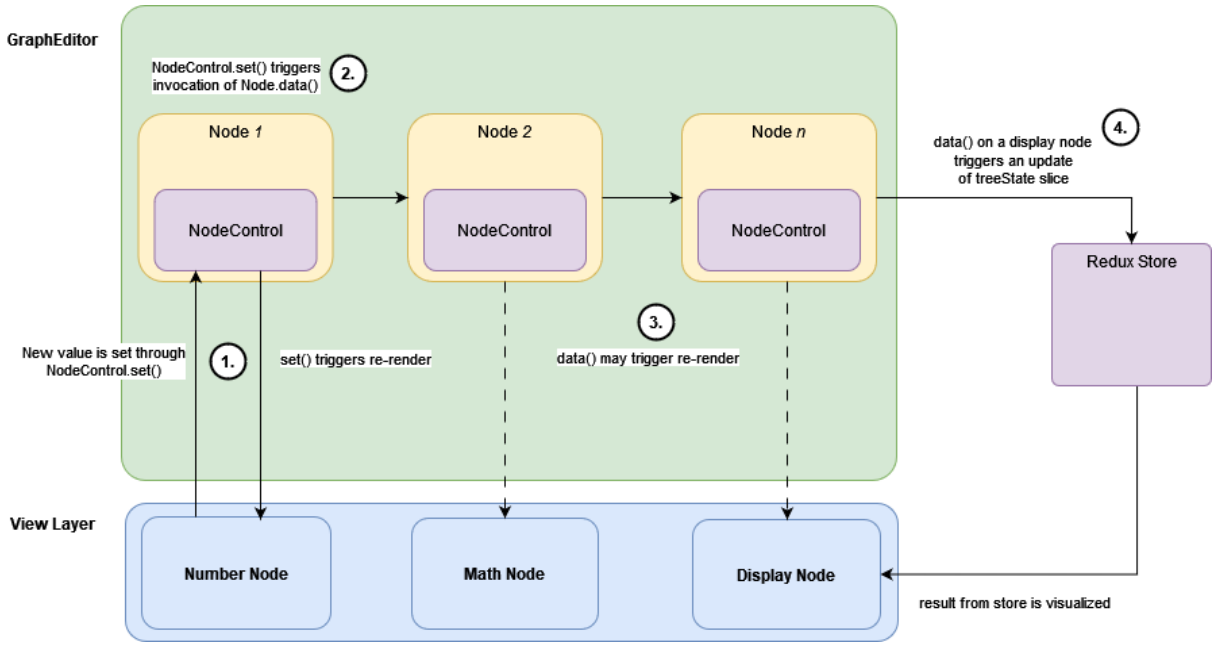


Figure 7.8: Event-flow on user interacting with a number node

is working as intended. A benefit of this is that bugs that would show in the app will show here first as they use the same react components and logic for rendering.

Following the red arrow we can see the data flow pass through the *non-graph components*, meaning menu bars and tool bars, and other components wrapping the canvas, where the user can add additional information like file name and version. This outer context has access to the GraphEditor interface as it is instantiated at the root of App.tsx, while nodes only have access to a set of private methods on the editor that are provided through injection. The store on the other hand is available in both contexts, though one must take care with how one uses the store inside nodes to prevent recursive state changes or performance issues.

Event flow

Figure 7.8 illustrates a typical sequence of events when an user interacts with any node where data can be set. Adding or deleting nodes or connections functions much the same, with the exception that it is the GraphEditor instance that triggers .data() invocations rather than a NodeControl instance through the use of pipes as described in section 7.3.1.

1. The user changes a value on a node, in this case the numeric value on a number node. .set() may induce any number of side effects, but in this case it triggers a re-rendering of the node.
2. In addition to the re-rendering, .set() also triggers recalculation of values in the graph by calling a function provided by the GraphEditor through injection on construction of the node.

3. Some nodes may have to re-render when their .data() method is called. This is generally only the case for math nodes as they display the numeric value of their operation applied to its inputs, which is recomputed when .data() is called.
4. When .data() is called on a Display node it results in the GraphEditor signaling a state change. This too is achieved through calling a function injected during construction of the node. The signaling causes logic in App.tsx to update the slice containing the treeState, which in turn causes a re-render of the display node as it depends on treeState.

7.3.3 Classes and Algorithms

This section will detail the major classes and algorithms that power the editor web application. While every class won't be detailed here, a class diagram can be found in appendix G providing an overview of the major classes and their relationships.

GraphEditor

The GraphEditor could nearly be said to *be* the editor app, as it contains the majority of application logic, providing methods to allow communicating with a view layer. Acting as a *facade* for its *component* classes, it directs the cooperation between them, allowing for a class that contains a lot of logic to be relatively maintainable by breaking it up into portions. Breaking the code up in this way let us re-use the code while also adhering to the SOLID principle of *single responsibility*[27], which states that a class should have a single reason to change, or phrased differently, "Gather together the things that change for the same reasons. Separate those things that change for different reasons."[28]. GraphEditor only has to change if the user needs a new way of interacting with the graph, while classes that make up part of GraphEditor, such as GraphSerializer, each have their distinct reason to change.

To allow extension of GraphEditor's graphical interface, such as adding toolbars for creating, deleting and renaming modules, we opted to implement the *observer pattern* [29] by providing methods that would allow a React component to subscribe to the GraphEditor as if it was a store, signaling a state change and providing the component with a method for retrieving a snapshot. This let us write components that would change the state of the GraphEditor through method calls without having to manually update state in the component through setState() calls.

Node

Figure 7.9 shows the basic layout of a rete node with inputs, outputs and controls. Inputs and outputs essentially boil down to a label, an id and a Socket instance, where the Socket is the actual connective point. Sub-typing the Socket class let us define legal connections.

Every node in the editor sub-classes one of two classes; BaseNode or ParseableBaseNode[5]. BaseNode is an abstract class extending the rete Node class and implementing the DataflowNode interface, adding methods for serializing and deserializing the node, and a data() method to satisfy the DataflowNode interface for use with the DataflowEngine plugin. ParseableBaseNode extends BaseNode with an additional method; toParseNode(). This method returns an object representation of that node in a fashion compatible with the web app data structure. In short, nodes that have a 1:1 equivalent in the web app data structure inherits this class, while nodes that have to be processed further, such as module nodes, do not.

Node GraphEditor Interaction

As a node may have the need to cause side effects in response to state change and events, up to three functions are injected during node construction.

1. updateRendering(id: string) => void
2. signalChange() => void
3. updateDataflow() => void

updateRendering simply tells whatever rendering plugin is being used to re-render a node of that id. This is invoked when the state of anything that is visually represented on the node has changed.

signalChange is used to indicate that the translated[6] graph data structure needs to be updated for visualization in display nodes.

Finally, updateDataflow causes the values of all nodes to be recalculated. An issue that might be immediately apparent to some is the fact that nodes are allowed to invoke these calls directly without deferring the decision of whether the code execution should go through or not to the GraphEditor. As Node objects do not have awareness of the full graph context, they cannot tell if a particular is redundant, leading to performance issues when several nodes trigger an update of the parseTree state.

This only became obvious near the end of the project due to the editor reaching the appropriate maturity for implementing larger formulas at that point, and for that reason we did not have time to prioritize fixing it when we also had to prioritize the thesis itself. For the sake of performance, the GraphEditor would need a more mature system to handle communication with its nodes. The likely path would be to only allow them to signal the desire to cause a recalculation, re-render or change to the parseTree state so that the GraphEditor itself can determine whether an action is actually necessary, and if so, do it in an optimized order.

[5]The parse portion of the name derives from the parseTree library in the common module, a misnomer that stuck throughout the project.

[6]NodeGraph data structure converted to a parseTree

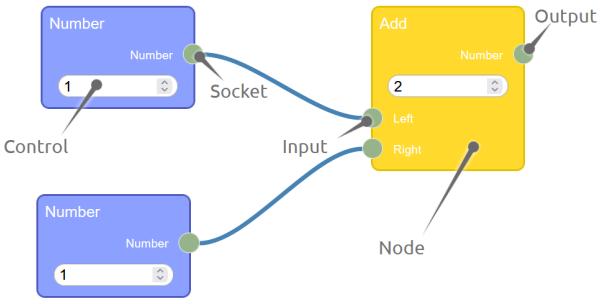


Figure 7.9: The basic structure of a rete node, sourced from rete.js documentation [26]

This could expressed as intents and payloads, reducing the code surface to a single method on the editor exposed to the nodes, rather than distinct methods for each. This could also help streamline node construction as they'd all just have the one method injected on construction.

NodeControl

In rete, a control is an object that is provided as a prop to a React component on rendering, acting as the connective tissue between a node's view layer and the lower layers. Rete makes no assumptions about the structure of a control aside an id property.

During development it was noticed that dragging a node on the canvas would lead the react component to be "re-deployed", meaning state held in the component itself would be lost. To avoid state becoming lost or out of sync with what was in the actual node, we opted to introduce a mechanism for state management in nodes, the NodeControl<T> class, an extension of the rete Control class, where T is the type of the data object for containing the state of a particular node.

A templated class, NodeControl expects the user to provide an update function with signature (data: Partial[7]<T>) => void on construction. Rather than writing getters for each property of T, the .get(string) method accepts any string that is equivalent to a property of T, achieved through the keyof[8] keyword, returning a Readonly instance of the accessed property.

This pattern allows us to enforce the use of .set() to update the state of the controller as the readonly objects returned from the setter cannot be mutated. This forces the developer to do any side effects related to state change in the update function in the NodeControl, which makes it easier to debug as it can be found in a single location rather than spread across Node, NodeControl and the respective React component.

[7]Partial is a TypeScript type that accepts any object whose properties are a subset of type T
[8]https://www.typescriptlang.org/docs/handbook/2/keyof-types.html

NodeFactory

As nodes have a wide variety of constructor signatures and require contextually dependant functions to be injected during construction, it became necessary to provide classes with a simplified way of constructing them. The solution was to create a class with a single factory method taking an enumerator as an argument, each enumerator being linked to a specific node implementation. During construction of the factory, context appropriate functions are injected through the constructor and kept as private properties, which in turn are injected into the nodes on construction.

GraphSerializer

The GraphSerializer has two methods; One for exporting a graph as a JSON friendly object, another for importing a graph from a serialized object. During construction of the GraphSerializer, the owner object injects references to its own NodeFactory and NodeEditor instances, and optionally an AreaPlugin object as a graph isn't necessarily rendered, as in the case with Module graphs during data() invocation. As classes aren't inherently serializable, the node implementation is encoded as an enumerator, and during de-serialization the factory is used to instantiate the appropriate node and setting its data from the parsed string or object.

ModuleManager and Modules

The ModuleManager helps the GraphEditor keep track of its modules, or sub-graphs. By utilizing a GraphSerializer to serialize modules, it keeps them in memory tied to a name for lookup. If a user desires to edit a module, the GraphEditor can stash its current main graph in a private property and load one of the modules from the module manager instead, allowing different graphs to be visualized and edited.

Graph Translation

When exporting the graph for use in the web app, the nodes and connections in the GraphEditor are used as the input of an algorithm that translates into a parseTree. For most nodes, those inheriting ParseableBaseNode, this involves taking the data of type T from a NodeControl and using it to construct a ParseNode object. For ModuleInputs, ModuleOutputs and Module nodes there is an extra step involved.

As modules represent sub-graphs inside the larger graph it becomes necessary to *flatten* the full graph when translating it into a parseTree. In essence, this entails resolving incoming and outgoing connections with the respective ModuleInput and ModuleOutput nodes within the module graph before the flattened graph is transformed into a parseTree.

7.4 parseTree Library

The parseTree library found in the common module is the glue that ties the app and editor together, defining the data structure representing a calculation tool with its inputs and results, info text, names and other meta data.

Rather than being written with classes like the editor, parseTree is defined on functions and objects only, the idea being that it would mesh better with Redux state management due to it not playing nice with class objects. The main object type exported by the library is TreeState, an object that holds the main data structure in the form of a list of subtrees that are linked together through reference nodes. In addition to the tree itself, it holds arrays with references to inputs, outputs, displaynodes and a root node with meta data about the calculator tools name and version.

7.4.1 Data Structure

A challenge we faced was finding some way of encoding not just the formula of the calculator, which can have an arbitrary amount of inputs and outputs, but also meta information about how these should be displayed to the user. The data structure would also preferably have to be easily derived from the GUI representation of the formula.

The idea then became to tie such a node system to something closely resembling a parse tree, a directed graph structure where the numerical values from user inputs is transformed to an arbitrary amount of output values that then can be visualized. We opted to add a few enhancements to the traditional parse tree structure to allow easy management of shared data between the different outputs. The output data structure is a collection of trees where the root of each tree is either an output node, or a node where its output is used by multiple nodes, such as a variable shared between different output values. This way, for any given output, the data structure can be traversed as a tree. A leaf node on one tree may refer to the root of another, and in that way share data between sub-trees to avoid redundant calculation of branches. By giving each node an unique id, a node may be a reference to another through an id property. This then makes the data structure serializable to formats such as JSON.

7.5 Backend

Selecting a cloud provider requires a holistic assessment. Major providers such as Amazon AWS, Google Cloud, and Microsoft Azure all meet the requirements set forth in section 3.2.6 and the technical design in section 5.4, so technically they're all equally capable. The final decision was made in favor of Google Cloud for two primary reasons: A) the team's

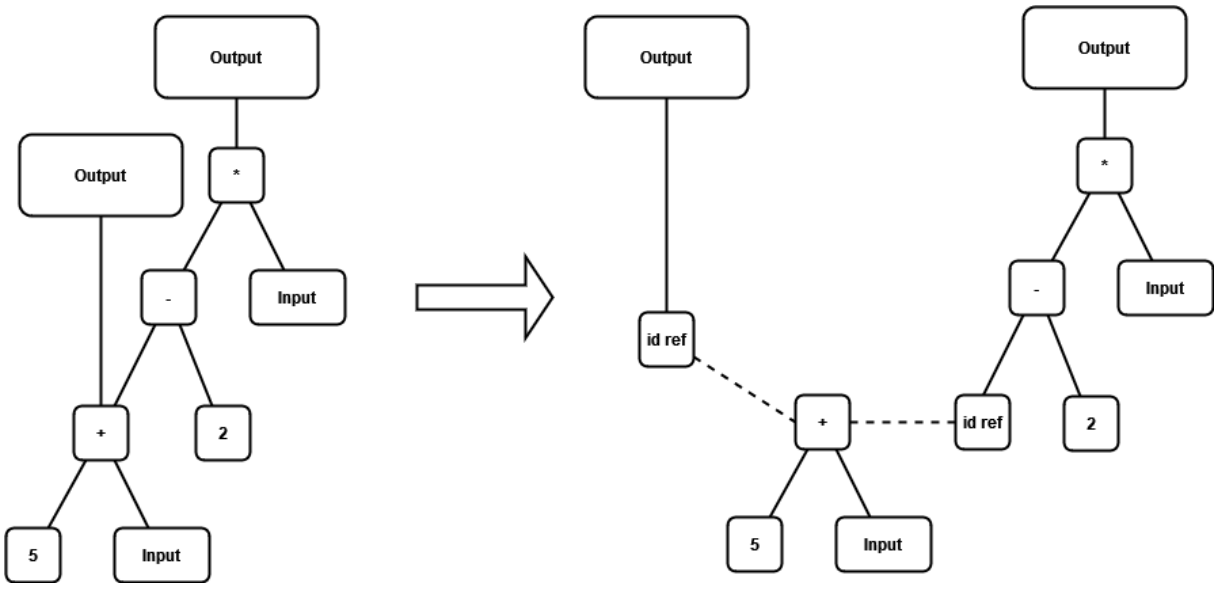


Figure 7.10: Graph as defined by the user split into serializeable subgraphs

familiarity with its Firebase database system, and B) Skogkurs already manages billing and administrative tasks through Google for promotional purposes, making the integration of additional services seamless.

The following services were utilized:

- **App Engine:** Serverless environment for hosting the API and static files (SPAs).
- **Firestore:** Document database for storing calculators.
- **Firebase Authenication:** IDaaS; a trusted intermediary between the editor and API

As part of project handover to Skogkurs, a systems operations manual has been written, detailing the configuration in Google Cloud. This document has been added as appendix B

7.5.1 API

As mentioned in the technical design (see section 5.4.3), the API functions as the entry point for for the database. The table below (see table 7.4) provides an overview of the available endpoints. Specifics on queries, headers, and the processes for data validation and transformation have been intentionally omitted for brevity.

Table 7.4: API endpoints

Endpoint	/api/v1/getCalculators
HTTP Method	GET
Authentication Required	No
Consumers	Web App & Web Editor
Description	Retrieves a list of basic information for every calculator
Endpoint	/api/v1/getCalculatorTree
HTTP Method	GET
Authentication Required	No
Consumers	Web App
Description	Retrieves the tree structure for a specific calculator
Endpoint	/api/v1/getCalculatorSchema
HTTP Method	GET
Authentication Required	No
Consumers	Web Editor
Description	Retrieves the configuration for a specific calculator
Endpoint	/api/v1/addCalculator
HTTP Method	POST
Authentication Required	Yes
Consumers	Web Editor
Description	Inserts a new or updated calculator into the database
Endpoint	/api/v1/calculate
HTTP Method	POST
Authentication Required	No
Consumers	Advanced users and partners
Description	Receives input values for a specific calculator and returns the results

Both web applications initially by sending a request to the *getCalculators* endpoint, to retrieve information on all calculators. Once a calculator has been selected by the web app or web editor, additional details are requested via the *getCalculatorTree* and *getCalculatorSchema* endpoints, respectively. This approach minimizes the processing required during retrieval, and that web application does not load in stale information in cases where the list have not been updated for a while. This is especially important for the editor. The only endpoint requiring authentication is *addCalculator*, which expect a token in the *Authorization* header of the HTTP request.

While all the endpoints mentioned thus far are designed for internal use by the web applications, the final endpoint, *calculate*, is a developer-friendly interface intended for public use. Although one might expect this endpoint to use the GET method, POST was chosen primarily for its support of sending data in JSON format, for several reasons:

1. **Readability:** Handling a large number of inputs can be cumbersome, especially when URL-encoded (percent-encoded) with non-Latin characters. JSON's structured format provides clearer syntax, making the data easier to read and debug.
2. **Nested data:** JSON allows for nested properties, eliminating the need to append page names to query keys, thereby simplifying the data structure.
3. **Data conversion**: Converting objects to JSON and parsing them back is generally easier and less error-prone than concatenating and splitting query strings.

Code listing 7.1: API Controller with Error Handler

```
// Handler for fetching metainfo on all calculator versions in the database
export function getCalculatorsInfo(db: IDatabase) {
    return async function(_req: express.Request, res: express.Response) {
        await respondToAsyncResult(db.getCalculatorsInfo(), res)
    }
}

// Helper function that resolves a operation and send the appropriate response to the caller
async function respondToAsyncResult(operation: Promise<any>, res: express.Response) {
    operation
        .then(result => { res.status(200).send(result) })
        .catch(e => {
            if (e instanceof BadRequestError) {
                res.status(400).json({error: "Some of your input is invalid: " + e.message})
            } else if (e instanceof NotFoundError) {
                res.status(404).json({error: "The resource you requested was not found: " + e.message});
            } else if (e instanceof DatabaseError) {
                res.status(500).json({error: "We are experiencing technical difficulties. Please try again later."})
            } else {
                res.status(500).json({error: "An unexpected issue occurred. Please try again later."})
            }
        })
}
```

Controller implementation

Code listing 7.1 shows how one of these endpoints (depicted as `getCalculatorsInfo`) are defined. Controllers, which handle endpoint logic, are implemented as closures[9] that take dependencies as arguments, effectively injecting them into the function for promoting modularity

[9] A closure is a function that retains access to variables in its outer scope, even after the outer function has finished executing. This means the inner function can continue to use and modify these variables across subsequent calls. [30]

while adhering to the format expected by the Express library. The function parameter includes the IDatabase type, which is an interface that facilitates dependency injection as mentioned in section 5.4.1. This interface allows for polymorphism, similar to abstract classes. Consequently, other database implementations that conform to the same interface can be easily integrated, enabling seamless migration to a different database without requiring changes to the controller.

Error handling

In the same code listing, the `respondToAsyncResult` function serves as a shared utility across all controllers, specifically for catching and managing errors. The standard `Error` class has been extended to semantic error types (`BadRequestError, NotFoundError, etc.`), as a means to classify and handle errors based on their type, which indicates the severity of the issue and informs how it should be communicated.

Most operations within the controllers are chained together by binding `Promise`[10] types together. This pattern is often referred to as *monadic error handling* [32]. The major benefit of this pattern is that errors can propagate through the chain in a controlled manner.

Code listing 7.2: Technique known as "monadic error handling"

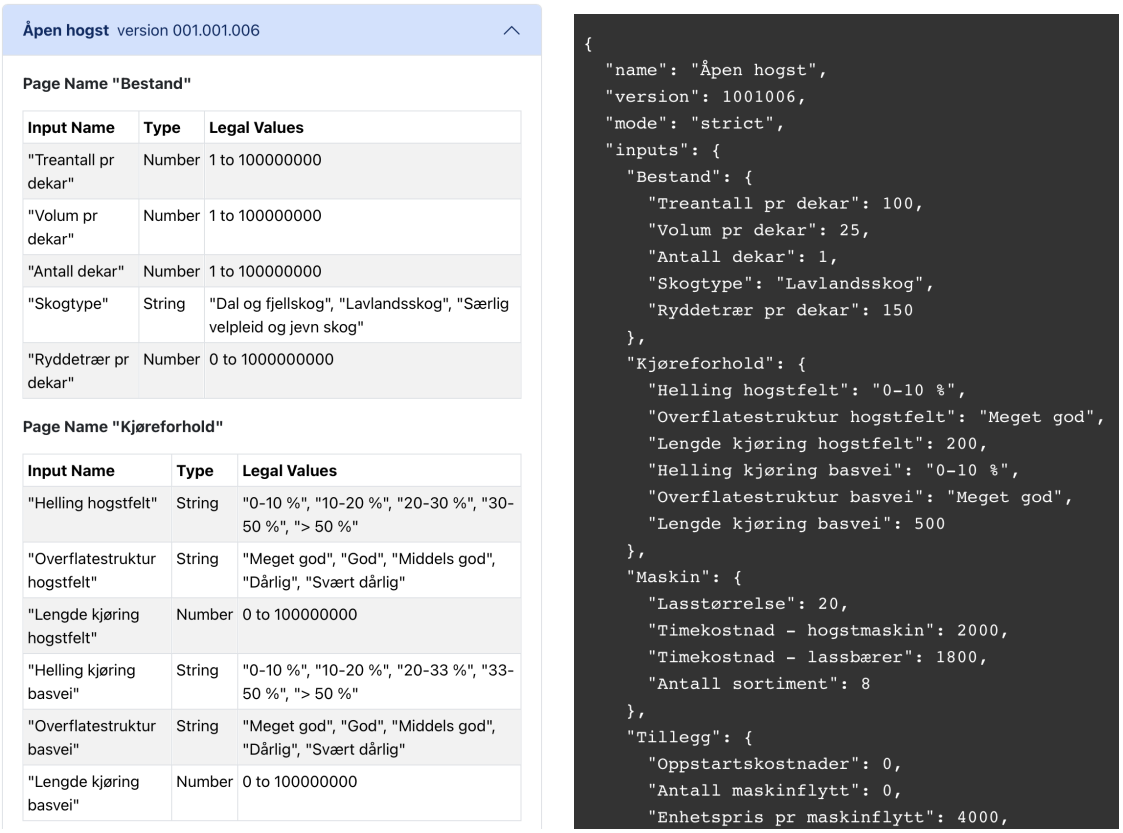
```
await respondToResult(auth.verifyToken(idToken).then(() => db.addCalculator(c)), res)
```

Using code listing 7.2 as an example, supposing `auth.verifyToken` is rejected with a `AuthenticationError`, then the following operations in the `.then`-path (i.e. `db.addCalculator`) would be skipped/"short-circuited". Instead, the error will get caught by the last `catch` in the path. In this case the `respondToResult` function will transform the rejected promise to an appropriate HTTP response based on the error type. This pattern is not just semantically easier to work with, but also highly maintainable.

Auto-generated API Documentation

Because calculators are dynamic structures that are susceptible to change through the editor; writing documentation is not as straightforward as compared to services where the inputs and outputs are predetermined. Consequently, the *API information page*[11] must auto-generate documentation for each calculator defined in the database. Even though this might appear complex at first glance, it's actually very simple, because all the information, including value types and ranges are embedded into the calculator for dynamic rendering. As such, it just becomes a matter of presenting the same information in descriptive form.

[10] Promises are objects that represent delayed result of asynchronous functions [31]
[11] `https://kostnadskalkulator.skogkurs.no/apiinfo`



Åpen hogst version 001.001.006

Page Name "Bestand"

Input Name	Type	Legal Values
"Treantall pr dekar"	Number	1 to 100000000
"Volum pr dekar"	Number	1 to 100000000
"Antall dekar"	Number	1 to 100000000
"Skogtype"	String	"Dal og fjellskog", "Lavlandsskog", "Særlig velpleid og jevn skog"
"Ryddetrær pr dekar"	Number	0 to 1000000000

Page Name "Kjøreforhold"

Input Name	Type	Legal Values
"Helling hogstfelt"	String	"0-10 %", "10-20 %", "20-30 %", "30-50 %", "> 50 %"
"Overflatestruktur hogstfelt"	String	"Meget god", "God", "Middels god", "Dårlig", "Svært dårlig"
"Lengde kjøring hogstfelt"	Number	0 to 100000000
"Helling kjøring basvei"	String	"0-10 %", "10-20 %", "20-33 %", "33-50 %", "> 50 %"
"Overflatestruktur basvei"	String	"Meget god", "God", "Middels god", "Dårlig", "Svært dårlig"
"Lengde kjøring basvei"	Number	0 to 100000000


```
{
  "name": "Åpen hogst",
  "version": 1001006,
  "mode": "strict",
  "inputs": {
    "Bestand": {
      "Treantall pr dekar": 100,
      "Volum pr dekar": 25,
      "Antall dekar": 1,
      "Skogtype": "Lavlandsskog",
      "Ryddetrær pr dekar": 150
    },
    "Kjøreforhold": {
      "Helling hogstfelt": "0-10 %",
      "Overflatestruktur hogstfelt": "Meget god",
      "Lengde kjøring hogstfelt": 200,
      "Helling kjøring basvei": "0-10 %",
      "Overflatestruktur basvei": "Meget god",
      "Lengde kjøring basvei": 500
    },
    "Maskin": {
      "Lasstørrelse": 20,
      "Timekostnad - hogstmaskin": 2000,
      "Timekostnad - lassbærer": 1800,
      "Antall sortiment": 8
    },
    "Tillegg": {
      "Oppstartskostnader": 0,
      "Antall maskinflytt": 0,
      "Enhetspris pr maskinflytt": 4000,
```


(a) Generated parameter specifications **(b)** Generated usage example

Figure 7.11: Auto-generated API specifications for a calculator, shown in the web app

Automatic Scaling

Operating in a containerized environment, the API is designed for automatic horizontal scaling[12], utilizing an instance class of F2[13]. At least one instance remains active at all times to meet standard demand. For traffic spikes, an additional instance is always on standby in a hibernated state, ready to be activated within seconds. The system's total capacity is configured to three instances, a threshold that should rarely, if ever, be reached.

7.5.2 Database

While the document database Firebase is technically capable of handling authentication, data validation, and transformations directly from the SPAs through various mechanisms, introducing an API as a single interface significantly simplifies the design, and substantially enhances the security by restricting connections solely to the API.

Firebase is inherently schema-less, but enforces an alternating hierarchy of *documents*,

[12]Horizontal scaling means that increased demand is met by launching additional instances rather than boosting the performance of a single instance. This approach is often more cost-effective.

[13]F2 classes are specified with a 1.2 GHz CPU and 768 MB of memory.

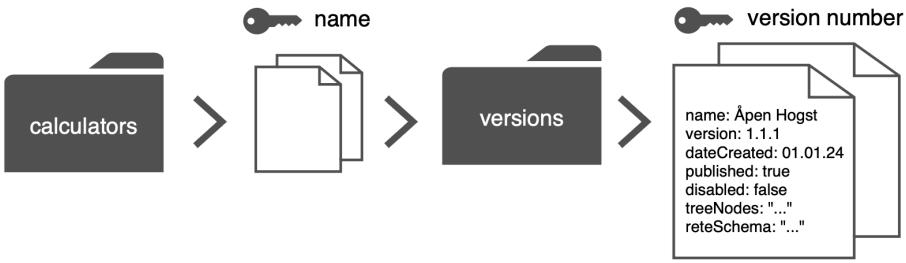
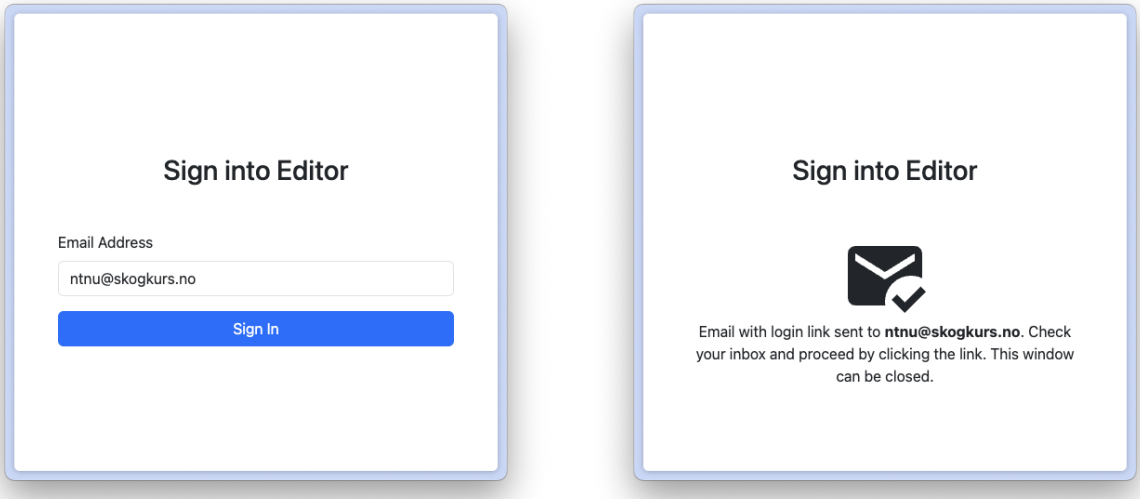


Figure 7.12: Document structure

which store data in key-value pairs, and *collections*, which are lists of documents. This setup prevents direct relationships between two documents or two collections. Figure 7.12 demonstrates how calculators are organized within these constraints. Most notably, documents within the *versions* subcollection include segment paths (name and version) as redundant data, which is generally not encouraged. However, this redundancy is strategic, as it greatly simplifies queries, particularly for collection groups (matching all subcollections in the database named "version"), without transformations or query composition. A key advantage of using the Firebase SDK is its ability to automatically convert data to and from JavaScript objects. Nonetheless, it faces a limitation with a maximum depth of nesting, which can lead to complications when creating complex calculators. To address this, configuration and tree structures are securely stored as JSON formatted strings instead.

7.5.3 Authentication

The only action that requires authentication is the export of calculators from the editor to the database via the API. If a valid authorization token is not found, the editor automatically redirects to the sign-in page at launch (see figure 7.13). This redirection is a user experience feature designed to prevent users from spending significant time editing a calculator only to find out they are not authorized to save it. However, this redirection is not a security measure for entering the editor, and can be easily bypassed by modifying the client-side code.



(a) Editor Sign-in page **(b)** Editor Link sent confirmation

Figure 7.13: Signin page in the Web Editor

Authorized users must have their email addresses pre-approved by the administrator (see Section 3.3) via the Firebase Authentication web console before logging in. If not pre-approved, they will receive an error message similar to the one shown in figure 7.13b instead. As detailed in the technical design chapter (see Section 5.4), the process of obtaining an authorization token operates entirely without API intervention. The sign-in process is completely password-less, removing the need for users to remember passwords. Instead, they confirm their identity using a time-sensitive, one-time confirmation URL sent from the Firebase Authentication backend. This link directs back to the web editor and includes a key used to verify the user and obtain an access and refresh token. Apart from the graphical interface, the Firebase SDK handles everything, including securely storing the tokens in the browser. The access token is then added to subsequent API calls to the `addCalculator` endpoint in the authorization header of the HTTP request.

Chapter 8

Testing and Quality Assurance

This chapter details and discusses our approach to testing of code, user testing of the application, and evaluating the mathematical equivalency of our implementation compared to the original calculator.

8.1 Unit Testing

8.1.1 Coverage

While some argue for setting specific goals, like 90 or 100% coverage, Khorikov [33] cautions against setting hard goals due to how it may influence developers into creating tests for the purpose of hitting targets rather than actually writing meaningful tests. A test may very well have full coverage while asserting very little about how your code behaves, or how it should behave. One should work towards having a solid coverage, but one should prioritize writing meaningful tests that cover important non-trivial code.

As a counter argument, setting goals could help students who aren't used to testing actively write tests, but as students we might be even more susceptible to writing tests for the purpose of hitting the target more so than someone with proper experience. We opted to take a pragmatic approach where we had no hard coverage level as a requirement for code to be accepted as done in a sprint, but with the caveat that coverage below 80% would be a strong indicator that we needed to pay better attention to testing.

By the end of the project, overall coverage for the editor was at 63% due to React components not being tested, wheres as testing of remaining code is in the 80-90% region. Coverage reports for the API and editor can be found in appendix P.

TDD

Test driven development was utilized periodically throughout the project, with varying degrees of adoption within the group.

Challenges

Staying on top of testing wasn't always easy. When we felt the time crunch it was always tempting to forego TDD, or even testing, in favor of "getting things done". This did of course end up catching up with us. A major oversight in the GraphEditor class was the fact that what worked from the perspective of a user was not necessarily easy to verify through tests. The main culprit were a group of methods on the rete.js NodeEditor class, highlighted in orange in appendix G, where adding nodes to the graph was done asynchronously. As a consequence, this async behaviour had bled into a large part of the code base, and as testing had not been done for a while, it did not occur to us that this might be a problem down the line.

Had the code been written in a way that properly propagated async code it would have been a matter of using await in the tests, but in a lot of methods the async calls were done in synchronous functions or methods with some logic inserted into .catch() and .then() to handle the result of the Promise. As these functions would return before the promise was resolved made it impossible to actually test that the code did what it was intended to do.

This problem greatly increased the difficulty we had with implementing sub-graphs, or 'Modules', in the editor; Graphs nested inside a single node with in and out connections corresponding with nodes in the graph it contained, essentially allowing the user to define reusable functions. The algorithm for connecting the inputs and outputs of a sub-graph with the corresponding nodes in the outer graph was hard enough to implement, but when the tests written for the algorithms were failing, it was hard to tell why they failed, as at this point many tests would have failed even if they tested for behaviour we already knew to be working due to the async problem.

Reflection

A big takeaway from the experience with the GraphEditor class is that what might seem like a trivial test on the surface is actually a test you will need as a foundation later on. Complex behaviour built on units of simple behaviour is hard to test if you haven't ensured that the foundation itself is testable. Not testing 'trivial' code is also problematic if you do not have enough experience with a programming language to determine whether the behaviour is actually trivial or not.

We had a single person working on making the modules function along with rewriting a lot of the code base to be testable for at least two weeks, which is a large amount of time in such a short lived four-man project. Writing those particular tests earlier would almost certainly have halved the time expenditure on the feature, if not more.

8.2 User testing

As a supplement to the regular feedback from the PO and stakeholder, it was perceived as useful to conduct dedicated user testing, in order to solicit input from real members of the potential user groups. The hope was that this would uncover problems and challenges with the product that might otherwise have gone unnoticed, as well as a broad confirmation that development was on the right track.

For the web application this took the form of a robust, formal user test, with defined areas of research, multiple users tested in a easily observable fashion, resulting in a comprehensive report documenting findings, reflections and suggestions for improving the uncovered problem areas.

Due to time constraints, the same was not the case for the web editor. The only "user testing" that was carried out for this component of the software, was noting the observations given by representatives from Skogkurs as part of a demonstration of the web editor towards the end of the development period.

The following section will describe and critique the methods used and results obtained from these tests, emphasising our evaluation of the total value of these activities.

8.2.1 MVP Test summary

A usability test of the MVP version of the web app was performed, mainly focusing on user interaction and navigation. A qualitative test was selected, where users' subjective impression of the app was investigated. Skogkurs provided us with contact info to potential users in the forestry industry. The test was performed with six users representing various forestry professions. Whether or not the selected users are a representative sample of the total user mass is difficult for us to assess, but different professions, age groups and geographical areas were represented. This way, the testing could be performed with unbiased users, previously unfamiliar with the project, as recommended in the course book from IDG1362 (User Centered Design) [34].

Test setup

The application was tested by users on desktop, with an emulated mobile unit. The reasoning behind this choice is that the applications' user interface at the time of testing was best adapted for the mobile format. Another aspect is that the app environment would be as equal as possible for all users. To facilitate a flexible testing setup, where users could attend physically or remotely via Teams, maintaining observability for the testing team.

Each test was split into three parts.

- Introductory questions: The user was asked some questions about digital units and

software used in the field. The purpose of this was getting an overall impression of the level of digitization in the forestry industry.

- Practical tasks: The user was given specific tasks to solve in the application. The test team observed the user working in the app, while taking notes of observations and citations from the user. Tasks were designed to be as realistic as possible, but also kept small enough to fit the time frame. As an example, one of the tasks was to fill in some values (not all) in each input page, and then view the results. This way, the user got to try both drop down and number input fields, as well as navigation between input pages and the results page. Other tasks/questions were more aimed at getting feedback on specific details, such as the application's response to erroneous input, or the location of information.
- Follow-up questions: The user was asked questions about the application itself, both as a whole and details not handled specifically in the practical part.

Post processing

Categorization and aggregation of data consisting of citations from users and our own observations was perhaps the largest issue. Compared to a purely quantitative test where results can be summed up in numbers, a qualitative test is more difficult because small nuances may affect interpretation. The categories used were also partly decided during the work with the report, because we opened up for feedback initially out of scope of the test. The task of writing a test report turned out more tedious than expected, but we think we managed to pinpoint the most important insights.

Results

- Majority of users say that the application is mainly well-designed, but navigation between input pages was difficult for some users.
- Most users found the requested information without difficulty.
- Users mainly understood the result page, but the pie chart for productivity was confusing, as the numerical value for only one machine was shown.
- More users wanted a function to save or share calculations.
- Users were mainly satisfied with the amount and content of feedback on interaction.
- Users had differentiating opinions whether the app should be as simple as possible, or as correct as possible.

The full test report is included in appendix C.

8.2.2 Reflections on user testing

Positive experiences

The majority of users could not meet us in person, and therefore most of the tests were performed via a Teams meeting, where the user shared the screen for us to observe. Doing the tests in a Teams meeting was a very positive experience. The test environment functioned as planned. We believe that from the users' perspective, a test performed in this way feels like a lower effort than having to meet in person. The users were generally very positive, and some even offered to participate in later user test if needed.

Negative experiences

Most of the test questions and tasks were intended to be quite specific. However, the users tended to go outside the scope of many questions when answering. We believe this is natural, and also beneficial, because these answers will provide insights that we hadn't thought about. At the same time, it made it more challenging to filter out the relevant insights. For Skogkurs, it is of importance that the calculator is based on research, which dictates most of the input parameters. However, much of the feedback we got from the users were suggestion of new parameters or simplifications that should be added to better suit the users' workflow or needs for accuracy, which we considered to be beyond the project scope. The users were informed in advance of the test about which were our areas of interest. But we also understand that the user has a need to express their requirements. Many of the users were passionate about their profession, and to keep the conversation during the tests as natural as possible, we let the users express these suggestions, maybe more than we ideally should have done.

Some of the prepared questions were interpreted differently by the users than intended. The answers given to these questions would therefore sometimes offer limited value. As an example, we wanted to test if the users noticed that values they had typed in were rendered in bold text. The question we asked was "do you notice any difference between the field you filled in and the others on the page?". The answers were often an explanation of the meaning of the input fields, rather than an assessment of visual appearance. As half of the project group were involved in writing the questions, we could have done a sample test with the other two members. Then, we would have a greater chance of avoiding these ambiguous questions, and end up with more precise feedback.

In some cases, we may have given the user more guidance than we should. But when the user asks a direct question, e.g. "is this the correct place?", it is difficult to avoid answering without the conversation feeling unnatural. These cases were however very limited, and we do not believe that they have affected the test results in a significant extent.

What could be done differently?

The timing of doing user tests is a challenge. When working with agile methods, the product is constantly evolving. There may always be some functionality one would wish to test. As the tests were performed over a whole week, we had to “freeze” the published version for that time period. Changes to the source code were constantly implemented during that time in another branch. A challenge with this is that functions wanted by many users in the user test may have been removed for reasons that are valid for the developers, but which is essential for the user to even consider using the product. Furthermore, added functionality unwanted by the users could have been implemented at the same time.

These situations may pose a risk because parts of the code may have to be rewritten, or even scrapped. A possible mitigation could be to stop development of the functionality or design related to what is being tested during the test period. In our case, there were no major differences in implemented functionality and the user test result and therefore no problems arose. The testing team and the developing team at the time might have focused too much on their own tasks, resulting in a lack of coordination. Some functionality was also added a very short time before the testing period, and therefore questions or tasks related to this were not made.

After the user test period, we have identified some more areas of interest that we feel we should have tested. For example, we did not ask the users to state the most positive or negative features of the app. By asking this, we could have gained more insight into what is most important for the users. The user’s natural workflow was not tested either. We should have some insights in how the application would fit into the users’ workflow. Examples of testing points could be “what preparations are done before opening the app?”, “ is the app opened and closed multiple times to gather and input all values?”, or “how are calculated results handled?”. These questions could have helped us seeing the larger picture of how the application actually will be used, which may be essential to the user adopting the app in the first place.

To sum up, we believe there is no perfect time to do user testing. There will always be more to test. To minimize the problems outlined above, the testing period should probably be kept as short as possible. Communication between the different teams is also essential for a well coordinated test period with minimal chance of having to redo the implementation.

Utility assessment

An interesting observation is that we got much of the same feedback in the user test, as we did via other channels approximately at the same time. Preparation, execution and evaluation of the user test was very time consuming. Therefore, one could argue that doing the user testing took too much time, while providing little added value. However, much of the

more informal input came from persons involved in the project from the beginning. Therefore, we think that it is of great value to get input from external users who hadn't seen the application prior to the usability test. The user test also gave us the information in a more organized manner. Informal input has been our main source during the project, and that the user testing supports this to a large extent is of great value as a validation of the choices we have made. We also got some new input, which we otherwise might not have gotten. For learning purposes, it was also a valuable experience.

8.3 Mathematical Equivalency Evaluation

As we shift our financial calculator to a new platform, it's essential to ensure it still delivers the expected results. These outcomes are key in making decisions about the economic viability of harvesting operations. While the calculator is meant to provide estimates, we need to carefully check that there are no significant errors in the calculations due to the transition.

8.3.1 Methodology

Given the large number of inputs, automated testing emerged as the only practical approach to verify equivalence between the two systems. We conducted 1000 automated tests, amounting to a total compute time of 20 minutes on a high-performance workstation, most of it attributed to processing Excel. The following steps outline the method used:

1. **Generate test data for inputs, and insert them into the tree.**
   The blackbox testing technique employed is known as 'random testing.' This method involves the program generating random, independent values for every input. Dropdown inputs are selected entirely at random, while numeric inputs are constrained to a range of 1-1000. The rationale for this specific range is detailed in the Limitations section below.
2. **Document the results from the tree state.**
   Four results were collected: Harvester productivity, harvester cost, forwarder productivity, and forwarder cost.
3. **Export the corresponding inputs and results to a CSV file.**
   Saving the results will conserve computational power for future runs and enable test repeatability.
4. **Individually map the inputs from the CSV file to the associated cells in Excel.**
   Python is used as the scripting language for integrating Excel into our tests, due to its extensive and impressive statistical libraries.
5. **Compare the results.**
   The metric chosen for evaluating deviation is percentage change. v_1 represents the old

value, and v_2 represents the new:

$$\text{Percentage Change} = \frac{\Delta V}{V_2} = \frac{V_1 - V_2}{V_2} \times 100\%$$

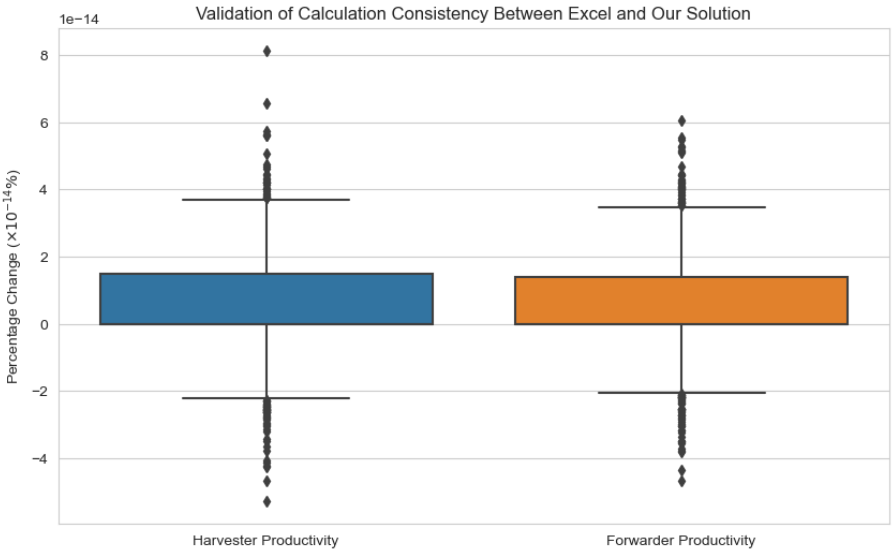


Figure 8.1: Relative change in results between Excel and our tree representation

8.3.2 Results

Across 1000 tests, the mean percentage change in calculated values between the legacy Excel spreadsheet and the new platform was 4×10^{-15} %, indicating extremely small rounding errors. The maximum observed difference was 8×10^{-14} %, with a standard deviation of 1.6×10^{-14} %. These results demonstrate the high level of accuracy maintained by the migration from Excel to a tree-based structure, ensuring mathematical equivalence between the legacy Excel spreadsheet and the new platform.

Figure 8.1 illustrates the distribution of percentage change in productivity results calculated with high decimal precision. In contrast, cost results are rounded down to four decimal places in the legacy Excel document, which when the original results are adjusted to the same precision, are found to be exactly equal. In conclusion, the difference in results are negligible, resulted from rounding error, which either way will be rounded further before being presented to the user.

8.3.3 Test Limitations

These tests are not designed to handle edge cases such as division by zero errors. Input nodes within the tree have custom legal ranges assigned to them; e.g. zero may be permissible in some scenarios but not in others. Currently, there is no upper limit for inputs.

For simplicity, the randomization of values for all numbered inputs is confined to the same range. To prevent integer overflows, which could compromise the tests, we have set an arbitrary upper limit of 1000. This number serves as a middle ground, accommodating scenarios where values might be considered high or low. Additionally, a lower bound of 1 ensures the requirements that some input must be positive. It is important to note that while the tests provide comprehensive coverage within these ranges, they do not address all possible scenarios, including extreme values outside the specified range

Chapter 9

Deployment Workflow

This chapter details the process from code organization to deployment. It covers using a monorepo to streamline code sharing and deployment, followed by the trunk-based branching strategy for continuous integration and deployment. Finally, it explains the CI/CD pipeline, emphasizing automation of testing and deployment.

9.1 Code Organization

To streamline deployment and facilitate code sharing among different packages (i.e. the web app, web editor, common, API), all the code has been organized into what is known as a *monorepo*[1] on GitHub[2]. This single repository consolidates all interdependencies, ensuring that changes across the project are atomic and consistent. This means that a single commit can encompass all necessary updates across dependent packages, rather than implementing partial, isolated changes [36]. As a result, the development and deployment process becomes much simpler to manage, as there is no need to handle version control complexities between the packages.

On the other hand, the flexibility offered by a monorepo can make it harder to preserve loose coupling, which is generally recommended. While polyrepos encourage the use of simple interfaces for interaction by necessitating clear versioning and separation, these boundaries are less pronounced in a monorepo. This lack of strict versioning in a monorepo can come at the cost of blurring the lines between packages, making it harder to maintain their separation.

npm[3] *workspaces* is used to manage the interdependencies and streamline development within the monorepo. This npm feature treats multiple packages as a single unit, simplifying package management. It centralizes the installation and linking of external dependencies at

[1] A repository that contains more than one logical project [35]
[2] https://github.com/oysteinqvigstad/SKOG-kostnadskalkulator
[3] A package manager coupled with the runtime environment Node.js

the root level, avoiding redundant installations for each package.

9.2 Branching Strategy

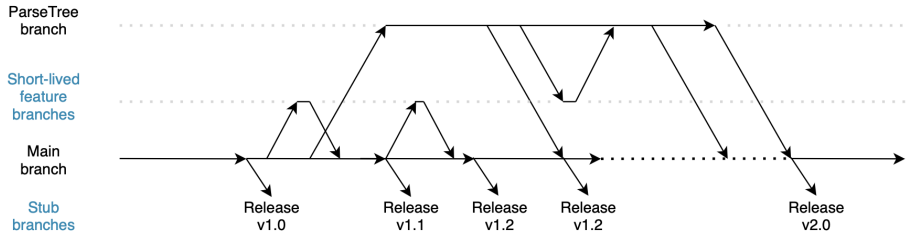


Figure 9.1: Branching Strategy

The repository largely follows a *trunk-based* branching strategy, distinct from *feature branching* or *GitHub Flow*. In this approach, most of the development is done directly on the *main* branch, promoting continuous integration; which involves frequently integrating individual pieces of work. This approach helps prevent the complexities and challenges associated with merging large, divergent sections of code later on. While we occasionally use feature branches for brief, specific tasks, the notable exception has been the release 2.x development branch, depicted as *ParseTree branch* in Figure 9.1. As mentioned in section 7.2.1, release 1.x was built with hard-coded linear data structures and mathematical functions in plain JavaScript. In contrast, release 2.x introduces more complex tree structure, which brought significant changes to state management. Initially, it seemed practical to keep these versions separate, whilst being developed partially in parallel.

One challenge with the trunk-based approach is that it makes peer-reviewing impractical. However, given our tight timelines, comprehensive peer reviews weren't feasible anyway. The main advantage of using trunk-based branching is not just continuous integration, but also continuous deployment (CI/CD); An automated strategy that ensures the latest code versions are seamlessly deployed to production servers, significantly shortening the feedback loop with stakeholders and speeding up patches and feature testing.

9.3 CI/CD Pipeline

The CI/CD pipeline serves a dual purpose: to perform automated testing on the codebase to identify potential issues, and, assuming all tests pass, to deploy the updated version into production. Every push to the main branch, regardless of size, triggers this pipeline. The pipeline is structured into three consecutive stages, with a failure in any stage halting the process:

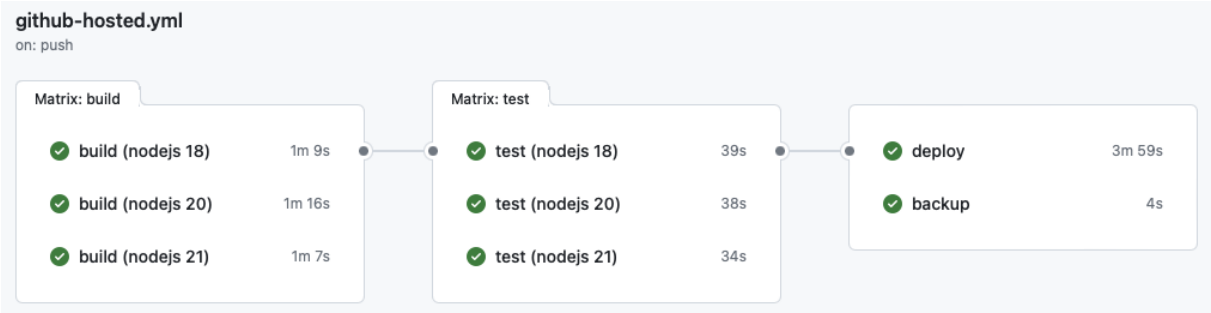


Figure 9.2: Github CI/CD Pipeline

1. **Build**: The code undergoes syntax checks, and a strict linter is applied to ensure code quality.
2. **Test**: All automated tests must pass. This stage acts as a crucial "sanity check" due to the interdependencies among the TypeScript packages.
3. **Deploy and Backup**: A new AppEngine container is created in Google Cloud. Upon successful deployment, traffic is redirected to this new container. Simultaneously, a backup job duplicates the commit history to a GitLab IDI server.

Furthermore, the first and second stages of the pipeline utilize a matrix strategy within GitHub Actions. A matrix job is a task that runs in parallel under different parameters. For us, this means building and testing across various versions of the build target and package manager. Currently, the CI/CD matrix includes the following versions of Node.js:

1. **Node.js 18**: The previous Long-Term Support (LTS) version.
2. **Node.js 20**: The current LTS version, also used on our production server.
3. **Node.js 21**: The latest stable release.

This approach boosts our confidence that the software remains versatile across different runtime environments. It also increases the likelihood of successful upgrades or downgrades as needed. The runners[4] are hosted by GitHub, which offers unlimited usage for public repositories. We also experimented with self-hosted runners on SkyHigh[5], but observed no significant performance difference. This lack of discrepancy is likely because GitHub's hosted runners are already highly optimized.

Deployment to Google Cloud's App Engine is the last stage of the CI/CD pipeline. This process is controlled by the verified Google GitHub Actions[6] package from the GitHub Marketplace[7], which appears to be a community project, with high activity, mostly from employees at Google. The authentication key to access Google App Engine is associated to a

[4]The servers responsible for executing the tasks/jobs in the pipeline
[5]An NTNU Gjøvik hosted IaaS-platform. Each bachelor project is allocated 16 CPUs and 64GB RAM with VM capacity. We only used it for experimental purposes.
[6]https://github.com/google-github-actions
[7]An app and actions library for extending and improving GitHub workflows

dedicated service account with minimum privileges required for deployment. This key is safely stored in GitHub Secrets; an environmental variables bank that can only be read by GitHub Actions runners.

Chapter 10

Security

Designing a secure application for a real world scenario requires careful considerations on the different components that make up the application. With basis in the CIA triad[37] and common principles of security, this chapter gives an overview of our approach for securing the different parts of the application, as well as tools and methods used.

10.1 Security principles, examples

A series of security design principles have been taught in various courses over our study, for instance in Software Security (IIKG2001). These principles have been used throughout the project as a framework when deciding what security measures to implement With basis in these principles as described in [38], we want to give some examples of implemented security countermeasures related to each principle.

- Defense in depth: Client must go through API to access the database. To alter the database, authentication is required.
- Fail secure: Error handling is implemented in the API
- Least privilege: The service accounts for deployment are set up with minimal rights. This applies both from the repository to Google App Engine, and from App Engine to Firebase.
- Separation of duties: We do not differentiate on users in the cloud service, so this is not covered.
- Economy of mechanism: We have tried to reduce the attack surface by utilizing IDaaS (identity as a service).
- Complete mediation: Authentication is checked every time one tries to do database operations. There is however no logging to track who stored what in the database.
- Open design: The software is planned to be open. No secrets are stored in the code.
- Least common mechanism: The code base only contains one privileged operation

(storing to database).

- Psychological acceptability: Authentication through email offers editor users a straightforward login system.
- Weakest link: The Google Cloud admin account may act as a single point of failure. Breaching this account lets attackers add allowed users to the editor client. Also, all administration of the Cloud account is available.
- Leverage existing technology: Throughout the project, we have taken extensive use of existing libraries. For security specific functionality, the most notable is the use of IDaaS.

10.2 Automated security audits

Automated security auditing of the code base can be a useful tool for detecting vulnerabilities, including dependencies, that are otherwise hard to uncover. Changes and deprecation of packages used may happen, and therefore the code base can be secure today, but may be vulnerable tomorrow. It should be noted that while security scans are automatic, updates are not. In the future, handling new reported problems may require some investigation and will have to be done manually. Below is a description of the tools used, as well as actions taken based on the scan results.

10.2.1 NPM audit

When using NPM to manage JavaScript/Typescript packages, project dependencies are checked against a package register via "NPM audit" [39]. The register contains information about each package in the project, among others about known vulnerabilities.

A challenge with many packages is that dependencies are nested in multiple layers, often resulting in a fan-out of dependencies. The libraries used in the project have transitive dependencies, and therefore many potential sources of vulnerabilities. NPM audit may report a vulnerability in a transitive dependency. Because the main library is often only tested with a specific version of the dependency, updating a transitive dependency using the fix suggested by NPM audit may introduce breaking changes [40]. The underlying dependencies are managed by the package maintainer, and emerging vulnerabilities should be handled by them.

The key takeaway is that developers' control normally stop at the first level of the dependency graph. When maintaining an app over time, discovering new vulnerabilities in the packages is inevitable. Ideally, the package maintainer should update their package. If however the package gets deprecated, the developer should look for other alternatives.

The security warnings from NPM audit increased in numbers over the course of the

project, and at first we were not sure how to correct them.

At the start of the project, Create-react-app was selected as build tool because we had experience with it from earlier. The Create-React-App (CRA) project was developed to get users started quickly with React projects [41], and it was also recommended by the creators of React itself [42]. To offer a one-size-fits-all solution for creating React apps, a large collection of dependencies are included by default [43]. As a consequence, users only needing a subset of the included functionality have to install a large number of dependencies that they really do not need. Sources online suggest that vulnerabilities reported by NPM audit are less critical than it may appear because the dependencies are only used in the build process[44] , but we do not have the qualifications to assess the validity of these suggestions. From a security perspective however, keeping dependencies at a minimum should be advised, as the attack surface can be kept smaller that way; more complexity often leads to more errors.

We later found out that CRA that we had been using is deprecated since 2023. NPM audit reported many vulnerabilities related to this, but simply updating them to newer would almost cetrainly introduce breaking changes, as CRA is not tested with the new versions.

After this discovery, we switched to a newer build tool, called Vite [45]. Vite as a build tool has a similarly simple approach for the inexperienced user as CRA, but under the hood, Vite differs from CRA in the way dependencies are handled. Where CRA process all dependencies at build time, Vite process dependencies on demand. The build times and performance are therefore significantly better. A significant amount of our original dependencies used by CRA only could be deleted from the project, which resulted in elimination of many of the vulnerabilities. After updating some dependencies unrelated to Vite, all vulnerabilities reported by NPM audit were resolved. So switching not only solved the original problem, it enhanced the build process as well.

10.2.2 GitHub security checks

GitHub offers a variety of automated tools for detecting security problems. A security scan is run whenever new code is pushed to the repository. Potential problems are listed under a "Security" tab in the repository, along with information about the severity as well as suggested solutions. The security checks are run when pushing to the configured branch, and additionally as described below. A basic setup with the following features was used in the project:

- **Code scanning:** The code base is scanned for errors and vulnerabilities [46]. Code scanning is in this project set up to use GitHub's own code analysis tool "CodeQL", while there is an option to use 3rd party tools as well. Using a basic setup, configuration is automatic, based on the code base at hand. Code scanning runs at weekly

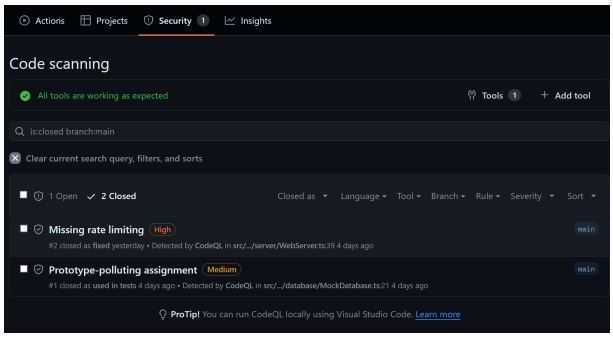


Figure 10.1: Example from GitHub Code scanning. The alerts have later been closed.

intervals.

- **Dependabot:** The project's dependencies are monitored and compared to the "GitHub Advisory Database" to check for vulnerabilities [47]. Dependencies are continuously monitored to detect emerging vulnerabilities.
- **Secret scanning:** The code base is scanned for secrets such as API keys and other credentials [48]. This offers protection against accidentally pushing secrets to the repo, exposing them to the public.

Figure 10.1 shows the result of a scan. The "Missing rate limiting" is a problem in need of solving. Similar functionality is present in the hosting platform, where a max limit of requests per IP address per minute is set, but this external functionality is not detected by CodeQL. Solving the problem in the app itself seems like a good solution because the external functionality may fail. Having most of the security built into the code base also can make migrating to other hosting platforms easier, since less configuration is needed. The solution was to this particular problem was to set up a basic rate limiter using "express-rate-limit" [49]. The other problem listed is ignored, because it is only related to parts of an internal test and therefore will not be exposed in the finished app.

10.3 Deployment / backend

To ensure a simple and safe deployment, a serverless solution is used. A trusted 3rd party provider manages all the infrastructure including network, storage, operating systems, scaling, and server security. Operations are thereby simplified, and offloads server administration from the service owner (Skogkurs). For the development team, the necessary administrative tasks in such a service are limited to setting a few environment parameters, before deploying the code.

Keeping the entire app within the Google Cloud ecosystem also means that secrets and certificates (e.g. for databases) do not need to be stored or transmitted from external sources. Normally, we would have to manage and distribute credentials for a service account,

in order to get access to the application in Google Cloud. Because all app components are located in google cloud, this distribution is handled internally; there are no external systems that need the credentials. An exception from this is the IDaaS solution. Login to admin-client is tied to Firebase. An API key is placed in the source code, in accordance wit best practices for Firebase. The key serves as project identification, but is not used for authentication [50]. The API key may give users access to the authentication endpoint, and thereby also let attackers try to brute force a login. As per recommendations from Google, the traffic quota on said endpoint has been restricted, to prevent massive brute force attempts at authentication through the API.

The database is configured not to allow traffic from the web app directly , as described in section 5.4.1. Communication between end users in the web app and the database therefore must go through an API. Adding this extra layer of security has the cost of slightly lower performance, but with the large benefit of keeping the database safer.

10.3.1 Identity as a service

To provide Skogkurs with a secure login system with minimal administration, Identification as a Service (IDaaS) is utilized [51] . The main benefit of using IDaaS in this project is added security through utilizing a well known user management system instead of writing an own implementation. Password-less authentication is used, with login through one-time links as described in 7.5.3.

10.4 Web Editor

The web editor described in section 5.3 is to some extent trust based. An administrator can grant access to editor users. As of now, only one privilege level, or role, is implemented. That means that editor users can edit all calculators present in the database, in addition to creating new ones. The operations that require authentication are creation and updating of calculators. We expect that creating calculators is a functionality that is infrequently used, and by only a few employees of the organization. Therefore it is up to the organization to manage its users in a responsible manner.

As a calculator may be quite complex to implement, and may require many stages and work hours, the calculator should be seen as a valuable asset. Allowing any user to change any calculator may therefore be a data integrity risk. To mitigate this, more robust rules for writing/overwriting calculators should be implemented, as outlined in 11.6,

Privilege levels of each user should be at a minimum. One could for instance let users create, edit or delete their own calculators, but only read or copy the ones made by other users. Alternatively, all changes could be considered for approval by the administrator. Ad-

min functionality on that level would however mean that we would have to implement access differentiated by role. More roles are not yet implemented.

To ensure accountability, some logging should be activated. Monitoring who does what and when may provide helpful information in case malicious actions have occurred from within the organization.

The above mentioned mitigations are not implemented, but should be prioritized in an eventual further development process.

10.5 Web App

The web app is designed to be openly available online to anyone. Because of this, it is not recommended to create calculators containing sensitive or protected data. For instance, if the calculator relies upon some constant that should be kept secret, by manipulating inputs and recording the output, one could be able to reverse engineer to find the constant. Preventing this from happening would be difficult without user authentication/authorization. Therefore, it is advisable not to create calculators based on secret data, as confidentiality might be at risk. Keeping the calculator open for all users was a specific request from Skogkurs, based on their main purpose of being a provider of knowledge in the forestry business. Not having to manage a user database was also a key point for Skogkurs. Keeping the calculator open makes it more available to the intended users.

10.5.1 Input validation

Input fields that allow user input may be a gateway to inject malicious code into an app. To ensure complete control of the input information, restrictions on all inputs are implemented. We have used various methods of input validation, depending on the type of input throughout the application.

Some inputs are implemented as drop down menu. That way, only a predefined set of choices are available to the user. Number inputs are validated using regular expressions, restricting the format, so that only valid numbers and signs may be input. No text input is available to the web app users; only numbers and drop down menus. From a security point of view, we consider this to be a minor concern, the largest unwanted effect would be a calculation with NaN as result.

In the calculator editor, the editor user can add info texts to inputs and results using a HTML editor. The input is then sanitized, which involves removing all script tags, and various other tags not needed for presenting the information. Removing script tags is essential to prevent cross-site scripting (XSS). Allowing any html tag could leave the application open to download or run malicious files.

More importantly, the inputs are validated in the client as well. There is no trust relationship between the client and the API, and therefore input is always sanitized. This prevents potential attackers from injecting malicious content through a man in the middle attack.

10.5.2 Web App Storage

The user may store calculations in the browser's local store, which might get deleted by accident, as described in 7.2.4. A safer and more persistent way would be to save the URL locally, but it is up to the user to actually do that.

Another possible solution for saving could be to offer cloud storage. This would be a good solution, for numerous reasons. The stored data would be device independent. Users could access the data across multiple devices, allowing for example starting a project on the computer at the office, and continuing in the field on a portable device. Also, if a device is lost or damaged, retrieving the information from the cloud for later use would be an easy task. But as login for end users was not wanted, this solution would not be viable at this point.

These are examples of data integrity and availability issues that may occur because of the way the application is constructed. To at least partially reduce the problem, users are warned about how the storing mechanism works when calculator results are stored, and are requested to save important calculations manually.

10.6 Security evaluation

Securing the application has certainly been a challenge. A major issue of software security is that you never can know for certain that all necessary security countermeasures have been implemented. There are no strict limits, and it all depends on the application and the severity of a potential attack. In our case, the application was required to mainly be open to most users. An overall security strategy has been to minimize the attack surface. Also, the information stored is kept at a minimum. Authentication is only required to maintain calculators. Automated security scans are used extensively, to remove known vulnerabilities otherwise difficult to detect, among others by changing the build tool at a late stage of the project. User inputs are sanitized to prevent malicious code injection. User authentication is outsourced using IDaaS, which should provide a safe and reliable functionality. Deployment is kept inside the same PaaS ecosystem, keeping credentials internally in the same system.

Confidentiality is preserved through authentication and authorization to do database operations. Data integrity might still be at risk because of the storing mechanisms, both in the editor and the web app, because user errors may lead to loss of data. Availability is ensured by hosting the service in a dynamically scalable environment, set up with measures

to prevent DoS attacks.Despite our efforts at securing the application, it is possible that the security measures implemented are insufficient. As mentioned above, there are numerous measures that could be implemented in a later version, most notably in relation to storage.

Chapter 11

Discussion and Conclusion

This chapter reviews the project's development process, reflecting on subjects such as time management and collaboration with Skogkurs. It evaluates key outcomes, project impact, learning achievements, and future initiatives. Additionally, sustainability effects on the forestry industry and ethical considerations are explored. Insights gained from using AI is also briefly discussed, before summarizing the project's main conclusions.

11.1 Process

The following aims to reflect on and critique our work process throughout the project period, focusing on the experiences with Scrum, cooperation with Skogkurs, and distribution of development responsibilities within the group.

11.1.1 Time Management

As noted in 4.1, we have pursued a comprehensive usage of Scrum activities and processes, under the assumption that the effort to plan and administrate the process in a iterative manner would be conducive to maintaining progress throughout the project period. Broadly speaking, we have found this to be the case. While the temptation to reduce time spent in meetings in favor of developing is always present, the benefits of synchronization and open discussion afforded by the consistent utilization of a development model has been of great benefit for the project as a whole.

Figure 11.1 shows the hours spent, categorized over a two-month period. This specific timeframe was chosen as it best represents typical work weeks during the project's development phase, and because time tracking was less consistent in the later phase. Analyzing and critiquing how time was allocated proves challenging due to ambiguously defined categories and the frequent overlap of tasks, such as research and documentation, which are not mutually exclusive. In contrast, some activities were quantified more precisely: for instance,

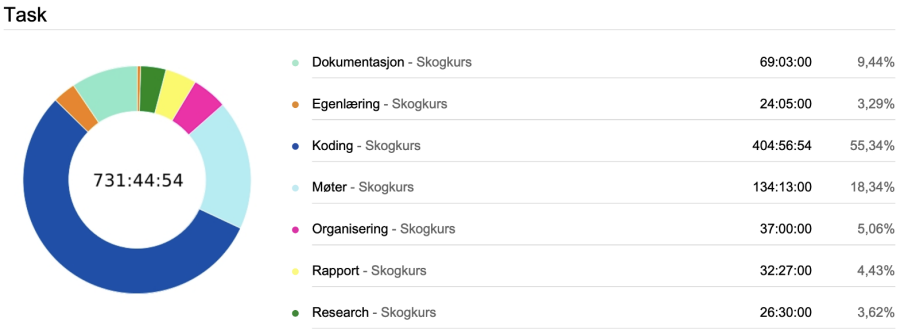


Figure 11.1: Hours categorized from February 1st to April 1st

meetings constituted approximately 20% of the total time spent, a figure that remained surprisingly stable despite a shift from one-week to two-week sprints. Additionally, active development, primarily coding, represented 50% of the project time during this period.

Looking back, it's challenging to determine if time could have been allocated more effectively and how that would have impacted long-term productivity. However, we believe that the distribution of time has been reasonable.

11.1.2 Co-operation with Skogkurs

The co-operation with Skogkurs (see 4.2 for details) has overall been a great benefit for the development process. By putting people and resources at our disposal, they have facilitated frequent feedback on the state of the software, suggestions for improvement, in addition to access to their specialists in the fields of forestry, interaction design and sustainability, well outside our domain.

Our collaboration with Skogkurs has significantly shaped our development process. The sprints were driven by incorporating their feedback, as well as ongoing evaluations by our developers. Having the application reviewed by experts with deep knowledge of the domain where it will be used has been invaluable. This level of insight would have been unattainable without Skogkurs's active involvement in the development process.

11.1.3 Scope creep

Bachelor's projects commissioned by external companies sit at the intersection of academic learning and practical application, each with their distinct goals. Balancing these areas has occasionally led to scope creep, potentially diverting focus from academic interests. We have been fortunate to collaborate with a highly dedicated company and their partners, all of whom are deeply committed to the success of the project. While the hospitality of our partners and the iterative feedback mechanisms emphasized by the Scrum framework have occasionally brought consultancy aspects into our work that conform to Skogkurs's interests, we

have made adjustments to ensure the project stays aligned with academic priorities. One of those priorities, which Skogkurs appeared less focused on, was to significantly halt feature development on the web application in favor of enhancing the technical capabilities of the editor and tree structures to catch up with the statically programmed version. This decision resulted in the omission of some planned features; however, the team ultimately agreed that it was essential to remain true to the original vision of developing a more maintainable calculation tool, which can be further enhanced in the future.

11.1.4 Presentation of the Web Editor

Considering how useful we found the feedback on the web app, presenting the web editor in order to receive more feedback from the product owners and stakeholders could have been done more comprehensively.

The primary focus of the weekly PO-meetings was receiving feedback on the web app. Consequently, less emphasis was placed on showcasing the web editor, reducing the amount of feedback we received. Considering how useful we found the feedback on the web app, this should have been amended. A possible solution to this might have been dedicating time in the PO meetings for both the web app and the web editor, so that both applications were presented on a weekly basis. However, given that development on the web editor began at a later date compared to the web editor, it might not have been mature enough to receive actionable feedback until a relatively late stage in the project period.

The web app was also more directly a representation of the functionality that Skogkurs wanted from the product, and therefore they might have more feedback to give than for the web editor. While the web editor was conceived as a means to meet Skogkurs requirements, particularly with regards to calculating the costs of alternate forms of logging, this was perhaps not communicated clearly enough. On the whole, less interest was generated around the web editor than the web app, again contributing to less, very valuable input being generated for this critical piece of the software.

11.2 Goal assessment

The product in its current state fulfils its purpose to a large degree. Functionality available in the original excel sheet is recreated in the web app, providing the users, with a more user friendly and available product. with some deficiencies handled separately in section 11.6.

The tables below list the goals presented in Introduction 1.2. Each goal goal is handled individually to assess to which degree the goals are met.

11.2.1 Outcome goals

The application will support:	**Assessment**
Creating new formulas with different cost drivers.	Through the web editor, new formulas can be created, updated and published, without the need of programming competence. Skogkurs' goal of adding "patch cut" operations can be realized through creation of a new formula in the web editor. The user interface for the web editor is functional, although improvements could be implemented for ease of use.
Saving and retrieving versioned formulas and cost calculations	Users may save performed calculations on their device for later retrieval, or share on email.
Usage on a variety of device types, like PCs, tablets, and smartphones	The application is responsive and well suited for use on different device types and sizes.
Usage on multiple operating systems, like Windows, iOS and Android	Making the application as a progressive web app allows a common app suitable for all regular devices (desktop, tablet, mobile), rather than having to develop platform specific versions.
Usage in the field, even without internet access	Web browsers will cache the web app for up to 365 days after visiting the site, without requiring any action from the user.

Table 11.1: Outcome goals assessment

The outcome goals are perhaps the most specific ones, and therefore can be assessed by comparing product features to the goals set. Working towards these goals were the basis of the product development process, and we can largely say that these goals have been met.

11.2.2 Impact goals

The application will:	Assessment
Ensure that all users at any given time have access to the latest version of any particular formula	The nature of progressive web apps, thanks to the service worker, ensures that users are prompted to load the latest app version, thereby assuring Skogkurs that the users use the same version.
Work as a tool to plan logging operations, as used by various actors in the forestry industry, like machine operators, forest owners and forestry operations managers.	The application can be used by its intended users to calculate cost related to forestry operations. Skogkurs and the external stakeholders are pleased with the resulting app. Time will tell if the industry adopts the application. Some Skogkurs employees have expressed worry that the application may render the industry's economy too transparent, while others welcome this as an advantage to ensure more correct valuation.
Serve as a tool in Skogkurs' teaching courses, like "Bedriftsøkonomi for entreprenører"	Skogkurs intends to use the application in its current state as part of a course in the near future.

Table 11.2: Impact goals assessment

There is uncertainty about how many have used the original excel sheet the application replaces, but there is hope that a web based calculator is perceived as more available, thereby attracting more users. The potential impact on the industry is uncertain, but it may well be large. Some Skogkurs employees have expressed worry that the application may render the industry's economy too transparent, while others welcome this as an advantage to ensure more correct valuation.

11.2.3 Learning goals

Expected learning outcome:	**Assessment**
Gain experience with processing and analyzing a real product's owner problem, formulate a solution for that problem area and finally implement that solution	Through close cooperation with Skogkurs, the project group has gained experience in working with a non-technical organization. Because of this, we have had to handle all the technical aspects of the application ourselves. The original requirements were quite open, and we have tried to both satisfy Skogkurs' wishes, while also creating an interesting project for ourselves by expanding on the original idea.
Expand and develop our experience in the field of web development	As the project evolved, we have discovered new libraries and tools, and further explored concepts already familiar. We have gained valuable experience by managing a large project from idea to finished application.
Integrate and expand upon experiences and skills acquired throughout the three year study program	We have utilized knowledge from various earlier courses throughout the project, while also learning new technologies as a supplement.

Table 11.3: Learning goals assessment

The learning goals are perhaps the most difficult to assess, as they may be too broad or general. It is also challenging to pinpoint what has been learned, but we feel more confident now with the tools, methods, frameworks, and libraries used throughout the project.

11.3 Sustainability

This section will touch upon the sustainability of the application and its development process, and how the product impacts the forestry industry in terms of sustainability.

The perspective used to evaluate sustainability for both aspects is grounded in the UN Sustainable Development Goals[52]. The concept of the Triple Bottom Line[53] will also inform the conceptualization of sustainability for how the application will impact the forestry industry.

11.3.1 Software

Sustainability has been an implicit goal of the requirement specification and development process for the project since its inception. By developing the application, we hope to pursue United Nation sustainability goal 12: Responsible Production and Consumption.

Skogkurs already maintains a number of different cost calculators, each of them necessitating a separate development process, and possibly even a dedicated project for the organization. The hope for the web editor then, is that it can be used as a tool for generating these kinds of calculators in a much more efficient manner, without requiring software developers or a larger process wherein a whole new application is designed and implemented.

In addition to sustainability advantages of the application itself, we have tried to utilize sustainable software development techniques. Primarily this has taken the form of utilizing established libraries and modules to provide features and functionalities which are either common-place in the field of web development, or well suited to our particular needs. For instance, by using the component library React-Bootstrap we were able to focus the development process on providing the product owners with the required functionality, utilizing pre-made UI elements that would otherwise have required investment of significant time and resources to implement. Similarly, adopting Rete.js allowed us to focus on creating a functional editor by utilizing their framework, without undertaking a potentially lengthy design and development process to achieve even the underlying infrastructure necessitated for the application.

11.3.2 Forestry

The application's impact on the forestry industry will, as previously mentioned, be evaluated within the framework of the Triple Bottom Line, which emphasises the value of measuring a business' social, environmental and economic impacts[53]. Combining this framework with the United Nation's Sustainable Development Goals, the application's use in the industry may pursue the goals:

- Goal 8: Decent Work and Economic Growth.
- Goal 12: Responsible Production and Consumption.
- Goal 13: Climate Action
- Goal 15: Life On Land

Social

Speaking with representatives from Skogkurs[54], an issue with the Norwegian forestry industry is that employment as harvester or forwarder operator is sometimes considered an unsustainable line of work. During a logging operation, the machines are run around

the clock, to maximize the amount of timber that is harvested in a given period of time. Maintaining this strenuous pace over the course of a career is considered incompatible with starting a family, for instance. Consequently,members of the profession often leave it for other jobs that more easily accommodate such concerns.

By providing tools for calculating the cost of an operation, and documenting those calculations, the representatives from Skogkurs have suggested that the application might aid these operators, and the other actors in a logging operation, in negotiating contracts that provide decent value to all parties. This would comply with Goal 8: Decent Work and Economic Growth

Environmental

While the application does not directly calculate the environmental impact of a logging operations, such as carbon emissions or loss of bio-diversity, the stated desire to add functionality for calculating the cost and productivity of alternate logging forms ("lukket hogst"[55] or selection cutting, see appendix D), might constitute an incentive to further the adoption of these forms of logging in the forestry industry.

Selection cutting has a number of environmental benefits, most notably bio-diversity[56], and the inclusion of formulae for those forms of logging might, albeit indirectly, pursue meeting Goals 13: Climate Action and 15: Life on Land.

Economic

Representatives from Skogkurs have presented the value of furthering the professionalization of the forestry industry in Norway [54] . While this can be broadly constructed, for our purposes we will focus on the planning of logging operations. According to the representatives from Skogkurs, today, the planning of a logging operation is often carried out in an informal manner, where more weight is placed on the lived experiences and personal connections of the participating actors, than the real conditions of the forested area that is being considered for logging. This planning, potentially ignoring the real conditions of the stock and the terrain of the logging operations, may mean that considerably less effective logging is carried out, measured in volume of timber produced per hour and cost per volume of timber produced.

The hope for the application, then, is its function as a planning and evaluation tools will allow actors in the forestry to make more informed decisions about which logging operations to undertake. Given that the results from the calculations provide a relatively accurate estimation of the actual cost and productivity of an operation, this might increase the efficiency of logging operations, thereby complying with Goal 12: Responsible Production and Consumption.

11.4 Ethical Concerns

The sustainability goals of the application are reliant on the calculator, and the formula it is based on, producing realistic results. As outlined in 8.3, the web-based calculator does accurately recreate the original Excel spreadsheet calculator. However, the accuracy of either of these implementations in a real-life situation has not been tested. Consequently, we cannot say with any degree of certainty that the cost and productivity of a logging operation, as calculated in the web app, would correspond with the actual cost and productivity of that logging operation once had it been carried out.

If the application is adopted in the manner outlined in 11.3.2, and used as tool to negotiate the payment and working conditions of the actors in a logging operation, an inaccurate calculation could result in those negotiations being based on incorrect grounds. Potentially, this could result in one or more parties receiving inadequate compensation, or other unfair conditions.

On the basis of these concerns, it seems prudent to verify the accuracy of the calculation results in the web app, before it is widely adopted in the forestry industry.

11.5 Practical Insights from Use of AI

Although the project did not directly integrate AI models into the product, a variety of AI tools, primarily GPT-based Large Language Models (LLMs), were employed for support. The team did not set specific guidelines on how extensively to use these tools, resulting in varied usage among team members. Below are the most common uses throughout the project:

- **Conceptual learning:** Modern web development, especially full-stack development, is built on layers upon layers of frameworks, libraries, and tools that continuously change in popularity. Some of these have very little or difficult-to-understand documentation, while others have so much information that it's tough to get started. Using LLMs for understanding trade offs between various technology solutions have been helpful to gaining insights into the available options, even if it's not always perfectly accurate, and sometimes give outdated information.
- **Suggestive code completion:** During this project, many team members experimented with GitHub Copilot for the first time. This tool excelled at completing boilerplate code and providing contextualized code completion that surpass the capabilities of traditional IDEs. However, these suggestions had a tendency to disrupt the programmer's thought process, often resembling background noise. We found that, while helpful, the tool's suggestions generally had a negative impact on code quality, as its recommendations are limited to previously seen code. Consequently, several team members chose to disable the tool after determining that its drawbacks outweighed its benefits.

- **Generate code comments:** Writing comments, particularly with *JSDoc tags*[1], is time-consuming. While GitHub Copilot helped streamline the process, the automatically generated comments were often superficial, providing little additional insight compared to a quick scan of the code. Substantial edits or full rewrites were frequently necessary. However, Copilot proved valuable in efficiently handling the layout and detailing of parameter names, which are typically the most tedious parts of writing *function comments*.
- **Image generation:** Although barely utilized, a generic illustration was created by Microsoft Designer for display on the API informational page[2]. This visual was intended to provide a stimulating contrast to all the technical text.
- **Copy-editing:** Crafting concise reports that use precise wording and correct sentence and grammatical structures is challenging but essential for ensuring readability. LLMs serve as effective copy-editors[3], however it's crucial that the original meaning is maintained, especially in respect to quotes and citations. Our experience with LLMs as copy-editors is that the adjusted text often becomes bland and generic, but this is usually mitigated by providing ample context. Despite the time-consuming and iterative nature of refining text with LLMs, the enhanced quality and smoother readability often justify the effort.

The consensus within the team is that, in its current state, AI tools are too limited to significantly enhance productivity. While effective for simpler tasks, LLMs tend to falter when dealing with more specialized and narrow problem domains. Consequently, the confidence level for information provided by LLMs is fairly low, despite the answer often appearing convincing. However, they can be valuable in areas where technical resources are difficult to find, or where alternatives like *Stack Overflow*[4] might be consulted for quick inquiries. Traditional search engines and documentation still continue to be the most reliable, and preferable resources for exploring new topics among the group.

11.6 Future Work

Over the course of the project, the product has evolved into a app that we are genuinely proud of. The original project idea has been implemented and expanded with the addition of the editor, which greatly improves flexibility for the user. Using an agile approach, new

[1] JSDoc tags are annotations in JavaScript comments that describe code metadata like parameters and return values.

[2] `https://kostnadskalkulator.skogkurs.no/apiinfo`

[3] The following definition was used to describe copy editing: "Copy editing means improving the text to be coherent while maintaining the same meaning and style. A copy editor (in charge of editing) needs to check the grammar, spelling, tone, and style and to give an overall easy flow to the entire text." [57]

[4] Stack Overflow is an online platform for developers to ask and answer programming questions.

requirements and ideas emerged along the way. To create the best possible product for the product owner, while also focusing on the academic side of the project, some features got higher priority than others. We believe we have developed a great product, which could be improved even further if more time was available. A document detailing proposed changes and additions is available in appendix N. A selection of specific improvements include:

- Web editor usability: The web editor is functional but the user could benefit from more guidance and feedback.
- Editor performance issues: Large graphs loaded into the editor can cause performance issues. This should be investigated and improved.
- Versioning and storage: Unintentionally overwriting an existing calculator is possible. A simple mitigation for this could to check the version and alert the user. Otherwise, a more robust versioning system, or alternatively, a role based access control system and calculator ownership could be implemented.
- Web app availability: The web app only uses Norwegian. Internationalisation by offering the user to select language could make the application available to more users. Further work also needs to be done to fully facilitate the use of screen readers.

11.7 Conclusion

The aim of this project was to extend the lifetime of the existing productivity calculator, by migrating and extending the functionality of the Excel spreadsheet to a web-based solution. Skogkurs, the company that commissioned the project, provided substantial support throughout its duration. The growing number of stakeholders also indicate that they saw value in participating and offering feedback to improve the product.

Given considerable flexibility in execution, our group pursued a more complex solution than Skogkurs might have initially anticipated. To mitigate risks, the group backported much functionality to the prototype, such that it could serve as a fallback, in case the vision could not be realized in time, but was ultimately finished.

In retrospect, the project achieved all its initial objectives, except for finalizing support for patch-cut operations (selective logging). However, the introduction of the visual editor has facilitated straightforward implementation through visual programming post-project, which Skogkurs plans to adopt. Overall, we are genuinely pleased with the project's outcome and the extensive work experience gained through close collaboration with Skogkurs.

While we were unable to test the editor with users to the degree we would have liked, hands on experience with the tooling indicated that while large math formulas in graph form could be tedious to work with, the benefit of ease of editing outweighed the drawbacks.

Bibliography

[1] M. W. Docs, *Progressive web apps*, Accessed: 2024-05-03, 2024. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps.

[2] M. W. Docs, *Progressive enhancement*, Accessed: 2024-05-03, 2024. [Online]. Available: https://developer.mozilla.org/en-US/docs/Glossary/Progressive_Enhancement.

[3] F. B. Alex Russel, *Progressive web apps: Escaping tabs without losing our soul*, Accessed: 2024-05-03, 2015. [Online]. Available: https://infrequently.org/2015/06/progressive-apps-escaping-tabs-without-losing-our-soul/.

[4] J. Wargo, *Learning Progressive Web Apps*. Addison-Wesley Professional, 2020.

[5] C. Love, *Progressive Web Application Development by Example*. Birmingham, UK: Packt Publishing, 2018.

[6] e. a. François Beaufort Pete LePage, *Add a web app manifest*, Accessed: 2024-05-03, 2022. [Online]. Available: https://web.dev/articles/add-manifest.

[7] H. Archive, *Web almanac 2022: Service worker usage*, Accessed: 2024-05-02, 2022. [Online]. Available: https://almanac.httparchive.org/en/2022/pwa#service-worker-usage.

[8] M. W. Docs, *Main thread*, Accessed: 2024-05-03, 2023. [Online]. Available: https://developer.mozilla.org/en-US/docs/Glossary/Main_thread.

[9] D. R. Andrew Guan, *Workers overview*, Accessed: 2024-05-05, 2020. [Online]. Available: https://web.dev/articles/workers-overview.

[10] C. for Developers, *Service worker lifecycle*, Accessed: 2024-05-05, 2021. [Online]. Available: https://developer.chrome.com/docs/workbox/service-worker-lifecycle.

[11] C. for Developers, *Strategies for service worker caching*, Accessed: 2024-05-05, 2021. [Online]. Available: https://developer.chrome.com/docs/workbox/caching-strategies-overview.

[12] J. Ingeno, *Software Architect's Handbook*. Packt Publishing, 2018.

[13] K. Schwaber and J. Sutherland. 'The 2020 scrum guide.' (2020), [Online]. Available: https://scrumguides.org/scrum-guide.html. Accessed: 2024-03-03.

[14] Atlassian. 'Kanban.' (2024), [Online]. Available: https://www.atlassian.com/agile/kanban#:~:text=In%20Japanese%2C%20kanban%20literally%20translates,in%20a%20highly%20visual%20manner.. Accessed: 2024-05-13.

[15] wrike. 'What are scrum story points.' (2024), [Online]. Available: https://www.wrike.com/scrum-guide/faq/what-are-scrum-story-points/. Accessed: 2024-05-11.

[16] S. Overflow, *Stack overflow developer survey 2023 | web frameworks and technologies*, Accessed: 2024-05-12, 2024. [Online]. Available: https://survey.stackoverflow.co/2023/#section-admired-and-desired-web-frameworks-and-technologies.

[17] W. Tech, *Usage statistics and market share of bootstrap for websites*, Accessed: 2024-05-12, 2024. [Online]. Available: https://w3techs.com/technologies/details/cs-bootstrap.

[18] React, *Managing state | passing data deeply with context*, Accessed: 2024-05-18. [Online]. Available: https://react.dev/learn/passing-data-deeply-with-context.

[19] Redux, *Why redux toolkit is how to use redux today*, Accessed: 2024-05-18. [Online]. Available: https://redux.js.org/introduction/why-rtk-is-redux-today.

[20] Kinsta, *What is express.js? everything you should know*, Accessed: 2024-05-18, 2023. [Online]. Available: https://kinsta.com/knowledgebase/what-is-express-js/.

[21] GoogleChrome, *Workbox: Javascript libraries for progressive web apps*, https://github.com/GoogleChrome/workbox, Accessed: 2024-05-02, 2024.

[22] e. a. Shubhie Panicker Addy Osmani, *Introducing aurora | chrome developers*, Accessed: 2024-05-03, 2021. [Online]. Available: https://developer.chrome.com/docs/aurora/overview.

[23] GoogleChromeLabs, *Sw-toolbox: A collection of service worker tools for offline runtime requests*, https://github.com/GoogleChromeLabs/sw-toolbox, Accessed: 2024-05-02, 2024.

[24] H. Archive, *Web almanac 2022: Workbox usage*, Accessed: 2024-05-02, 2022. [Online]. Available: https://almanac.httparchive.org/en/2022/pwa#workbox-usage.

[25] Google, *Workbox documentation: Handling service worker updates*, Accessed: 2024-05-03, 2021. [Online]. Available: https://developer.chrome.com/docs/workbox/handling-service-worker-updates.

[26] V. Stoliarov, *Rete.js documentation: Plugin system*, Accessed: 2024-05-18. [Online]. Available: https://retejs.org/docs/concepts/plugin-system.

[27] R. C. Martin, *Clean Architecture: A Craftman's Guide to Software Structure and Design*, 1st ed. Prentice Hall, 2018.

[28] R. C. Marthin, *The clean code blog: The single responsibility principle*. [Online]. Available: https://blog.cleancoder.com/uncle-bob/2014/05/08/SingleReponsibilityPrinciple.html.

[29] E. Gamma, R. Helm, R. Johnson and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, 1st ed. Addison-Wesley, 1994.

[30] M. W. Docs, *Closures - javascript*, Accessed: 2024-05-16. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Closures.

[31] M. W. Docs, *Promise*, Accessed: 2024-05-16. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise.

[32] A. Tukalo, *Promise and other tools for monadic error handling*, Accessed: 2024-05-16. [Online]. Available: https://dev.to/airtucha/promise-and-other-tools-for-monadic-error-handling-480h.

[33] V. Khorikov, *Unit Testing: Principles, Practices and Patterns*, 1st ed. Manning, 2020.

[34] K. C. Kathy Baxter Catherine Courage, *Understanding Your Users*, 2nd ed. Morgan Kaufmann, 2015, p. 433.

[35] Atlassian, *Monorepos in git*, Accessed: 2024-05-06. [Online]. Available: https://www.atlassian.com/git/tutorials/monorepos.

[36] e. a. Ferdinando Santacroce Aske Olsson, *Git: Mastering Version Control*. Packt Publishing, 2016.

[37] Fortinet. 'Cia triad.' (2024), [Online]. Available: https://www.fortinet.com/resources/cyberglossary/cia-triad. Accessed: 2024-05-19.

[38] M. Paul, *Official (ISC)2 Guide to the CSSLP*, 2nd ed. CRC Press, 2014, pp. 180–191.

[39] npm Docs, *Npm-audit*, Accessed: 2024-04-24, 2024. [Online]. Available: https://docs.npmjs.com/cli/v10/commands/npm-audit.

[40] L. L. Hewawasam, *How you should treat npm audit results*, Accessed: 2024-04-24, 2023. [Online]. Available: https://www.syncfusion.com/blogs/post/how-to-treat-npm-audit-results.

[41] C. R. App, *Getting started*, Accessed: 2024-04-15, 2021. [Online]. Available: https://create-react-app.dev/docs/getting-started.

[42] React, *Create a new react app*, Accessed: 2024-05-13. [Online]. Available: https://legacy.reactjs.org/docs/create-a-new-react-app.html#create-react-app.

[43] R. Kamra, *Inside create-react-app*, Accessed: 2024-04-15, 2022. [Online]. Available: https://medium.datadriveninvestor.com/inside-create-react-app-8f37ed9372f8.

[44] D. Abramov, *Help, npm audit says i have a vulnerability in react-scripts!* Accessed: 2024-04-15, 2021. [Online]. Available: https://github.com/facebook/create-react-app/issues/11174.

[45] Vite, *Why vite*, Accessed: 2024-04-30. [Online]. Available: https://vitejs.dev/guide/why.html.

[46] GitHub, *About code scanning*, Accessed: 2024-04-23, 2024. [Online]. Available: https://docs.github.com/en/code-security/code-scanning/introduction-to-code-scanning/about-code-scanning#about-code-scanning.

[47] GitHub, *Detection of insecure dependencies*, Accessed: 2024-04-23, 2024. [Online]. Available: https://docs.github.com/en/code-security/dependabot/dependabot-alerts/about-dependabot-alerts#detection-of-insecure-dependencies.

[48] GitHub, *About secret scanning*, Accessed: 2024-04-23, 2024. [Online]. Available: https://docs.github.com/en/code-security/secret-scanning/about-secret-scanning#about-secret-scanning.

[49] express-rate-limit, *Overview*, Accessed: 2024-05-08. [Online]. Available: https://express-rate-limit.mintlify.app/overview.

[50] G. Firebase, *Api keys for firebase are different from typical api keys*, Accessed: 2024-04-23, 2024. [Online]. Available: https://firebase.google.com/docs/projects/api-keys#api-keys-for-firebase-are-different.

[51] G. D. Maayan, *What is identity as a service (idaas)?* Accessed: 2024-04-30, 2023. [Online]. Available: https://www.computer.org/publications/tech-news/trends/identity-as-a-service.

[52] United Nations. 'The 17 goals.' (2024), [Online]. Available: https://sdgs.un.org/goals. Accessed: 2024-05-13.

[53] K. Miller. 'The triple bottom line: What it is and why it's important.' (2020), [Online]. Available: https://online.hbs.edu/blog/post/what-is-the-triple-bottom-line. Accessed: 2024-05-16.

[54] C. Glomsås, private communication, Apr. 2024.

[55] B. Lauten and L. Haugsrud. 'Lukket hogst.' (2024), [Online]. Available: https://skogkurs.no/artikkel/lukket-hogst/. Accessed: 2024-05-20.

[56] C. S. Mathis Lunde Maximilian M. Zimmermann and H. A. K. Sørlie, 'Lukkede hogstformer,' Skogkurs, Tech. Rep., 2022.

[57] Publishdrive, *What is copy editing*, Accessed: 2024-05-12. [Online]. Available: https://publishdrive.com/glossary-what-is-copy-editing.html.

Appendix A

Project Agreement

NTNU
Norges teknisk-naturvitenskapelige universitet

Fastsatt av prorektor for utdanning 10.12.2020

STANDARDAVTALE

om utføring av studentoppgave i samarbeid med ekstern virksomhet

Avtalen er ufravikelig for studentoppgaver (heretter oppgave) ved NTNU som utføres i samarbeid med ekstern virksomhet.

Forklaring av begrep

Opphavsrett
Er den rett som den som skaper et åndsverk har til å fremstille eksemplar av åndsverket og gjøre det tilgjengelig for allmennheten. Et åndsverk kan være et litterært, vitenskapelig eller kunstnerisk verk. En studentoppgave vil være et åndsverk.

Eiendomsrett til resultater
Betyr at den som eier resultatene bestemmer over disse. Utgangspunktet er at studenten eier resultatene fra sitt studentarbeid. Studenten kan også overføre eiendomsretten til den eksterne virksomheten.

Bruksrett til resultater
Den som eier resultatene kan gi andre en rett til å bruke resultatene, f.eks. at studenten gir NTNU og den eksterne virksomheten rett til å bruke resultatene fra studentoppgaven i deres virksomhet.

Prosjektbakgrunn
Det partene i avtalen har med seg inn i prosjektet, dvs. som vedkommende eier eller har rettigheter til fra før og som brukes i det videre arbeidet med studentoppgaven. Dette kan også være materiale som tredjepersoner (som ikke er part i avtalen) har rettigheter til.

Utsatt offentliggjøring
Betyr at oppgaven ikke blir tilgjengelig for allmennheten før etter en viss tid, f.eks. før etter tre år. Da vil det kun være veileder ved NTNU, sensorene og den eksterne virksomheten som har tilgang til studentarbeidet de tre første årene etter at studentarbeidet er innlevert.

1. Avtaleparter

Norges teknisk-naturvitenskapelige universitet (NTNU) Institutt:	Institutt for datateknologi og informatikk (IDI)
Veileder ved NTNU: e-post og tlf.	Peter Stefan Nussbaum peter.nussbaum@ntnu.no, +47 95 87 28 15
Ekstern virksomhet: Ekstern virksomhet sin kontaktperson, e-post og tlf.:	Skogkurs Mikael Fønhus, mf@skogkurs.no, +47 99 28 70 02
Student: Fødselsdato:	Øystein Qvigstad, +47 90 29 16 22 27.05.1989
Ev. flere studenter[1]	André Marthinsen, Paul Schiager, Even Stetrud

Partene har ansvar for å klarere eventuelle immaterielle rettigheter som studenten, NTNU, den eksterne eller tredjeperson (som ikke er part i avtalen) har til prosjektbakgrunn før bruk i forbindelse med utførelse av oppgaven. Eierskap til prosjektbakgrunn skal fremgå av eget vedlegg til avtalen der dette kan ha betydning for utførelse av oppgaven.

2. Utførelse av oppgave

Studenten skal utføre: (sett kryss)

Masteroppgave	
Bacheloroppgave	x
Prosjektoppgave	
Annen oppgave	

Startdato:	05.01.2024
Sluttdato:	20.05.2024

Oppgavens arbeidstittel er: Nettbasert kostnadskalkulator for skogsbedrifter

[1] Dersom flere studenter skriver oppgave i fellesskap, kan alle føres opp her. Rettigheter ligger da i fellesskap mellom studentene. Dersom ekstern virksomhet i stedet ønsker at det skal inngås egen avtale med hver enkelt student, gjøres dette.

Ansvarlig veileder ved NTNU har det overordnede faglige ansvaret for utforming og godkjenning av prosjektbeskrivelse og studentens læring.

3. Ekstern virksomhet sine plikter

Ekstern virksomhet skal stille med en kontaktperson som har nødvendig faglig kompetanse til å gi studenten tilstrekkelig veiledning i samarbeid med veileder ved NTNU. Ekstern kontaktperson fremgår i punkt 1.

Formålet med oppgaven er studentarbeid. Oppgaven utføres som ledd i studiet. Studenten skal ikke motta lønn eller lignende godtgjørelse fra den eksterne for studentarbeidet. Utgifter knyttet til gjennomføring av oppgaven skal dekkes av den eksterne. Aktuelle utgifter kan for eksempel være reiser, materialer for bygging av prototyp, innkjøp av prøver, tester på lab, kjemikalier. Studenten skal klarere dekning av utgifter med ekstern virksomhet på forhånd.

Ekstern virksomhet skal dekke følgende utgifter til utførelse av oppgaven:

Dekning av utgifter til annet enn det som er oppført her avgjøres av den eksterne underveis i arbeidet.

4. Studentens rettigheter

Studenten har opphavsrett til oppgaven[2]. Alle resultater av oppgaven, skapt av studenten alene gjennom arbeidet med oppgaven, eies av studenten med de begrensninger som følger av punkt 5, 6 og 7 nedenfor. Eiendomsretten til resultatene overføres til ekstern virksomhet hvis punkt 5 b er avkrysset eller for tilfelle som i punkt 6 (overføring ved patenterbare oppfinnelser).

I henhold til lov om opphavsrett til åndsverk beholder alltid studenten de ideelle rettigheter til eget åndsverk, dvs. retten til navngivelse og vern mot krenkende bruk.

Studenten har rett til å inngå egen avtale med NTNU om publisering av sin oppgave i NTNUs institusjonelle arkiv på Internett (NTNU Open). Studenten har også rett til å publisere oppgaven eller deler av den i andre sammenhenger dersom det ikke i denne avtalen er avtalt begrensninger i adgangen til å publisere, jf. punkt 8.

5. Den eksterne virksomheten sine rettigheter

Der oppgaven bygger på, eller videreutvikler materiale og/eller metoder (prosjektbakgrunn) som eies av den eksterne, eies prosjektbakgrunnen fortsatt av den eksterne. Hvis studenten

[2] Jf. Lov om opphavsrett til åndsverk mv. av 15.06.2018 § 1

skal utnytte resultater som inkluderer den eksterne sin prosjektbakgrunn, forutsetter dette at det er inngått egen avtale om dette mellom studenten og den eksterne virksomheten.

Alternativ a) (sett kryss) Hovedregel

x*	Ekstern virksomhet skal ha bruksrett til resultatene av oppgaven

Dette innebærer at ekstern virksomhet skal ha rett til å benytte resultatene av oppgaven i egen virksomhet. Retten er ikke-eksklusiv.

* Bruksretten omfatter også fullmakt til å videreutvikle, oppdatere eller på annen måte foreta endringer i kildekoden og alle andre komponenter som utgjør leveransen, inkludert men ikke begrenset til dokumentasjon, skript osv.

Alternativ b) (sett kryss) Unntak

	Ekstern virksomhet skal ha eiendomsretten til resultatene av oppgaven og studentens bidrag i ekstern virksomhet sitt prosjekt

Begrunnelse for at ekstern virksomhet har behov for å få overført eiendomsrett til resultatene:

6. Godtgjøring ved patenterbare oppfinnelser

Dersom studenten i forbindelse med utførelsen av oppgaven har nådd frem til en patenterbar oppfinnelse, enten alene eller sammen med andre, kan den eksterne kreve retten til oppfinnelsen overført til seg. Dette forutsetter at utnyttelsen av oppfinnelsen faller inn under den eksterne sitt virksomhetsområde. I så fall har studenten krav på rimelig godtgjøring. Godtgjøringen skal fastsettes i samsvar med arbeidstakeroppfinnelsesloven § 7. Fristbestemmelsene i § 7 gis tilsvarende anvendelse.

7. NTNU sine rettigheter

De innleverte filer av oppgaven med vedlegg, som er nødvendig for sensur og arkivering ved NTNU, tilhører NTNU. NTNU får en vederlagsfri bruksrett til resultatene av oppgaven, inkludert vedlegg til denne, og kan benytte dette til undervisnings- og forskningsformål med de eventuelle begrensninger som fremgår i punkt 8.

8. Utsatt offentliggjøring

Hovedregelen er at studentoppgaver skal være offentlige.

Sett kryss

x	Oppgaven skal være offentlig

I særlige tilfeller kan partene bli enige om at hele eller deler av oppgaven skal være undergitt utsatt offentliggjøring i maksimalt tre år. Hvis oppgaven unntas fra offentliggjøring, vil den kun være tilgjengelig for student, ekstern virksomhet og veileder i denne perioden. Sensurkomiteen vil ha tilgang til oppgaven i forbindelse med sensur. Student, veileder og sensorer har taushetsplikt om innhold som er unntatt offentliggjøring.

Oppgaven skal være underlagt utsatt offentliggjøring i (sett kryss hvis dette er aktuelt):

Sett kryss		Sett dato
	ett år	
	to år	
	tre år	

Behovet for utsatt offentliggjøring er begrunnet ut fra følgende:

Dersom partene, etter at oppgaven er ferdig, blir enig om at det ikke er behov for utsatt offentliggjøring, kan dette endres. I så fall skal dette avtales skriftlig.

Vedlegg til oppgaven kan unntas ut over tre år etter forespørsel fra ekstern virksomhet. NTNU (ved instituttet) og student skal godta dette hvis den eksterne har saklig grunn for å be om at et eller flere vedlegg unntas. Ekstern virksomhet må sende forespørsel før oppgaven leveres.

De delene av oppgaven som ikke er undergitt utsatt offentliggjøring, kan publiseres i NTNUs institusjonelle arkiv, jf. punkt 4, siste avsnitt. Selv om oppgaven er undergitt utsatt offentliggjøring, skal ekstern virksomhet legge til rette for at studenten kan benytte hele eller deler av oppgaven i forbindelse med jobbsøknader samt videreføring i et master- eller doktorgradsarbeid.

9. Generelt

Denne avtalen skal ha gyldighet foran andre avtaler som er eller blir opprettet mellom to av partene som er nevnt ovenfor. Dersom student og ekstern virksomhet skal inngå avtale om konfidensialitet om det som studenten får kjennskap til i eller gjennom den eksterne virksomheten, kan NTNUs standardmal for konfidensialitetsavtale benyttes.

Den eksterne sin egen konfidensialitetsavtale, eventuell konfidensialitetsavtale den eksterne har inngått i samarbeidprosjekter, kan også brukes forutsatt at den ikke inneholder punkter i motstrid med denne avtalen (om rettigheter, offentliggjøring mm). Dersom det likevel viser seg at det er motstrid, skal NTNUs standardavtale om utføring av studentoppgave gå foran. Eventuell avtale om konfidensialitet skal vedlegges denne avtalen.

Eventuell uenighet som følge av denne avtalen skal søkes løst ved forhandlinger. Hvis dette ikke fører frem, er partene enige om at tvisten avgjøres ved voldgift i henhold til norsk lov. Tvisten avgjøres av sorenskriveren ved Sør-Trøndelag tingrett eller den han/hun oppnevner.

Denne avtale er signert i fire eksemplarer hvor partene skal ha hvert sitt eksemplar. Avtalen er gyldig når den er underskrevet av NTNU v/instituttleder.

Signaturer:

Instituttleder: Dato: 25/1-24
Veileder ved NTNU: Dato: 24.01.2024
Ekstern virksomhet: Stenvar Oppdrag Dato: 21/1-24
Student: Øyvind Aagaard Dato: 19/1-24
Ev. flere studenter Paal Solvang 19/1-24 Andreas Mathisen 19/1-24 Even Stetind 24/1-24

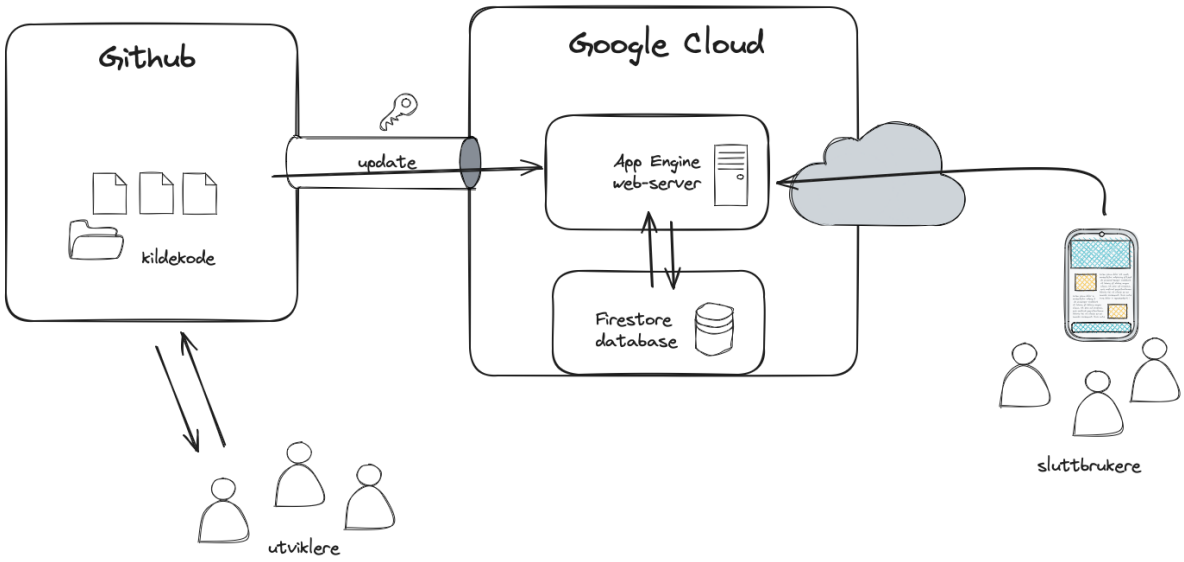
This document outlines the agreement between the bachelor group and the company that commissioned the project, covering key aspects including ownership, rights of use, and publication rights.

Appendix B

System Operations Manual

This document provides an overview of how backend components, service accounts, and deployment processes integrate from an administrative perspective. It is part of the product handover to Skogkurs. While the document does not contain sensitive information, some details have been redacted to err on the side of caution.

Driftsdokument av nettbasert Skogskalkulator



1. Google Cloud

1.1 Overordnet

Vi benytter i dette prosjektet Google Cloud (https://cloud.google.com) som tjenesteleverandør under Skogkurs eierskap. Primær administrasjonskonto er video@skogkurs.no.

Project ID (Google Cloud): REDACTED

Project number (Google Cloud): REDACTED

De viktigste produktene som anvendes i Google Cloud er:

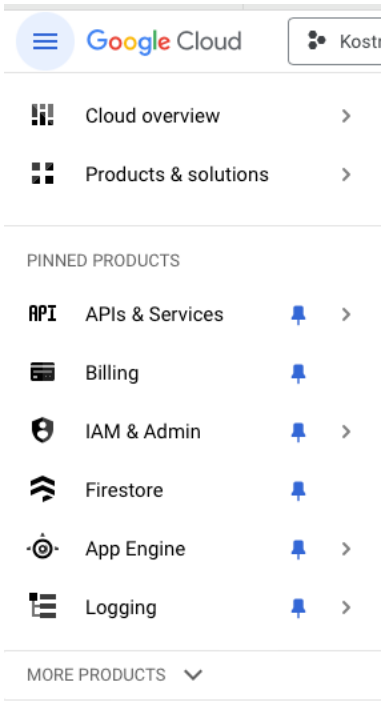
- **App Engine**: Kontainerløsning for bygging, kjøring og skalering av tjenesten. Denne tar utgangspunktet i en konfigurasjonsfil (App.yaml) og kildekoden, og bygger da opp tjenesten i driftsammenheng.
- **Firestore**: Dokumentbasert (NoSQL) database som lagrer alle kalkulatorversjonene som opprettes i kostnadskalkulatorens administrasjonsverktøy, og lastes inn i kostnadskalkulatoren on-demand.
- **Firebase Auth**: Sørger for adgangskontroll og utsendelse av midlertidig passord for innlogging i kostnadskalkulatorens administrasjonsvertøy.

Av ekstra interesse er det også en tilkobling til følgende tjenester produkter:

- **GitHub Actions**: Ved oppdatering av kildekode i «main»-grenen, kjøres det et automatisk script som oppretter ny kontainer i App Engine og sørger for at trafikk rutes mot denne.
- **Domeneshop**: Et subdomene rutes til tjenesten. Google Cloud håndterer automatisk fornyelse av TLS-sertifikat, men dette innebærer at subdomenet har verifisert eierskap, gjennom ulike nøkler/verifiseringsstrenger.

1.2 Grunnleggende oppsett

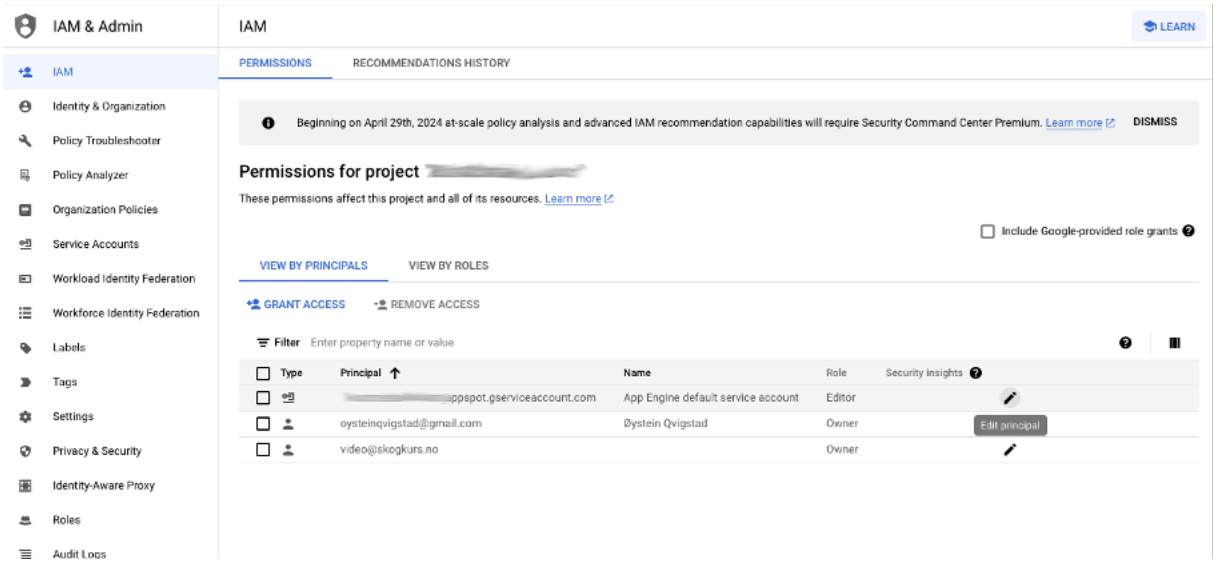
I hamburgermenyen i venstre hjørne anbefales det å feste de mest brukte produktene. Bruk «More Products»-knappen for å finne produkter som ikke allerede er festet.



1.3 Tilganger

Det må opprettes en *Service Account*. Dette er en konto som ikke tilhører en person, men benyttes i kontrollflyten når tjenesten skal oppdateres fra Github.

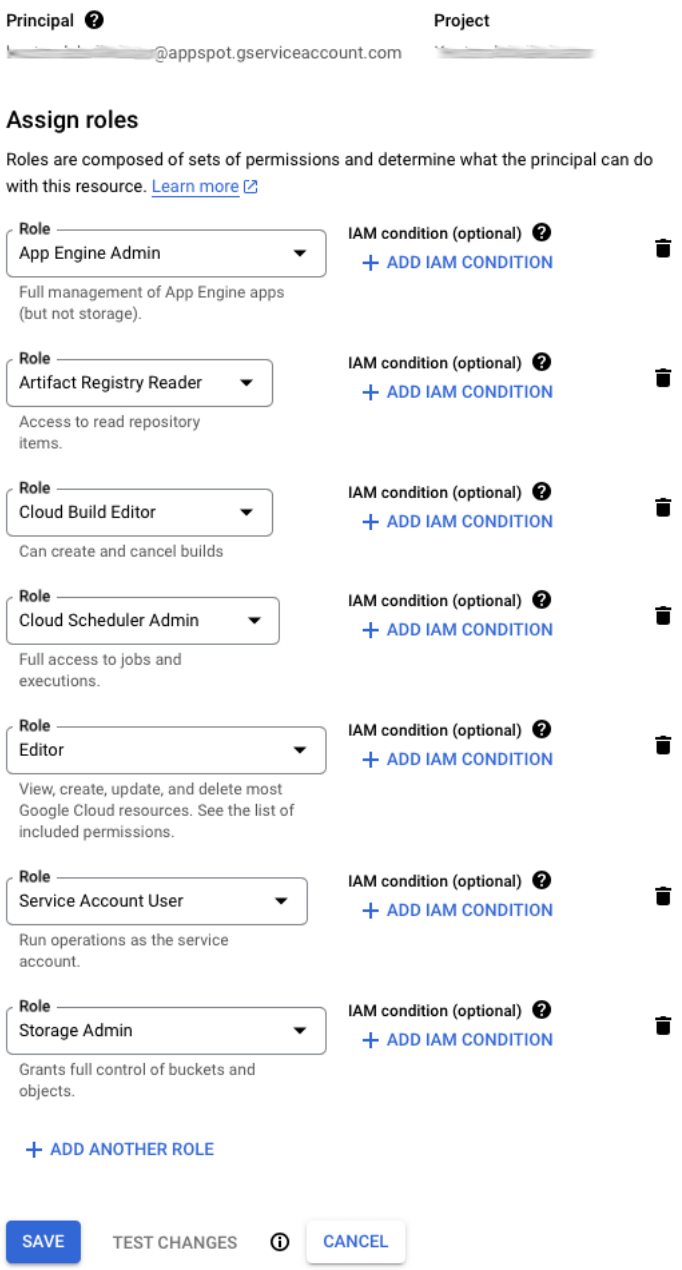
1. I vår tilfelle fulgte vi punkt 1.4.1 - 1.4.3 for å opprette en standard service account med rettigheter tilhørende App Engine.
2. Gå deretter til **IAM & Admin** (hamburgermeny) → **Service Accounts** for å bekrefte at denne er opprettet.
3. Velg deretter **Keys**-fanen i toppen, og velg **Add Keys → Create new key**. Dette er nøkkelen som skal brukes for å oppdatere tjenesten. Velg «JSON» som nøkkeltype, og last den opp som beskrevet i seksjon «2. GitHub» senere i dette dokumentet.
4. Gå deretter til **IAM & Admin** (hamburgermeny) → **IAM**
5. Trykk blyantknappen **Edit principal** på servicekontoen, i vårt tilfelle REDACTED*@appspot.gserviceaccount.com*.



6. Gi følgende roller:

- App Engine Admin
- Artifact Registry Reader
- Cloud Build Editor
- Cloud Scheduler Admin
- Editor
- Service Account User
- Storage Admin

Begrunnelser/forklaringer for hvorfor rollene er nødvendige er gitt ved https://github.com/google-github-actions/deploy-appengine



7. I tillegg må det også aktiveres API-er. Dette er for at diverse programvare skal kunne kommunisere med ulike tjenester. Gå til **APIs & Services** i hamburgermenyen.
8. Verifiser at «App Engine Admin API» og «Cloud Firestore API» er aktivert, hvis ikke så aktiver dem. Det kan også være aktuelt å aktivere flere, iht. skjermbildet under.

Filter Filter

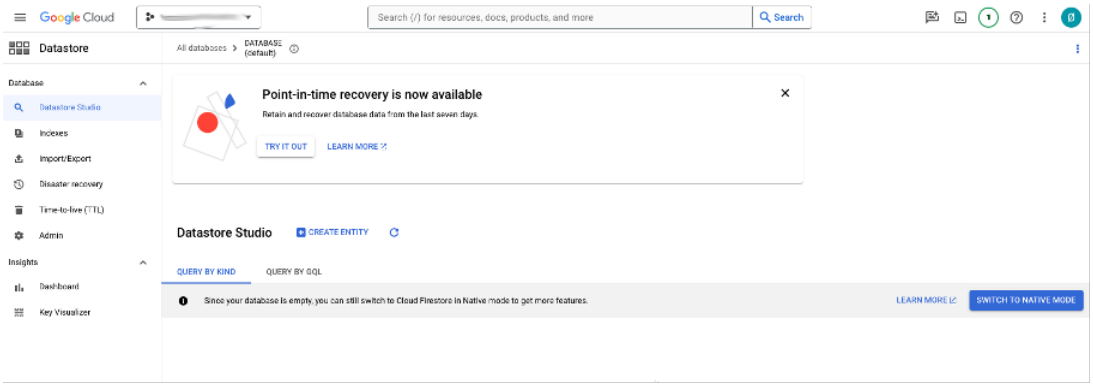
Name	Requests	Errors (%)	Latency, median (ms)	Latency, 95% (ms)
App Engine Admin API	239	47	132	255
Cloud Logging API	52	11	77	125
Cloud Firestore API	28	17	54	511
Cloud Datastore API	9	0	46	101
Cloud Pub/Sub API	6	100	229	484
App Engine				
Artifact Registry API				
Cloud Build API				
Container Registry API				
Firebase Rules API				
Google Cloud Storage JSON API				

1.4 App Engine oppsett

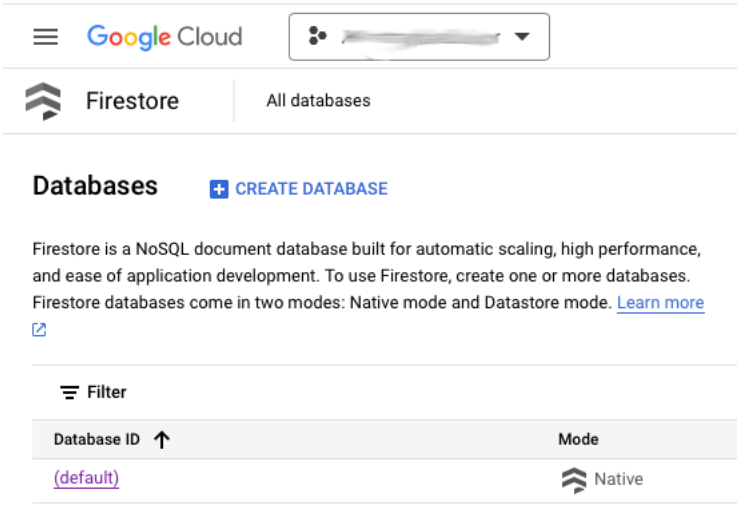
1. Gå til **App Engine** (hamburgermeny) → **Dashboard**
2. Klikk **Create Application**
3. Velg region/datasenter:
   I dette tilfellet valgte vi «europe-west3»
   Velg service account:
   I dettet tilfellet valgte vi «App Engine default service account»
4. Man skal nå ha fått bekreftelse på at «App Engine» er opprettet.
5. Sørg for at alle tilganger er gitt ved å følge seksjon «1.3 Tilganger».
6. Følg seksjon «1.6 Utrulling» for manuell utrulling og «2 Github» for automatisk utrulling.

1.5 Database oppsett

1. Gå til **Firestore** (hamburgermenyen).
2. Velg «(default)» databasen. Denne vil benyttes for prosjektet. Det kan opprettes ytterlige databaser, men disse vil ikke ha støtte for alle funksjoner.
3. Velg å endre database til «Cloud Firestore Native». Dette er nødvendig for å benytte seg av Firestore APIet, og andre tilleggsfunksjoner.

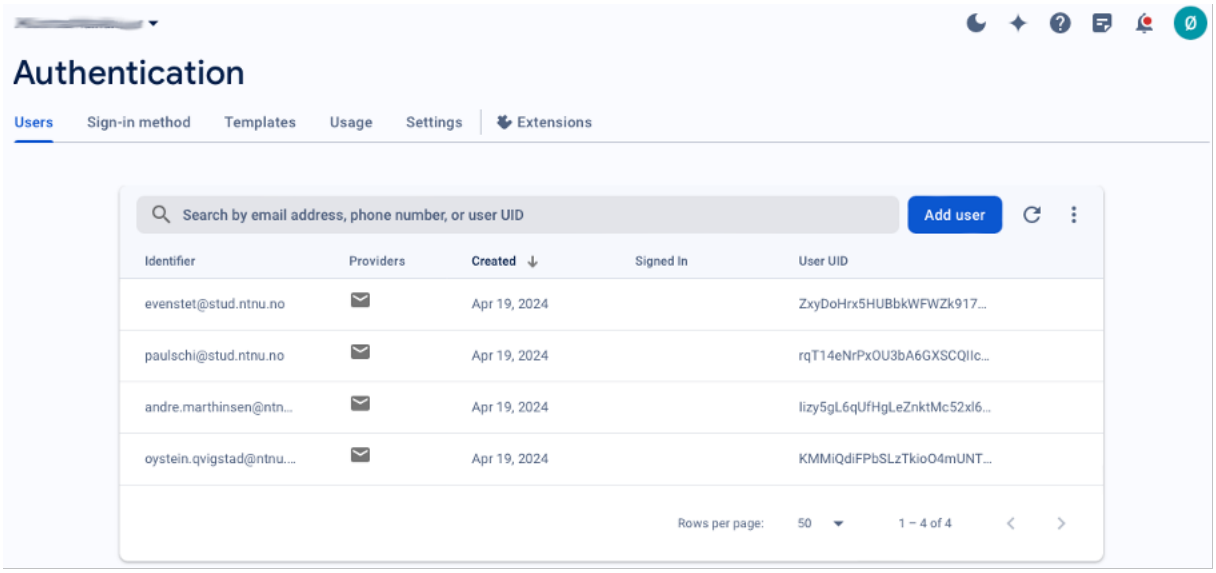


4. Ved å returnere til **Firestore** i hamburgermenyen, kan man bekrefte om Firebase er i «Native» mode. Hvis dette ikke er tilfellet, kan det være nødvendig å slette og opprette «(default)» databasen på nytt.

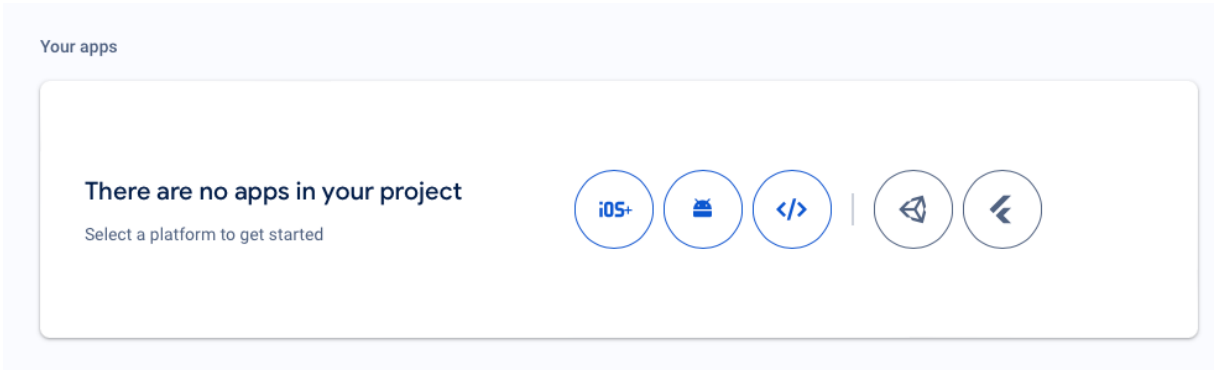


1.6 Autentiseringstjeneste

1. Sørg for at databasen er satt opp iht. 1.5 først
2. Gå til https://console.firebase.google.com og opprett et nytt firebase prosjekt, som knyttes til Google Cloud-prosjektet (Firebase vil foreslå samme prosjektnavn).
3. Gå til **Build → Authentication** i Firebase hovedmenyen
4. Velg «Email/Password» som *sign-in method*
5. Aktiver "Email link (passwordless sign-in)"
6. Deretter må brukere som skal ha tilgang opprettes. Gå til «Users», og legg til brukere. Passord for hver må opprettes, selv om det ikke skal brukes. Det anbefales å skrive inn et langt og sikkert passord, da dette i teorien kan misbrukes.

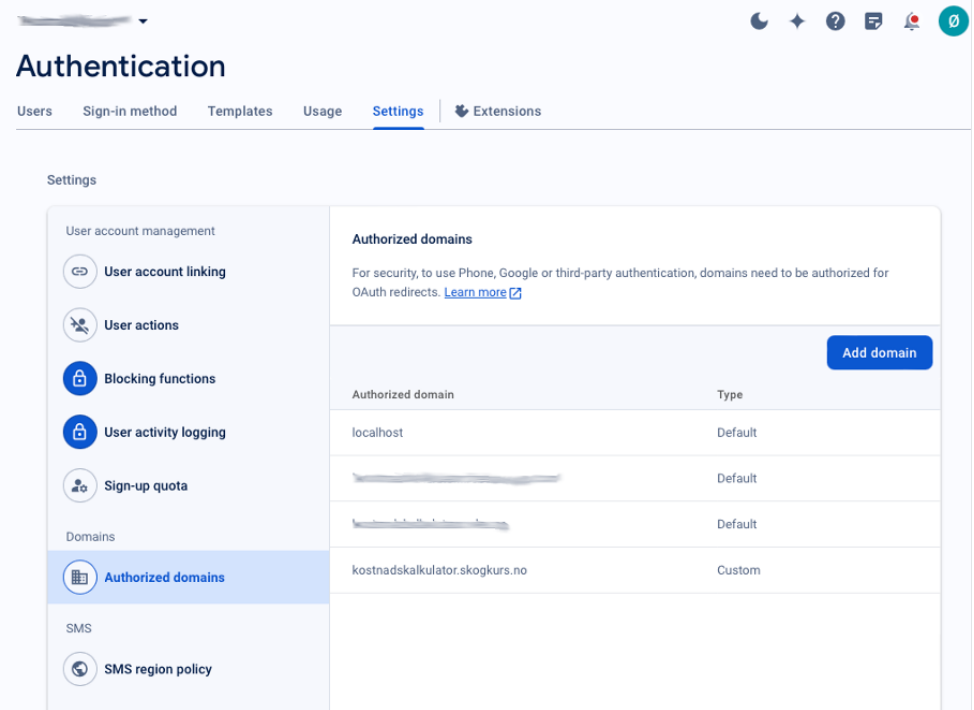


7. Neste steg er å knytte editor-klienten til tjenesten. Gå til **Project Settings > Apps**. Og velg Web (ikonet ser ut som «</>»).



Det vil bli opprettet et konfigurasjonsobjekt som skal legges inn i kildekoden til editoren.

8. Til slutt, må man autorisere domenet man ønsker at passwordless lenke skal brukes mot, i dette tilfellet https://kostnadskalkulator.skogkurs.no



1.7 Manuell utrulling av tjeneste/applikasjon

Oppdatering av kildekode til App Engine er en automatisert prosess som håndteres av Github Actions i seksjon «2 GitHub» under. Dersom man har behov for å laste opp oppdatert kode manuelt kan man gjøre dette ved å:

1. Laste ned og installere «Google SDK» på datamaskinen.
2. Åpne terminal på datamaskinen initiere kontoopplysninger opp mot prosjektet:

   `*gcloud init*`

   Nettleser vil da åpne seg slik at man kan logge inn med Google konto.
3. Gå til prosjektets kildekode

   `*cd /src*`

4. Installer avhengigheter og bygg koden (NB: *node.js/npm* forventes å være installert)

   `*npm install*`

   `*npm run build*`

5. Last opp koden

   `*gcloud app deploy*`

Man vil da bli veiledet gjennom prosessen, hvor man velger hvilket prosjekt og App Engine som skal benyttes. Selve parameterne for skalering, ytelse og HTTP-routing er definert i /src/app.yaml

2. GitHub

2.1 Overordnet

Github er tjenesten som brukes for å organisere koden under utvikling.

Prosjekt URL: https://github.com/oysteinqvigstad/SKOG-kostnadskalkulator

Viktige filer:

- *.github/workflows/github-hosted.yml*
  Script for automatisk utrulling av kode til App Engine

- */src/app.yaml*
  Script for å sette parametere i Google App Engine for miljø/skalering/routing/mm.

Viktige nøkler:

- *GCP_PROJECT_SKOGKURS*
  Prosjekt ID i Google Cloud (dokumentert øverst i seksjon «1 Tjenesteleverandør» i dette dokumentet)

- *GCP_SA_KEY_SKOGKURS*
  Nøkkel til servicekontoen i Google Cloud. Denne hentes ut manuelt fra «IAM» i Google Cloud Console.

2.2 Automatisk utrulling av tjeneste

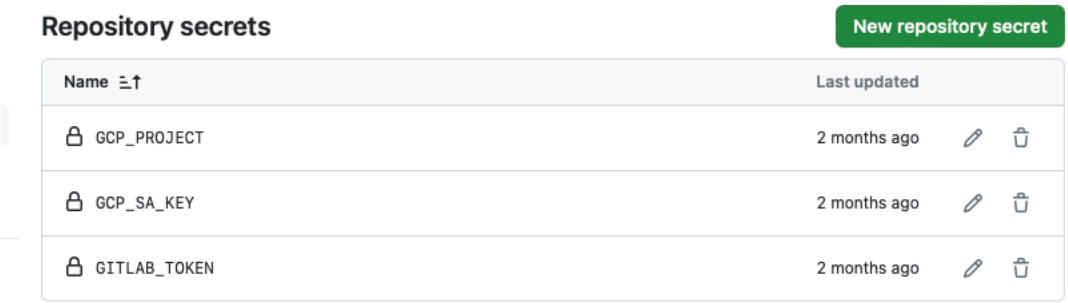
«GitHub Actions» benyttes som CI/CD (kontrinuerlig utrullingsmekanisme) for å oppdatere App Engine hver gang det forekommer en endring i kildekoden.

Ethvert privat Github-prosjekt inkluderer 2000 minutter måndentlig gratiskvote med Actions. Hver utrulling benytter omtrent 13-15 minutter med akkumulert tid.

Ved å navigere til «*Actions*» på Github-siden, vil man kunne se en liste over hver utrulling. Når koder forekommer så vil koden gå gjennom prosessen definert i filen *.github/workflows/github-hosted.yml*. Den kan beskrives slik:

Prosjektet blir bygget med Node.js versjon 16, 18 og 20. Dette er en form for kvalitetssikring av syntaksfeil i koden. Dersom alle versjonene bygget vellykket, vil det kjøres automatiske tester på kjernefunksjonalitet, også i ulike versjoner. Dersom både bygging og testing er vellykket, vil koden lastes opp og aktivere i Google App Engine, iht. steget «*deploy*».

For at Github Actions skal klare å rulle ut til Google App Engine kreves det autorisasjon. Denne gis ved å gå til **Settings → Secrets and Variables → Actions**.



Nøklenes hensikt er beskrevet lenger oppe i dokumentet, med unntak av sistnevnte nøkkel, *GITLAB_TOKEN*. Denne benyttes kun for backup til NTNUs lokale Gitlab-server, og er planlagt å bli fjernet senere.

Appendix C

User Test Report

This document describes the user test performed on the MVP version of the web app. The test method is described, along with aggregated test results organized into categories. Finally, proposed actions based on the feedback collected are listed.

Kostnadskalkulator Skogkurs

Brukertestrapport MVP

Testdato: 26.02.24 - 01.03.24

Mål for testen:

- Avdekke svakheter og mangler i brukergrensesnitt, under følgende kategorier:
    - Navigasjon
        - Er det enkelt å bruke input-kategorinavbar til å navigere mellom info, input og resultatside?
        - Er informasjon i appen plassert der brukeren forventer?
        - Navigasjon (returknapp, hamburgermeny, forsidebanner) i topp-bar, er disse intuitive?
    - Presentasjon av informasjon
        - Er resultatsiden forståelig? Er resultatene det brukerne er interessert i?
            - Er "Innsikt i kostnadsdrivere"-grafen intuitiv for brukerne?
        - Kommuniserer informasjonsknappene ved hvert input-felt sin funksjon?
        - Er det tydelig at tallene som ligger i inputfeltene fra før er standardverdier og ikke placeholder?
        - Er reset-knappen intuitiv?
        - Er det tydelig når bruker har fylt inn en verdi?
    - Tilbakemeldinger til bruker
        - Får brukeren tilstrekkelig respons ved interaksjon?
        - Får brukeren forståelige feilmeldinger?
    - Arbeidsflyt
        - Skjønner brukerne intuitivt hvordan og til hvilke formål appen skal brukes?
        - Er det behov for en knapp for å resette alle input-felter?
- Sekundært:
    - Funksjonalitet

Av mindre interesse er spørsmål om hvilke parametere som skal inn i kalkulatoren og andre rent skogfaglige spørsmål - dette bør imidlertid kommuniseres til Skogkurs.

Brukertesten er i hovedsak kvalitativ, der brukernes subjektive meninger om ulike forhold ved appen kartlegges. Innsamlet informasjon danner grunnlaget for videre utvikling av applikasjonen.

Gjennomføring:

- 6-7 brukere - representerer tre brukergrupper: skogeiere, maskinentreprenører og skogdriftsledere.
- Alle brukere tester appen innenfor samme uke, og med identisk versjon av appen.
- Brukere tester appen i browser på PC, men med simulert mobilvisning. Applikasjonen er på test-tidspunktet best tilpasset mobilvisning. Brukere kunne ha testet på mobil, men det ville blitt vanskeligere for oss å gjøre observasjoner.
- Hver test estimert til 20-30 minutter
    - Innledende spørsmål
    - Praktiske oppgaver
    - Avsluttende spørsmål
- Vektlegger at brukeren skal tenke høyt og dele betraktninger rundt bruk av og navigasjon i applikasjonen - dette for å avdekke lite brukervennlige aspekter i grensesnittet. Oppfølgingsspørsmål stilles underveis etter behov.
- Observasjon av brukeren under testgjennomføringen
    - Noterer sitater fra brukeren og beskrivelser av brukerens handlinger gjort under testing
    - Etter endt test kommenteres observasjonen for å klargjøre utydelige aspekter eller for å trekke ut innsikt fra observasjonene.

Aggregering av informasjon:

- Vi har gjennomgått testene bruker for bruker, og sett etter tendenser som deretter ble sortert inn i kategoriene i tabellen under.
- Vi har også i tabellen under lagt inn egne betraktninger - dette kunne kanskje være delt ut i en egen kolonne: på den andre siden er vi nå så langt fra direkte testobservasjoner at det er vanskelig/lite ønskelig å opprettholde et klart skille.
- En del av innsikten angår områder vi ikke hadde som mål å teste, og kategoriene ble bestemt av både testresultatene og testmålene. Flere av testspørsmålene/oppgavene var dessuten åpne for å tillate nettopp en slik fleksibilitet.
- De to første kategoriene er for å kartlegge brukernes bruk av digitale verktøy.
- Fokuset ligger på å få frem verdifulle innsikter fra testen, og ikke på å opprettholde en rigid kategori-inndeling. En viss overlapp må påregnes.

Kategorisert informasjon

Kategori	Betraktninger
• Digitale enheter brukt i skogen	• Mobiltelefon og nettbrett: spenn av forskjellige produsenter, enkelte organisasjoner har f.eks. bare Apple-produkter • Noen har også tilgang til PC på kontoret/i hogstmaskin
• Programvare/ andre kalkulatorer	• Allma - kartsystem, det vanligste verktøyet blant brukerne (noen bruker også andre kartapper) • Applikasjon for skogbruksplan • Egne applikasjoner for feltdata (ferdskriver osv.), og generell dataflyt mellom maskin og oppdragsgiver • Egne interne kalkulatorer (spesifikt for organisasjon/bedrift) - iblant basert på Skogkurs sin. • I overkant av halvparten kjenner til Skogkurs sin eksisterende kalkulator, men ingen benytter den regelmessig. • Organisasjonsspesifikke applikasjoner
Nåværende funksjonalitet	• Hovedinntrykket er at brukerne er fornøyd med eksisterende funksjonalitet - men at det er rom for forbedring (se neste Funksjonalitetsendring og Parameterendring) • De fleste brukere tar “Innsikt i kostnadsdrivere”-grafen intuitivt, og forstår både hvordan den skal brukes og forstås. Flere uttrykker at det er nyttig å kunne se hvordan en kostnadsdriver påvirker resultatet i en slikt grensesnitt. • De fleste brukere skjønner formålet til reset-knappen i input-feltene, én tror at den trigger ny beregning av resultat, en annen tror at den stiller tallet til null og ikke default.

Funksjonalitetsendring	• Foreslåtte tilleggskostnader er godt mottatt av flere brukere • Ønske om å kunne dele resultat (implementert per testdato, men ble ikke testet på grunn av uheldig format på mobil) • Ønske om å lagre resultat. Ønske om å knytte navn på skogeier, hogstfelt e.l. til disse • Ønske om å kunne lagre egne defaultverdier • Ønske om å kunne beregne flere drifter som man får vektet snittverdier ut/knytte flere kalkulasjoner sammen. Særlig interessant når det store forskjeller innenfor en drift/et hogstfelt (dette er ofte tilfelle ved for eksempel veiforhold). Dessuten ved større drifter, er det mer interessant å se et volumveid snitt for kjøringer, fordi man kan måtte kjøre langt etter et lite volum, og kort etter et stort ett (eller motsatt). • Én uttrykker ønske om å kunne sammenligne sine verdier med snittverdier. Det er imidlertid uklart hvilke snittverdier man skulle lagt til grunn. • Reset alle felter-knapp: Delte meninger om behovet for dette, men kan være fint ved mange beregninger. Også litt delte meninger om funksjonen; tilbakestill til 0 (hvis mulig) eller default. Egen knapp for å starte ny kalkulasjon ønsket av enkelte brukere, heller enn reset. Noe bekymring rundt tap av data ved feiltrykk på resetknapp. • Én bruker: Gjøre det kompatibelt med kartverk for å få ut riktige verdier, slik at brukeren må registrere mindre selv.

Parameterendring	• Foreslåtte tilleggskostnader er godt mottatt av flere brukere
	• Ønske om tilleggsparametere
	  ○ Treantall per kubikk (Flere ønsker dette sterkt!)
	  ○ Total driftstørrelse (totalt volum som skal ut, eventuelt areal for hogsten som deretter ganges med volum per areal)
	  ○ Sesong/årstid, da dette er vesentlig for kjøreforholdene
	  ○ Hogstform (lukket vs åpen) - noen påstår at det er den viktigste enkeltfaktoren for deres kostnadsberegning
	  ○ Forhåndsrydding som ekstrakostnad
	  ○ Én bruker ønsker også å legge inn kubikk per time som overstyrer de andre parameterne. Dette er jo noe av det som beregnes i kalkulatoren. Mulig at vi feiloppfattet hva brukeren uttrykte.

Resultatside	• Viktigst: "Produktivitet" og "kostnad per kubikk" virker å være de fleste brukere er opptatt av.
	  ○ Begge tallene bør vises for begge maskinene, ikke bare ett i midten av sirkelgraf/speedometer
	• Graf virker å være forståelig for de fleste, og dessuten virker den å tilby verdifull funksjonalitet. Litt forvirring om valg mellom "vis som kostnad" og "vis som produktivitet"
	  ○ Dessuten litt forvirring om det å velge en annen kostnadsdrivere i grafen påvirker de andre displayene.
	• Enkelte har også uttrykt ønske om å se totalkostnad for hele driften (kubikkostnad ganger driftsvolum)
	• Litt delt tilbakemelding på om tilleggskostnader bør vises separat eller innbakt i kostnad per kubikk. For små drifter blir "faste" kostnader (som flytting og rigg) en større andel av totalkostnaden for drifta.

Tilbakemeldinger/respons i appen	• Brukere er generelt fornøyde med respons og tilbakemelding.
	• Noen foreslår at det kan komme melding ved ikke-utfylt verdi, men er usikre på om det kan bli litt masete.
	  ○ De fleste syns at det bra at de ikke kan gå til resultat ved manglende verdi.
	  ○ De fleste syns også at det greit at man kan gå bort fra siden der det mangler verdi.
	• Lite eksplisitt feedback på at det som står i feltene oppfattes som standardverdi (ca. halvparten sier det eksplisitt) - likevel er det lite problematikk knyttet til å fylle inn verdier selv om standardverdien ikke forsvinner automatisk.
	• Kun et mindretall av brukerne uttrykker at de legger merke til at deres innfylte verdier blir uthevet. Vi registrerer ingen forvirring knyttet til hvilke felter de selv har fylt ut, men vi tester heller ikke direkte for dette. Usikkert om nåværende løsning kan føre til forvirring rundt det om en verdi er standardverdi eller fylt ut av bruker.
	• Enkelte foreslår varsel om man legger inn urimelige verdier (disse må i så fall defineres)

Organisering og utforming av informasjon	• De fleste brukere finner lett informasjon om en kostnadsdriver ved å trykke på tilhørende info-knapp. • Brukere finner informasjon vi ber dem om i testen raskt og enkelt. • Kun et fåtall av brukerne er kjent med begrepet G15-time. Forvirring kan være knyttet til “G15” (som er brukt i appen) snarere enn “G15-time”. Alle bruker en variant av time som måleenhet for produktivitet. • Én uttrykker forvirring rundt at helling oppgis i prosent og ikke grader (i hogstmaskin vises grader)

Navigasjon	• Input- og resultatside
	  ○ Noen har uttrykt ønske om piler mellom de forskjellige input-kategorisidene så det blir tydelige at det finnes flere kategorier enn de som er synlig i kategori-navbar. Flere brukere virker å slite med å skjønne at man må scrolle i kategori-bar for å se alle knapper. Dette kan nok delvis skyldes at vi simulerer mobil på en PC, og at navigasjonen ville falt mer naturlig ved touch-input.
	  ○ Noen sliter også med å finne resultatsiden, når denne knappen ikke er synlig i scrollable-kategoribar og de prøver da hamburgermeny
	  ○ Noen brukere foreslår en "neste"-knapp på bunnen av hver side for å indikere flere sider igjen
	• De aller fleste går i hamburgermeny for å finne informasjon om forskningsgrunnlag. Noen leter på andre sider, men går direkte til hamburgermeny når de ikke finner det de leter etter på siden de først gikk til. Bør denne fortsatt være utskilt fra info-side?
	• De aller fleste bruker retur-knapp i navbar for å gå tilbake til siden de kom fra. Majoriteten bruker den også for å komme tilbake til forsiden, kun et fåtall bruker banner.
	• Noen ønsker å slippe å scrolle i kategoribar, fordi det kan bli vanskelig med våte fingre på våt skjerm i skogen.
	• Generelt virker det om om navigasjonen er ganske intuitiv for de fleste.

Arbeidsflyt	• Virker som om de fleste synes appen er ryddig og oversiktlig bygget opp. • Enkelte har uttrykt ønske om at resultat og inputfelt skal vises samtidig, uten at man må navigere mellom input og resultatsider.

Andre innsikter	• Brukerne har stor interesse i at dette er en telefonapp, som også fungerer uten nettilgang
	• Litt delt om det viktigste er en enkel kalkulator som gir et "godt nok" resultat eller en mer kompleks tjeneste som gir et så nøyaktig resultat som mulig.
	  ○ Enkelte brukere påstår at det må være enkelt for å få oppslutning i brukergruppen, andre sier det stikk motsatte.
	  ○ Det er også delte meninger om en slik app i det hele tatt kan bli for komplisert. Maskinførere bruker for eksempel mer avanserte verktøy i sitt daglige virke, men dette kan variere med de andre brukergruppene.
	  ○ Innsats man legger i å lære seg appen kan betale seg med muligheten for å få et nøyaktig resultat, og dette kan være en god investering.
	• Noen vil egentlig at det skal være en priskalkulator - spesifikt for å finne ut hvor mye entreprenøren skal ha betalt.

Konklusjon

Viktigste resultater:

- Hovedinntrykket er at applikasjonen fungerer bra, er godt presentert og med god navigasjon. Navigasjon mellom de ulike input-sidene er litt utydelig for noen brukere. Begrepsbruken er i noen tilfeller ukjent eller utydelig.
- De fleste brukere finner informasjonen vi ber om på første forsøk.
- Kun et fåtall trykker på logoen i top-bar for å navigere til startside. Det er kanskje ikke tydelig nok at den er klikkbar.
- Fortsatt usikkert om innhold i input-feltene forstås som default eller placeholder, dessuten om det er tydelig nok hvilke felter som har blitt fylt inn av brukeren. Det virket imidlertid ikke som presentasjonen av input-feltene førte til problemer for noen av brukerne.
- Resultatsiden blir generelt godt mottatt. Sirkelgrafen for produktivitet per maskin virker noe forvirrende på enkelte brukere. Det er viktig å kunne se produktivitet for begge maskiner samtidig. Grafene som viser kostnadsdrivernes påvirkning virker å være forståelige, og oppfattes som et nyttig tillegg for de fleste.
- Brukerne virker fornøyd med tilbakemeldinger og respons ved interaksjoner i appen, for eksempel i feilsituasjoner.
- Flere ønsker å kunne lagre eller dele resultat av en beregning.
- Litt delte meninger om behovet for en "reset alle felter"-knapp. Noen uttrykker bekymring for tap av data om man trykker på den ved et uhell. Se ellers i tabell for mer utfyllende info (Funksjonalitetsendring).
- Noe delte meninger om tilleggskostnader skal vises som et eget resultat eller bli bakt inn i kubikkostnaden.
- Noe delte meninger om applikasjonen skal være så nøyaktig som mulig eller så lett å bruke som mulig. Underforstått: Skal det legges til eller fjernes parametere? Vi spiller ballen videre til Skogkurs.
- Ønske om at trær per kubikk, driftstørrelse og hogstform blir inkludert som parametere.

Foreslåtte tiltak (se tabell for utfyllende brukerønskeliste):

- For å fange opp tilleggskostnader som kun enkelte brukere har kan vi lage et samlefelt som heter f.eks. "Andre tilleggskostnader"
- Tydeliggjøre at grafen på resultatsiden ikke påvirker beregningen; den viser bare den enkelte kostnadsdrivers påvirkning. Vi kan for eksempel legge til en info-knapp tilsvarende den som er på hvert input-felt.
- Legg til piler på kategori-bar eller "neste"-knapp, eller på annen måte gjøre det tydeligere at det kan være flere kategorier enn de som synes.
- Toppbar, inkludert hamburgermeny beholdes. Vurder å gjøre det tydeligere at man kan trykke på logoen for navigasjon til forside.
- Legg til funksjon for lagring av resultat lokalt på brukers enhet.

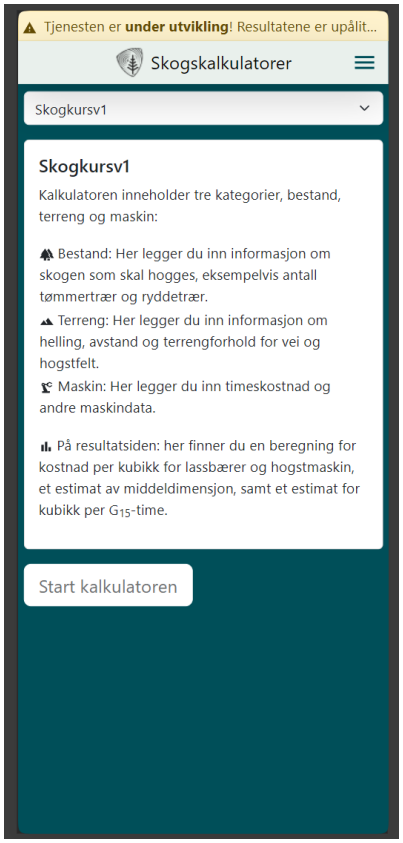
- Tilpass resultatdelingsrubrikk til mobilvisning.
- Lag “Start ny kalkulasjon”-knapp som lar brukere starte prosessen på ny med tilbakestilte verdier - med bekreftelsesspørsmål.

Innsikter og ønsker som må behandles av Skogkurs:

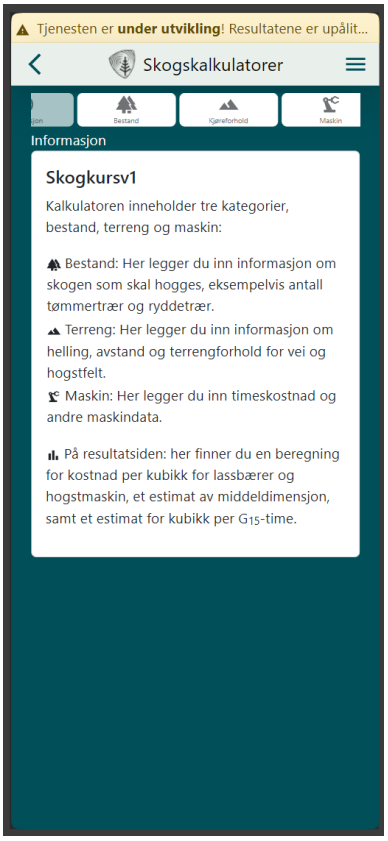
- Det bør oppklares hva G15 betyr - Legg til -timer der det bare står G15. Legg inn forklaring på hva G15 betyr på infosiden til kalkulatoren. Dette drøftes med Skogkurs.

- Vurdering av om tilleggskostnader skal fordeles per kubikk eller vises som ren kostnad
- Legge til parametere som driftstørrelse, tre per buikk og hogstform (se for øvrig “Parameterendring” i tabell over for fullstendig liste)
- Foreta en avveining mellom enkelhet kontra nøyaktighet - hva er viktigst?

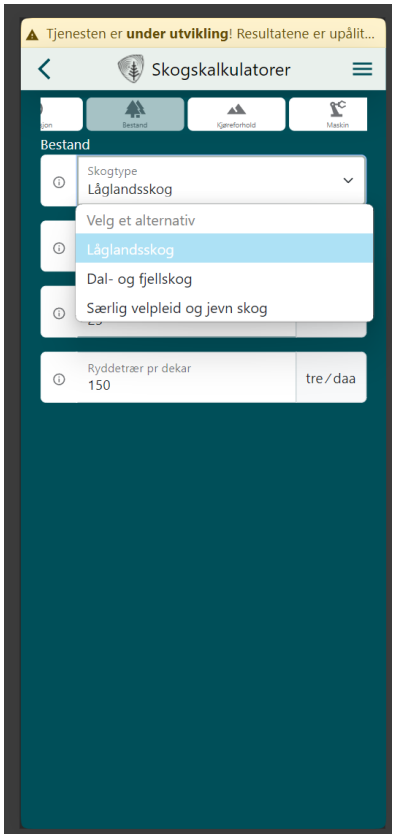
Skjermbilder fra testet versjon:



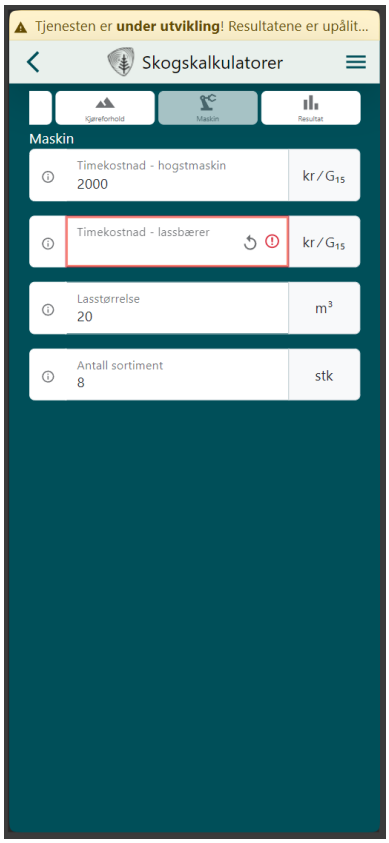
Startside



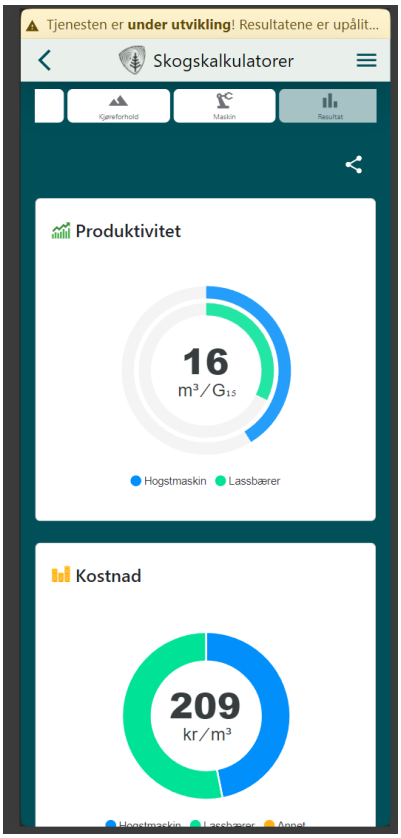
Infoside



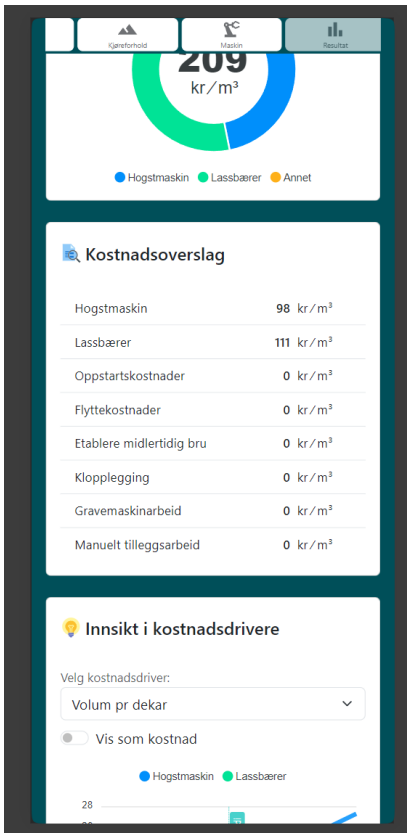
Input-side (eksempel)



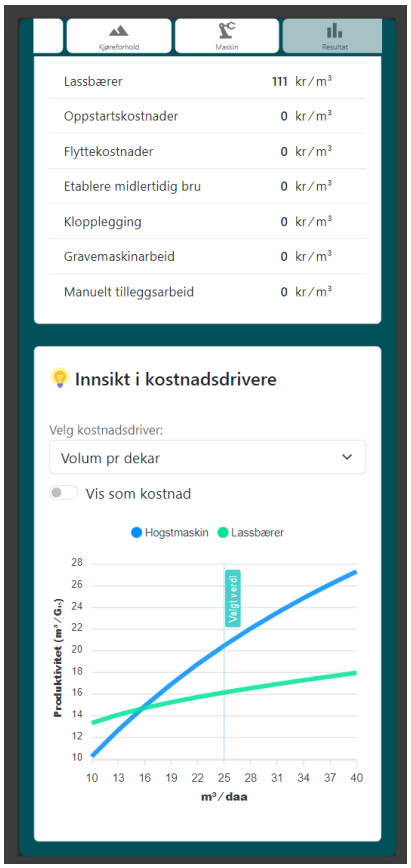
Input-side, med tomt felt



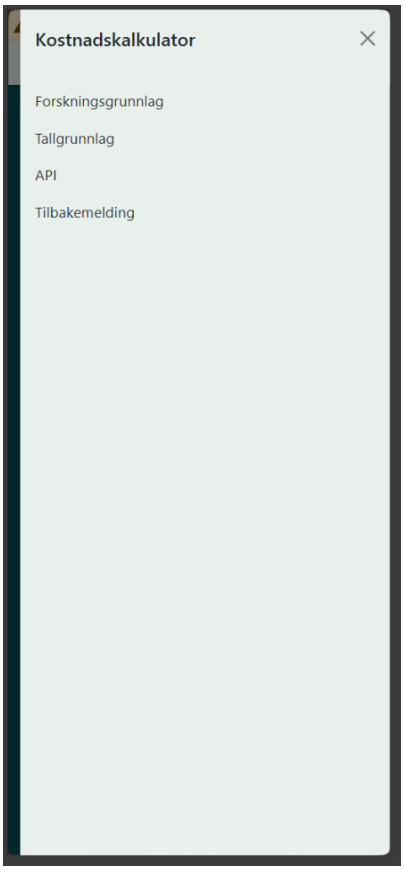
Resultatvisning



Tilleggskostnader (ikke implementert input)



Graf for innsikt i kostnadsdrivere



Innhold i “hamburger”-meny

Appendix D

Assignment Description

This document is the original bachelor's project proposal submitted by Skogkurs to NTNU. It contains a description of the project, outlining its primary objectives and the requirements for the final outcome.

Oppgave 10 BPROG, BIDATA 2-4stk

Oppgavetittel: Nettbasert kostnadskalkulator for skogsbedrifter

Bedrift:	Skogkurs
Kontaktperson:	Mikael Fønhus
E-post:	mf@skogkurs.no
Telefon:	99287002
Tilholdssted:	Brumunddal

Beskrivelse av oppgaven:

BAKGRUNN

Skogkurs sin produktivitets- og kostnadskalkulator, kalt Norske produksjonsnormer, ble lansert i 2018. Den har blitt svært godt mottatt i næringen og veldig mange har lastet den den til lokale PC-er. Kalkulatoren har blitt svært populær i entreprenørbransjen og brukes av mange for å simulere og forklare konsekvensen av ulike kostnadsdrivere.
Kalkulatoren er imidlertid bygd i Excel, og når du har lastet ned én gang, får du ikke tilgang til endringer og oppdateringer før du eventuelt laster ned på nytt. Det er det erfaringsmessig sjelden folk gjør.
For at brukerne til enhver tid skal ha oppdaterte versjonen, vil vi gjøre kalkulatoren nettbasert. Dette har blitt etterspurt av mange, og særlig fra skogsentreprenørene.

OPPGAVENS HOVEDMÅL

Bygge en nettbasert Produktivitets- og kostnadskalkulator med samme forskningsbaserte datagrunnlag som dagens Excelversjon.

DELMÅL

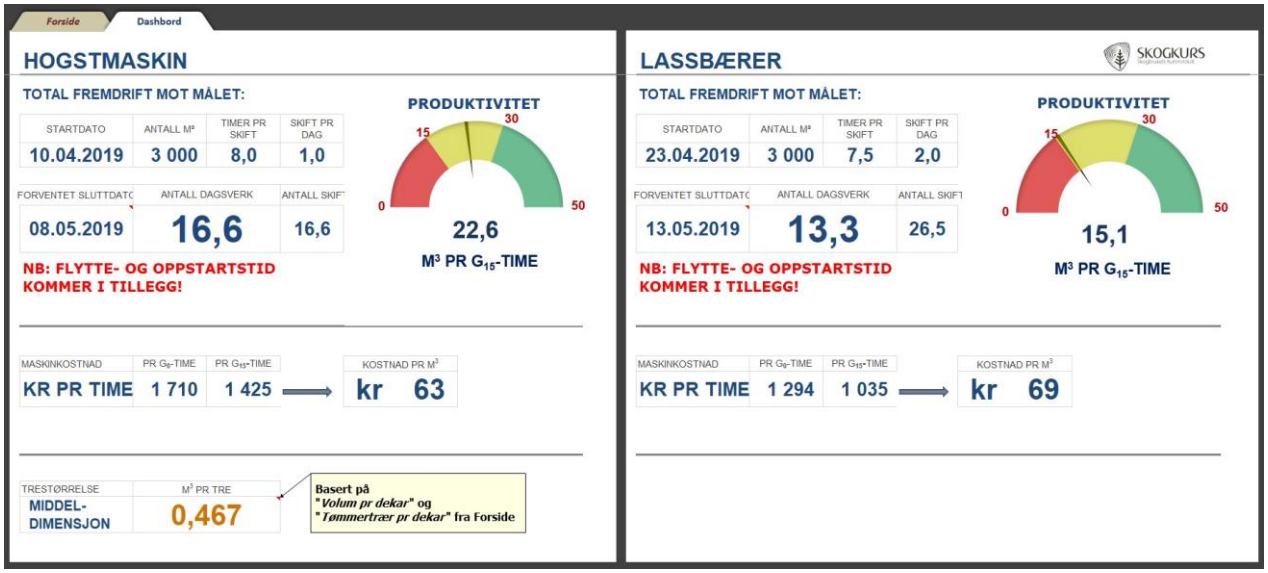
a) Sikre at alle brukere til enhver tid har siste kalkyleversjon
b) Gjøre kalkulatoren tilgjengelig på alle plattformer, både PC, nettbrett og mobil.
c) Kalkulatoren kan brukes i felt som støtte i planleggingen ute.
d) Kunne ta inn nye kostnadsdrivere etter hvert som nye forskningsresultater skal formidles og tas i bruk

KRAV TIL GJENNOMFØRING:

Bygge en nettbasert løsning der brukeren kan gjøre sine beregninger direkte på nettet, uavhengig av programvare og plattform (PC/Mac, nettbrett, smarttelefon).
Gjøres gjerne i tett samarbeid med oppgavestilleren.

Vedlagte bildefiler viser skjermdump av Excel-versjonen. Filen kan lastes ned med lenken nedenfor, men inneholder mange makroer som diverse brannmurer kan stenge for: https://skogkurs.no/artikkel/verktoy-for-kalkulasjon-av-skogsdrift/


Forside
Dashbord
PRODUKTIVITETS- OG KOSTNADSKALKULATOR
Revidert 04.03.2019
Versjon 1.8
Dal- og fjellskog
Låglandsskog
Særlig velpleid og jevn skog - bruk av ekstra store maskiner
Driftsstørrelse 3000 m³
Volum pr dekar 70 m³/daa
Tømmertrær pr dekar 150 tre/daa
Ryddetrær pr dekar 75 tre/daa
Hogstmaskin - oppstartsdato 10.04.2019 dato
Hogstmaskin - timer pr skift 8,0 timer
Hogstmaskin - skift pr dag 1,0 stk
Lassbærer - oppstartsdato 23.04.2019 dato
Lassbærer - timer pr skift 7,5 timer
Lassbærer - skift pr dag 2,0 stk
Overflatestruktur i terrenget 3 Middels gode
Overflatestruktur på eventuell traktorvei 3 Middels gode
Helling på hogstfeltet 3 20-33 %
Helling på eventuell traktorvei 3 20-33 %
Kjøreavstand i terrenget 800 meter
Kjøreavstand på eventuell vei 100 meter
Lasstørrelse 20 m³
Antall sortiment 4 stk
Timekostnad - hogstmaskin 1425 kr/G15-time
Timekostnad - lassbærer 1035 kr/G15-time


Forside
Dashbord
HOGSTMASKIN
TOTAL FREMDRIFT MOT MÅLET:
STARTDATO 10.04.2019
ANTALL M³ 3 000
TIMER PR SKIFT 8,0
SKIFT PR DAG 1,0
FORVENTET SLUTTDATO 08.05.2019
ANTALL DAGSVERK 16,6
ANTALL SKIFT 16,6
NB: FLYTTE- OG OPPSTARTSTID KOMMER I TILLEGG!
PRODUKTIVITET
0 15 30 50
22,6
M³ PR G15-TIME
MASKINKOSTNAD KR PR TIME
PR G9-TIME 1 710
PR G15-TIME 1 425
KOSTNAD PR M³ kr 63
TRESTØRRELSE MIDDEL-DIMENSJON
M³ PR TRE 0,467
Basert på
"Volum pr dekar" og
"Tømmertrær pr dekar" fra Forside
LASSBÆRER
SKOGKURS
TOTAL FREMDRIFT MOT MÅLET:
STARTDATO 23.04.2019
ANTALL M³ 3 000
TIMER PR SKIFT 7,5
SKIFT PR DAG 2,0
FORVENTET SLUTTDATO 13.05.2019
ANTALL DAGSVERK 13,3
ANTALL SKIFT 26,5
NB: FLYTTE- OG OPPSTARTSTID KOMMER I TILLEGG!
PRODUKTIVITET
0 15 30 50
15,1
M³ PR G15-TIME
MASKINKOSTNAD KR PR TIME
PR G9-TIME 1 294
PR G15-TIME 1 035
KOSTNAD PR M³ kr 69

Appendix E

Project Plan

This document contain the original plan outlined by the team; especially focusing on methodology and unifying the team's perceived scope and goals for the project.

Prosjektplan

1. Mål

1.1 Bakgrunn

Vår oppdragsgiver Skogkurs (Skogkurs, u.å.) forvalter kostnadskalkulator som lar en skogsentreprenør eller annen interessent i skogbruket regne ut kostnaden av en skogsdrift. Denne kostnadskalkulatoren har per prosjektstart form av et ganske komplisert excel-ark som har blitt satt opp og vedlikeholdt av en ekstern aktør. Skogkurs hadde et ønske om at denne kalkulatoren skulle bli nettbasert, slik at den sikrer at alle brukere bruker siste kalkyleversjon, at den er tilgjengelig på et bredt spekter av enhetstyper og -størrelser, og ikke minst at den tilgjengeliggjøres også for benyttelse i felt.

- Excel-arket inneholder makroer, og er derfor stadig mer problematisk å bruke (restriksjoner)
- Excel-arket begrenser bruk til lap- og desktop-pcer, ikke tilrettelagt for feltbruk.
    - Nedlastet versjon er ikke nødvendigvis oppdatert
- Ønske om utvidelse av kalkulator for å beregne lønnsomhet av andre hogstformer enn kun flatehogst (lukket hogst i første omgang)

1.2 Produktmål

- Applikasjonen skal støtte opprettelse av nye formler med andre kostnadsdrivere
- Applikasjonen skal støtte lagring og uthenting av formelversjoner og kostnadskalkulasjoner
- Applikasjonen skal støtte bruk på flere typer enheter (PC, nettbrett, mobil)
- Applikasjonen skal støtte flere operativsystemer, som Windows, iOS og Android
- Applikasjonen skal støtte bruk i feltet, også uten nett-tilgang

1.3 Effektmål

- Applikasjonen skal sikre at alle brukere til en hver tid har tilgang til siste versjon av kalklyleformelen
- Applikasjonen skal kunne fungere som et verktøy for skogdriftsledere, entreprenører, skogeiere og andre interessenter når de planlegger hogst - i større grad enn ved bruken av gammel versjon
- Applikasjonen skal kunne inngå verktøy i kursopplegget “bedriftsøkonomi for entreprenører”

1.4 Læringsmål

- Få erfaring med å analysere en problemstilling fra en virkelig oppdragsgiver, samt formulere og gjennomføre en løsning til en slik problemstilling

- Få dypere innsikt i ferdigheter og temaer knytt til web-utvikling
- Få erfaring med å gjennomføre og integrere tilbakemelding fra brukertester med brukere fra det virkelige liv

2. Omfang

2.1 Domene

I motsetning til timeoppgjør som er vanlig i mange andre bransjer er det i skogbruket vanlig å få oppgjør for antall kubikkmeter tømmer produsert. Målet er dermed å kunne maksimere tømmerproduksjonen med et minimum av kostnader.

Det er mange kostnadsdrivere involvert. For eksempel vil terrengforholdene på stedet være av betydning: hellningsgrad, avstand til bilvei, mot eller med kjøring, markas bæreevne og behov for hjelpetiltak som-bro eller klopplegging være av betydning. Selvfølgelig vil skogens (eller bestandets) tilstand og struktur også være en viktig faktor. Om trærne er noenlunde like store og det er foretatt rydningshogst, vil man komme bedre ut, enn om mange små trær må ryddes unna mens man hogger. All aktivitet som ikke produserer direkte til å få tømmer ut av skogen vil således utgjøre en ekstra kostnad som påvirker sluttresultatet i større eller mindre grad.

Kostnaden påvirkes også av hvilken hogstform som benyttes. Tradisjonelt har åpen hogst, eller flatehogst vært mest brukt, særlig i granskog. Da hogger man alle trær innenfor et spesifikt område. Økt fokus på alternative hogstformer bidrar til sørre behov for å kalkulere driftskostnader knyttet til lukkede hogster. En lukket hogst krever mer av både planlegging og gjennomfing av skogsdriften. Det er flerre av kostnadsdriverne som vil agere annerledes i en lukket kontra en åpen hogst. Det er av interesse å indenfisere dette i kostnadskalkulatoren.

En kalkulator som beregner kostnadsdrivernes påvirkning på lønnsomheten i en hogst vil være av interesse for både skogbruksledere, entreprenører og skogseiere. For at entreprenører, skogeiere og tømmerkjøpere skal kunne få et best mulig estimat av hvorvidt en tømmerdrift vil være lønnsom, er det nødvendig med gode verktøy for beregning av dette.

2.2 Oppgavebeskrivelse

I den originale oppgaven var det et ønske om en nettbasert implementasjon av en konkret formel. En bakdel ved å levere en slik løsning, en nettaplikasjon som leverer en ufleksibel implementasjon av en gitt kalkyle, er behovet for å hyre inn utviklere for å justere kalkulatoren etter avsluttet prosjekt. Vi utvider derfor oppgaven til å inkludere et grensesnitt hvor forskere eller andre faglige kompetente kan lage distinkte kalkulatorer med inn og ut felt og tilhørende versjonering.

Vi anser en slik løsning for å helhetlig være mer bærekraftig for involverte parter. Ved at fagkyndige innenfor skogsdrift kan utvikle varianter av kalkulatorer uten at man er avhengige av å jobbe med selve koden vil videreutvikling kunne foregå raskere, billigere og smidigere. Den største faglige utfordringen blir da å utvikle et grensesnitt som er fleksibelt nok men samtidig

intuitivt å bruke for oppretning og endring av kalkulatorer, samt å utvikle et brukergrensesnitt for selve kalkulatoren som møter behovet ute i feltet.

2.3 Rammer

- Fra oppdragsgivers side stilles det ingen særskilte krav til teknologivalg, utviklings- eller utrullingsmiljø.
- Programvaren skal imidlertid driftes, vedlikeholdes og videreutvikles etter at vi har overlevert prosjektet
    - Derfor stiller vi særskilte krav til modulær og vedlikeholdbar kode, samt utfyllende dokumentasjon for å muliggjøre overtakelse av programvaren.

2.4 Avgrensning

Oppdateringer av kalkyle-formelen med nye variabler, som for eksempel opprigg, eller endring av variabler som beror seg på eksisterende forskningsgrunnlag, er ikke del av prosjektomfanget. Slike vurderinger bør eller må basere seg på forskningsarbeider som fastslår at eventuelle endringer i formelverket produserer et riktig resultat.

Dette vil si at applikasjon skal støtte fremtidige endringer i formelverket, fra fagpersoners side, men at slike endringer ikke tilfaller prosjektgruppen.

3. Prosjektorganisering

3.1 Medlemmer og rollefordeling

Figur 3.1, Interessenter og gruppemedlemmer

Tabell 3.1, rollefordeling og ansvarsområder

Rolle	Ansvarsbeskrivelse	Team-medlem
Scrum Master	• Overordnet ansvar for gjennomføring av Scrum ○ Organisere og lede scrum-aktiviteter som sprint-review, -planlegging osv. • Fasilitere kommunikasjon mellom utviklingsteam og PO/andre involverte • Dobbel stemme i tilfelle uenigheter	Øystein Qvigstad

UI/UX Designer	• Organisere brukertester • Analayse/innhenting av PO/interessenters grensesnittkrav • Tilgjengelighetsanalyse • Figma-eierskap	Paul Schiager

Arkitekt	• Utarbeide arkitektur
	  ◦ Kodearkitektur
	  ◦ Systemarkitektur
		André Marthinsen

Sikkerhetsansvarlig	• Verdianalyse
	• Kartlegge risikoer og tiltak
	  ◦ Deployment
	  ◦ Systemarkitektur
	  ◦ Kode
	• Etterse oppfyllelse av sikkerhetskrav
		Even Stetrud
		Paul Schiager

Dokumentasjons-ansvarlig	• Påse at diagrammer for kode lages • Identifisere tidlige rapportskrivingsmuligheter • Pådriver for rapportskriving	Even Stetrud
Deploymentansvarlig	• Utforme pipeline og Git arbeidsflyt • Koordinere oppsett/struktur for utrulling og hosting	Øystein Qvigstad
Testansvarlig	• Oppfølge utviklere på testing • Vurdere om tester er hensiktsmessig utformet	André Marthinsen

3.2 Rutiner og Regler

Generelle

- Det forventes at man holder seg oppdatert i gjeldene rutiner og regler.
- Man oppfordres til å melde fra hvis det oppstår utfordringer med å forstå eller gjennomføre oppgaver.

Fravær og sykdom

- Fravær skal meldes fra til alle i teamet så tidlig som mulig.
- Ved sykdom holder man seg hjemme for å unngå å smitte andre gruppemedlemmer.

Møter

- Fysisk oppmøte forventes i møtevirksomhet, med unntak av "Standup"-møter.
    - Digitalt oppmøte tillates dersom det avtales på forhånd og det ikke gir negativ innvirkning.
    - Møter utenfor fastsatte tidspunkter eller med kort varsel kan avvikles digitalt.
- Man oppfordres til å dele sine meninger og ideer under møter og diskusjoner.
    - Man skal være respektfull ovenfor andres synspunkter

Frister

- Det forventes at man loggføre arbeidstimer hver dag. Senest i slutten av uken.
- Det forventes at man sjekker kommunikasjonskanaler daglig.
- Det forventes at man informere om arbeid står i fare for å ikke bli fullført innenfor forventet tid.
    - Oppgaver man er tildelt i en sprintperiode forventes å bli fullført innen "Sprint Review" dersom annen beskjed ikke gis på forhånd.
    - I tilfeller hvor arbeid gjentatte ganger ikke blir fullført, vil teamet sammen komme til en enighet om å hvorvidt en advarsel skal gis. Dersom det ikke skjer forbedringer, kan det aktuelle medlemmet risikere å bli ekskludert fra bachelorgruppen.

4. Planlegging

4.1 Valg av prosessrammeverk

Prosjektet vil benytte ***Scrum*** (Schwaber & Sutherland, Scrum Guide) som primær utviklingsmodell/prosessrammeverk. Fra tidlig oppstartsmøte ble det tydelig at det ville bli utfordrende å lage en endelig kravspesifikasjon. Dette skyldes først og fremst høyt antall eksterne selskaper som deltar i prosjektet som interessenter, og at ulike målgrupper ser ut til å ha ganske så forskjellig bruksmønster. Et annet aspekt er at bedriften er åpen for ulike løsninger, og ikke selv har ytret egne ønsker til design, ol.

Vår vurdering ble derfor at en agile/iterativ utviklingsmodell - som gir oss mulighet til å innhente tilbakemeldinger regelmessig - vil resultere i lavere risiko, sammenlignet med en planbasert utviklingsmodell. Av de iterative modellene, har vi tidligere erfaring med Scrum, og anser rammeverket som kompatibelt med arbeidslivets behov og forventninger. Dermed faller valget naturlig på Scrum.

4.2 Sprintperioder

- Sprintperiode i oppstartsfasen settes til én uke med hensikt å redusere feedback-loopen.
- Senere økes sprintperioden til to uker for å gi høyere andel produktive timer.
- Forlengelse av sprints kan ikke gjøres på bakgrunn av forsinkelser.

Tabell 4.2, sprintoversikt, med lengde og dato

ID	Start	Slutt	Uker
Sprint 1	10 Jan 2024	16 Jan 2024	1
Sprint 2	17 Jan 2024	23 Jan 2024	1
Sprint 3	24 Jan 2024	30 Jan 2024	1
Sprint 4	31 Jan 2024	06 Feb 2024	1
Sprint 5	07 Feb 2024	13 Feb 2024	1
Sprint 6	14 Feb 2024	27 Feb 2024	2
Sprint 7	28 Feb 2024	12 Mar 2024	2
Sprint 8	13 Mar 2024	26 Mar 2024	2
Påskefri	27 Mar 2024	02 Apr 2024	🐣
Sprint 9	03 Apr 2024	16 Apr 2024	2
Sprint 10	17 Apr 2024	30 Apr 2024	2
Sprint 11	01 May 2024	20 May 2024	2+

4.3 Faste møter

- Tirsdag kl 13 passet best for interessenter å ha “sprint review”-møte.
- Aktivitetskalenderen er oppstykket i hele timer, og reflekterer derfor ikke lengden av hvert møte med nøyaktighet.

Tabell 4.3.1, eksempel-ukeplan med faste møter

Tid	Mandag	Tirsdag	Onsdag	Torsdag	Fredag
		Sprint slutt	**Sprint start**		
09:00	Standup	Standup	Sprint-planning	Andre emner	Standup
10:00					Backlog-raffinering
11:00					

12:00	Lunsj	Lunsj	Lunsj		Lunsj
13:00		Sprint-review	Veilednings-møte		
14:00		Sprint-retrospective			
15:00		Intern-review			
16:00					

Tabell 4.3.2, beskrivelse av sprint-aktiviteter

Aktivitet	Tid	Beskrivelse
Møter		• Møteinnkalling med Teams sendes med agenda minst én virkedag i forveien.

Sprint-planlegging	3:00	• Tilgjengelig tid estimeres for neste sprint basert på forrige sprint og timeplan
		• Product Backlog Item-er tidsestimeres av utviklerne i fellesskap
		• PBI-ere velges ut i tråd med prioritering og tid
		  ○ Utviklere plukker fra de utvalgte oppgavene basert på lyst og kompetanse.
		  ○ Der mulig deles utviklere inn to og to med et gitt ansvarsområde for å promotere par-programmering og kunskapsdeling
		  ○ Hver utvikler deler sine PBI-er opp i “tasks” og estimerer tid for gjennomføring.

Sprint-review	0:30	• Demo av sprintens resultater hvor PO og interessenter er invitert • Fokus er på at interessenter skal gi tilbakemelding • Møtet blir spilt inn gitt alle gir tillatelse • Resten av utviklerne noterer tilbakemelding i felles dokument

Sprint-retrospective	0:30	• Gjennom uken noterer medlemmer forslag til tiltak for bedre prossesstyring
		  ○ Forslagene følger kategorier: “stopp med”, “fortsett med”, “start med”
		  ○ Forslag til Retrospective
		• Scrum Master tar opp forslagene fra medlemmene til diskusjon
		  ○ Det blir tatt avgjørelser pr punkt med tilhørende tiltak
		  ○ Scrum Master sørger for at tiltak blir fulgt opp
		• Tilbakemelding fra veileder tas opp for diskusjon

Internt Kode Review	1:00	• Utviklere viser frem kode utviklet under sprinten seg i mellom.
		  ○ Sammenheng og funksjonalitet beskrives
		  ○ Spørrerunde
		• Utviklere gir status på fortrolighet med biblioteker og kodebase, behov for kunnskapsdeling luftes.

Backlog Raffinering	1:00	• PO og utviklere diskuterer ønsket forretningsmål for følgende sprint • Det opprettes nye PBI-er basert på forretningsmålet • PBIs prioriteres of deles opp i mindre PBIs for omtrent to sprinter ○ Forretningsmålet balanseres med tekniske og akademiske behov
Standup	0:15	• Kort møte hvor hvert medlem oppdaterer teamet med status på sine arbeidsoppgaver, og kan løfte behovet for bistand/diskusjon.
Veilednings-møte	0:30	• Gjennom uken noterer medlemmer spørsmål til veileder ○ Spørsmål til veileder • Scrum Master skriver agenda dagen før • Veilederens råd noteres og diskuteres senere i Sprint Retrospective

4.4 Beslutningspunkter

- Beslutninger fattes kontinuerlig og dokumenteres med `/decision`-mikroen i Confluence.
- Ukentlig:
  - Produkteieren, i samsvar med utviklingsteamet, har er ansvarlig for å fastsette prioriteringer av oppgaver (PBI-er) i hvert “backlog raffinering”-møte.
  - Endringer i arbeidsprosess vedtas vanligvis i “sprint retrospective”-møter.
- Datosatte beslutningspunkter:

Tabell 4.4, frister for beslutningspunkter

Tidspunkt	Frist/tidspunkt for beslutning
22.-24. januar	• Signering av gruppekontrakt og prosjektavtale • Intern frist for ferdigstilling og innlevering av prosjektplan
24.-25. januar	• Frist for teknologivalg og strategi for MVP
1.-5. februar	• Seneste tidspunkt for å finne deltakere og planlegge for UX-brukertesting
28.-29. februar	• Evaluere tilbakemeldinger fra UX-testing og fastsette hvilke endringer som skal implementeres innen MRF (13. mars).
14.-16. mars	• Frist for teknologivalg og strategi for administrasjonskonsoll.
3.-5. april	• Bestemme omfang/avgrensing av fremtidig utvikling og omprioritering av tid til rapportskriving
14-17. april	• Planlegge for sluttevaluering av produktet
1.-3. mai	• Seneste avslutningsfase for utvikling og overlevering av produktet

5. Organisering av Kvalitetssikring

5.1 Rammeverk for dokumentasjon og verktøyhåndtering

5.1.1 Dokumentasjon

For å sikre kunnskapsdeling innad i gruppen og fasilitere videreutvikling av produktet etter bachelorprosjektets sluttdato er det en sentral målsetting at programvaren skal dokumenteres grundig. Formålet er at en utenforstående utvikler lett kan sette seg inn i hele eller deler av kodebasen på bakgrunn av dokumentasjonen.

- Dokumenter tidlig: har man implementert noe så dokumenterer man det så fort som mulig. Har man oppdatert noe i kodebasen som fordrer oppdatering av dokumentasjon så fort som mulig.

- Det oppfordres til å produsere diagrammer og andre visuelle hjelpemidler som kan gi innsikt i software-arkitekturen, samt den overordnede strukturen innad i og mellom de forskjellige modulene.

Siden produktet skal overleveres og driftes av Skogkurs etter endt prosjektperiode skal det også produseres nødvendig dokumentasjon for utrulling og vedlikehold av programvaren.

Det etterstrebes å føre referater fra alle møter, for å sikre at alle viktige diskusjoner, beslutninger og betraktninger dokumenteres:

- Fokus er på kortfattede notater som støtter en bærekraftig møteprosess. Alle gruppemedlemmer har ansvar for å påse at det blir ført møtereferater.

5.1.2 Prosjektverktøy

Prosjekthåndtering

- **Jira** brukes for organisering av sprints, backlog items, issue board, timeline, og milepæler.
- **Confluence** brukes for å organisere dokumentasjon for prosjektet.
    - Andre verktøy kan også bli aktuelle.
    - **Overleaf** brukes for å skrive endelig bachelorrapport.
- **Github** brukes som kildekode-repository.
    - Jf. pkt. 5.1.3 i planen for flere detaljer.
- **Figma** brukes som verktøy for prototyping

Følgende tabeller viser kriteriene som må fylles Definisjon av Klar og Definisjon av Ferdig.

- Definisjon av Klar er kriterier som må oppfylles for at at en PBI kan flyttes til "Sprint backlog"
- Definisjon av Ferdig er kriterier som må oppfyles for at en PBI kan ferdigstilles.

Tabell 5.1.2.1, kriterier for Definisjon av Klar.

Definisjon av Klar
• PBI er detaljert nok til at utviklergruppen kan vurdere hvorvidt PBI kan gjennomføres
• PBIs avhengigheter er definert, og ingen avhengigheter blokkerer for fullføring av PBI
• Gruppen har tilstrekkelige ressurser til å gjennomføre PBI
• PBI har passende størrelse til å kunne gjennomføres i løpet av en sprint.

• PBI har akseptkriterier definert

Tabell 5.1.2.3, kriterier for Definisjon av Ferdig.

Definisjon av Ferdig
• Tilstrekkelig med kodekommentarer • Tilstrekkelig dokumentasjon av grensesnitt (diagrammer, ol.) • Navngivning av funksjoner/klasser/interfacer gir semantisk mening • Parprogrammerer er kjent med hele kodeendringen
• Tilstrekkelig med testing (ca. 80% dekning) • Eventuell regresjonstester er utført
• Ingen vesentlige feil
• Eventuelle PBI-akseptansetester er utført

Øvrig administrasjon

- **Discord** brukes som primær digital kommunikasjonsplattform for internbruk.
- **Teams** brukes for arrangering av digitale møter med skogkurs og prosjektets interessenter.
- **Epost møteinnkallinger** brukes for å organisere tidspunkt og lokasjon for møter/fellesaktiviteter.
- **Clockify** brukes til registrering av timer

Følgende kategorier er definert i Clockify for timeføring:

Tabell 5.1.2.3, Clockify-kategorier

Kategori	**Eksempler**
Dokumentasjon	• Kode-diagrammer • Vedlegg til bachelorrapporten hvorav prosjektplan, loggbok, ol.
Egenlæring	• Tilegnelse av kunnskap om metodikk og bruk av teknologi, inkl. språk og biblioteker

Koding	• Kildekode med kommentering
Møter	• Gruppediskusjon og planlagte møter
Rapport	• Skriving av bacheloroppgave
Research	• Undersøke teknologivalg, biblioteker, ol.

Organisering	• Organisering rundt prosjektet, som f.eks
	  ○ møteinkallelser
	  ○ planlegging av møter
	  ○ utforming av prosjektplan
	  ○ øvrig prosjektstyring

5.1.3 Kodeversjonering

- **GitHub** benyttes som repository. Dette forsikrer enkel overføring av driftsansvar ved endt prosjekt.
  - https://github.com/oysteinqvigstad/SKOG-kostnadskalkulator
- **Trunk-based** "branching"-strategi foretrekkes for å sikre kontinuerlig integrasjon og testing.
  - Feature-branches tillates i korte perioder ved behov for større strukturelle endringer eller eksperimentering, men avgrenses til maksimalt 1 ukes levetid.
  - Utviklere jobber direkte på main branch ("trunk") samt nevnte korte feature branches.
- **GitHub Actions** benyttes som **CI/CD** system.
  - Pipeline skal bygge koden og verifisere automatiske kodetester.
  - Automatisk deployment er høyst aktuelt, men faktiske krav er avhengig av hvilken tjeneste og ansvarsområde som foreligger for videre drift.

Figur 5.1.3, eksempel-diagram på Git Flow-branching, fra "Trunk-based Development vs. Git Flow", av K. Gadzinowski, u.å. (https://www.toptal.com/software/trunk-based-development-git-flow)

5.1.4 Ressursbehov

- **CI/CD-Runnere** i utviklingsfasen (NTNUs SkyHigh-platform).
  - Ved endt prosjekt kan man benytte gratiskvote som tilbys for Githubs egne CI/CD-runnere.
  - Standard bachelorkvote i SkyHigh er 32 vCPU, 64 GB RAM, 300 GB volumlagring.
- Serverless arkitektur
  - **PaaS-tjeneste** for *hosting* av API web service og statiske filer.
    - Tjenesten bør helst støtte node.js direkte og tilby GitHub Actions SDK for enkel CD-pipeline
  - **BaaS-tjeneste** for *hosting* av database
  - Skogkurs har pr i dag ikke dialog med aktuelle partnere eller tjenesteleverandører for utrulling av tjenesten vår.
    - Vi er ikke avhengig av å ta en umiddelbart beslutning av tjenesteleverandør.
    - Skogkurs besitter blant annet en Google-konto for selskapet og et Azure-abonnement, og er åpen for å betale lisenskostnader.
- **Subdomene** med tilhørende TLS-sertifikat.
  - Skogkurs oppretter og administrerer dette for oss gjennom http://domeneshop.no .

5.1.5 Teknologianvendelse

- **Typescript** brukes som primært programmeringsspråk i både frontend og backend.
    - Årsaken er behov for deling av kode knyttet til formelbehandling.
    - **Node.js 20.x** (siste LTS) brukes som runtime- og pakkesystem.
- **Frontend** avhengigheter/rammeverk:
    - **Bootstrap** brukes som CSS rammeverk
    - **React** brukes som frontend komponentbibliotek
    - **Redux** brukes for state management
    - **Rete** brukes for visualisering av matematisk formel i administrasjonskonsollet
- **Backend** avhengigheter/rammeverk
    - **Express** brukes som web-service rammeverk
- **PaaS/BaaS**
    - **Firebase** eller **MongoDB** brukes som database
    - **Google App Engine** brukes som vertsplatform i utviklingsfasen.

5.2 Plan for inspeksjoner og testing

5.2.1 Testing av kode

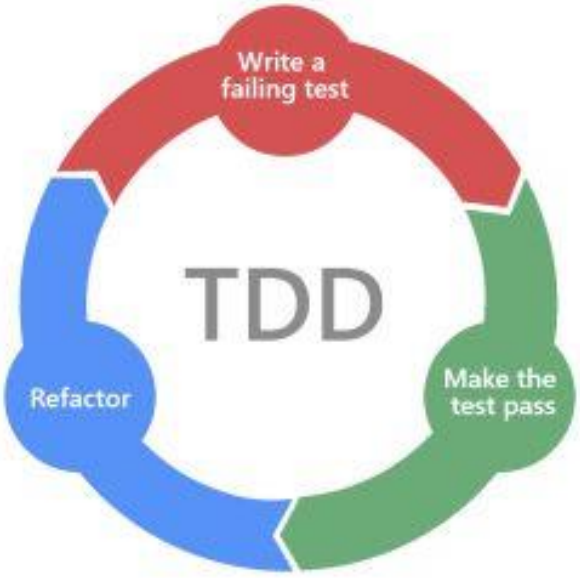
Hva bør enhetstestes?

- Fokuser på tester som verifiserer at funksjonalitet etterlever funksjonelle og operasjonelle krav
- Legg mer vekt på funksjoner som implementerer applikasjon eller forretningslogikk
- Legg mindre vekt på tester som validerer presentasjon av tilstand
- Resultat fremfor rigid testing: Å lære seg Test Driven Development (se figur 5.2.1) er en prosess. Er TDD et stort hinder der og da, vurder å legg det til siden til fordel for tester i etterkant.

Tabell 5.2.1, beskrivelse av teamets TDD-prosess.

Steg	Beskrivelse
1	Velg ut en liten bit av funksjonalitet som skal implementeres
2	Skriv tester samtidig som du velger ut parameter og returtype for funksjon/metode • Utform testene i tråd med akseptkriterier
3	Kjør tester og bekreft at de nye testene er de eneste som feiler
4	Skriv den minste mengde kode som lar funksjonen bestå testene
5	**Refaktoriser:** Jobb videre iterativt på kode og tester om noe kan forbedres
6	Gjenta

Den iterative testskriving->implementeringssyklusen bør være så kort som mulig. Minutter fremfor timer.



Figur 5.2.1, illustrasjon av TDD-prosess, fra "Why Test-Driven Development (TDD)", u.å. (https://marsner.com/blog/why-test-driven-development-tdd/)

Akseptkriterier

For å lettere å kunne vite hva som bør testes bør PBIs ha en liste over kriterier som må utfylles for at noe skal møte DoD. På denne måten finnes alt av kravspek i Backlogen slik det skal etter Scrum metodikken. DoR, DoD og Akseptkriterier vil da fungere sammen med tilhørende tester for å sikre kvaliteten på produkt.

3-5 kriterier per story foreslås som et greit antal. Er det behov for fler bør man kanskje vurdere å dele opp PBI videre (Hutchison, 2018, "What Characteristics Should Be Included and Not Included", avsn. 2).

5.2.2 Par-programmering

- Det ønskes at medlemmer settes sammen i par for hver sprint
- Medlemmene i hvert par har ansvar for tildelte PBI-er, fra planlegging til sluttføring.
- Hvert par diskuterer tildelte oppgaver og oppfordres til å kode iht. "par-programmering".

5.2.3 UX/UI: Testing og utforming

Prototyping

- Paul Schiager leder arbeid med å utvikle interaktive prototyper av design

Brukertesting

- Eventuelle prototyper deles med interessenter i møteinvitasjon for kontinuerlig feedback

Tabell 5.2.2, foreløpig brukertestplan

Produkt	Dato	Varighet	Hva skal testes?	Brukergruppe

Design Prototype 1.0	07.02	~15 min	• Grensesnitt
			  ○ Lett navigering
			  ▪ Side til side
			  ▪ Mellom parametre og dashboard
			  ○ Kategorisering
			  ○ Lett å finne info
				• Brede
				• Even
				• Interessenter (Glommen/Mjøsen, Allskog, MEF)
				• Merete (UX-designer)

MVP	21.02	~30 min pr. testsubjekt	• Feltapplikasjon
			  ○ Tilbakemelding på endringer fra prototyp 1.0
			  ○ Eventuelt ny-implementert funksjonalitet?
			• Avslutningssamtale
			  ○ Hva trenges for å gå over fra Excel-ark / begynne å bruke kalkulator
				• Skogdriftsledere
				• Entreprenører
				• Skogeiere?

5.3 Risikoanalyse på prosjektnivå

Denne risikoanalysen dreier seg i hovedsak om risikoer knyttet til prosjektgjennomføringsprosessen, og i mindre eller ingen grad om sikkerhetsrisikoer ved sluttproduktet. Disse vil bli behandlet i forbindelse med programvarens kravspesifikasjoner.

I arbeidet med risikoanalysen har vi valgt å formulere tiltak for de hendelsene med samlet risikonivå medium eller høyere. Enkelte av de vurderte hendelsene er derfor analysert i henhold til sannsynlighet og alvorlighetsgrad, men ikke gitt tiltak fordi det totale risikonivået er ansett som lavt.

Risikoer vurderes etter tre kriterier:

- **Alvorlighetsgrad (A)**: hvor alvorlig er det hvis risikoen inntreffer?
- **Sannsynlighet (S)**: hvor sannsynlig er det at en risiko inntreffer?
- **Samlet risikonivå (R)**: produktet av alvorlighetsgrad og sannsynlighet.

Hvert av kriteriene blir så vektlagt til **Høy (H)**, **Medium (M)** eller **Lav (L)**, ved hjelp av en forenklet Risk Poker metodikk (Risk Poker, u.å)

- Sikkerhetsansvarlige og eventuelle andre team-medlemmer går sammen og gir sine inviduelle vurderinger av hvert kriteriums vektnivå.
- Har alle de involverte gitt sammen vurdering blir denne satt som endelig; hvis ikke drøftes risikoen til en konsesus blir nådd.
- Fokus ligger på at teamet skal ha en felles forståelse av hva en risiko innebærer, fremfor å komme fram til en "objektivt" riktig vurdering.

Tabell 5.3.1, viser hvordan samlet risikonivå blir beregnet på bakgrunn av sannsynlighet og alvorlighetsgrad

Sannsynlighet			
Høy	Medium	Høy	Kritisk
Medium	Lav	Medium	Høy
Lav	Lav	Lav	Medium
Alvorlighetsgrad	**Lav**	**Medium**	**Høy**

Basert på forelesnings-slides fra prog1004

Tabell 5.3.2, prosjektgrupperisikoer

Prosjektgrupperisiko					
Beskrivelse	**Konsekvens**	**A**	**S**	**R**	**Tiltak**
Sykdom som påvirker et medlems arbeidsevne over lengre tid.	Gruppens totale arbeidskapasitet reduseres med 25% eller mer i et betydelig tidsrom	H	L	M	• Omfordele oppgaver • Vurdere reduksjon av prosjektets omfang • Holde seg hjemme om man er smittsom
Gruppemedlem med spesielt mye kunnskap om en viktig del av prosjekt blir indisponert.	En eller flere sprint kan bli alvorlig bremset ned som følge av kunskapsmangel.	H	L	M	• Tirsdager blir det gjort intern sprint-review med kunnskapsdeling • Diagrammer over kode underveis.

Redusert kapasitet hos oppdragsgiver, veileder eller annen ekstern kontaktperson	Betydelig reduksjon i oppfølging og avklaringer nødvendige for prosjektets fremdrift	H	L	M	• Avklar fremdriftskritiske spørsmål tidlig • Avklar stedfortredere der mulig
Intern konflikt i gruppen	Tidkrevende konfliktresolusjon, dårlig trivsel, redusert samarbeidsevne	H	L	M	• Tydelige grupperegler • Utnevn prosjektleder med vetorett/dobbelstemme

Tabell 5.3.3, forretningsrisikoer

Forretningsrisiko					
Beskrivelse	**Konsekvens**	**A**	**S**	**R**	**Tiltak**
Uavklart prosjektomfang/ mangelfull prioritering av funksjonalitet fra PO	Utviklingsarbeid kan bli utsatt eller blokkert. Kan bli mye fokus på ting som ikke er viktig slik at man får mange halvveis ferdigstilte produktfunksjonaliteter.	M	M	M	• Sørg for at overhengende foretningsmål er etablert som pekepinn for oppretning av PBI i fravær av PO. • Benytte smidig metodikk for å sikre iterativ utvikling • Sette deadline for når ønsker om ny ønsket funksjonalitet tas imot.
Produktet ferdigstilles ikke i løpet av prosjektperioden	Dårlig utgangspunkt for rapport, eventuelt utfordrende å viderearbeide for arbeidsgiver.	M	L	L	• Sørg for ryddig strukur og dokumentering underveis
Dårlig tilgang til brukere for brukertesting	Manglende grunnlag for å vurdere om produktet løser problemstillingen.	M	L	L	• Vurder alternativer til bruker-tester • Vurder alternative testgrupper som kan gi tilnærmelig mye verdi.

Tabell 5.3.4, Teknologirisikoer

Teknologirisiko					
Beskrivelse	**Konsekvens**	**A**	**S**	**R**	**Tiltak**
Tredjeparts-biblioteker/rammeverk/verktøy blir utilgjengelig eller ubrukelige	Hele eller deler av kodebase kan bli ubrukelig. Verktøy som brukes for work management kan bli utilgjengelig, som påvirker arbeidsflyt og prosjektstyring.	H	L	M	• Bruke rammeverk og verktøy fra anerkjente leverandører • Vurdere flere alternativer • Ta lokale backuper av kritiske verktøy/bruk version-pinning
Tredjeparts-biblioteker introduserer sikkerhetsrisiko eller feil.	Prosjektet arver svakheter fra øvrig bibliotek, som i sin tur kan føre til alvorlig sikkerhetsrisiko eller stabilitetsproblemer.	H	M	H	• Sørg for at sikkerhetsadvarsler, eksempelvis i NPM, ikke ignoreres, men at en dokumentert vurdering blir gjort per advarsel. Sikkerhetsvurderinger
Tjenester blir utilgjengelig eller ubrukelige kortsiktig	Tjenesten kan ikke kjøre som normalt	M	L	L	• Sørg for at standardversjone av kritiske datastrukturer, som f.eks formelstruktur, lagres/caches på server som leverer app kode, og kan caches av klient.
Tjenester blir utilgjengelig eller ubrukelige langsiktig	Tjenesten vil kunne helt eller delvis slutte å fungere.	H	L	M	• Skriv kode slik at prosjektet ikke er 100% knyttet til en spesifikk implementasjon av eksterne tjenester, hved hjelp av f.eks interfaces og dependency injection.

Tap av kildekode og/eller dokumentasjon	Deler eller alt utført arbeide blir borte, som kan sterkt påvirke eller forhindre gjennomføring av prosjektet.	H	L	M	• All kode/dokumentasjon lagres i repo/cloud • Backup av kode i repo hos annen leverandør • Backup av dokumentasjon hos alternativ cloud-leverandør.
Ønsket funksjonalitet lar seg ikke implementere (innenfor tidsrammene?) ved hjelp av utvalgte teknologier eller verktøy	Produktet møter ikke oppdragsgivers krav (eller dekker ikke deres behov)	M	M	M	• Vurdere endring av omfang • Vurder alternative teknologier eller verktøy

5.4 Sikkerhetsstrategi

Sikkerhet er en prioritet for programvaren på lik linje med for eksempel brukervennlighet og ytelsesoptimalisering, og programvaren skal utvikles etter "Build security in"-prinsippet: sikkerhet skal være i fokus fra første stund.

På et overordnet plan skal man ta hensyn til følgende sikkerhetsmål når programvaren utformes og implementeres, både hva angår design, arkitektur og utrulling:

- Data og programvare sikres for å ivareta CIA og AAA (konfidensialitet, integritet, tilgjengelighet, autentisering, autorisering og ansvarlighet).
- Identifiser sikkerhetskrav så tidlig som mulig med særlig hensyn til bruker-autentisering, session management, access-kontroll, kommunikasjonssikkerhet, kryptering, osv.
- Når programvaren oppdateres eller utvides skal potensielle nye sårbarheter identifiseres og elimineres.
- Unngå kodeteknikker som resulterer i tekniske sårbarheter (se kriterier for sikker koding under).
- Korrekt og sikker utrulling av programvaren. Programvare skal være testet før den gjøres tilgjengelig for bruker.

I utviklingsarbeidet skal team-medlemmene sikre at følgende kriterier/hensyn er møtt før kode committes til repository:

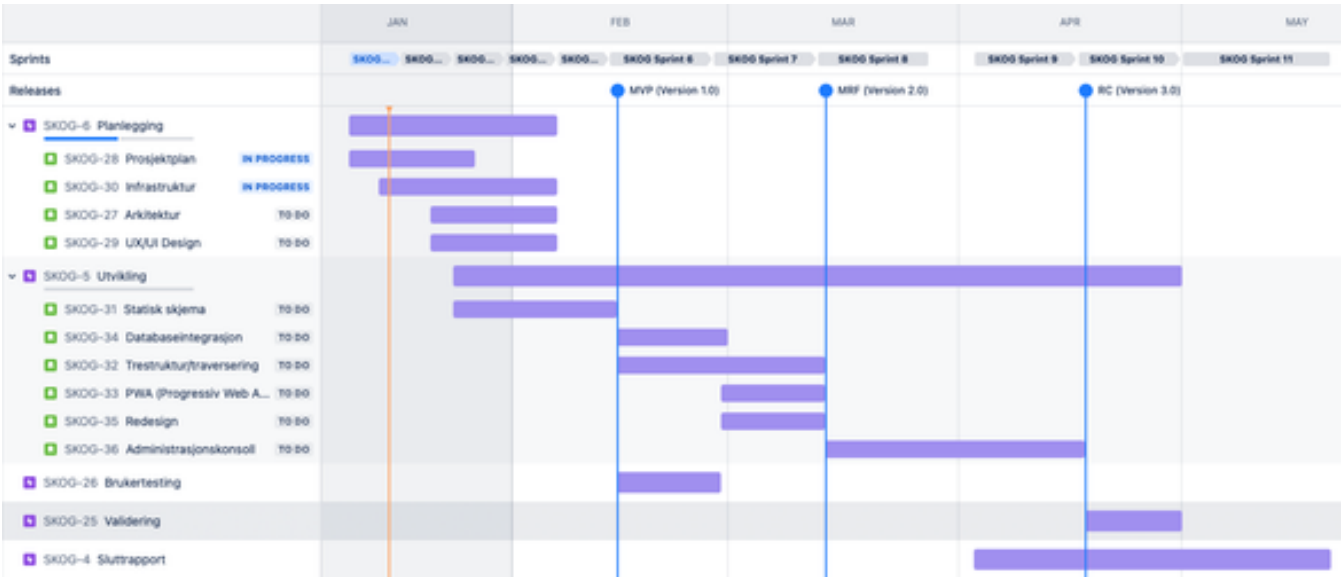
- Bruk kjente og vedlikeholdte bibliotek framfor å lage alt fra bunnen. Vurder alltid troverdigheten til tredjepartsprogramvare.
- Valider all data fra usikre kilder (databaser, filer, bruker-input, o.l.) mot forventet type, intervall, størrelse og tillatte tegn. All uvalidert data skal avvises.

- Ikke la applikasjonen sende kommandoer direkte til OS, men benytt det enkelte språks eller rammeverks innebygde APIer for OS-oppgaver.
- Unngå race-conditions i database-kall, eller forespørsler til andre delte ressurser, ved å bruke synkroniseringsmekanismer som f.eks. mutex.
- Initialiser variabler og data stores, enten ved deklarasjon eller like før første bruk. Vær bevisst på typebruk, og hvordan typer representeres i språket, særlig med tanke på numeriske typer (casting, truncation, int-overflow, precision, signed vs unsigned osv.).
- Hvis applikasjonens privilegienivå må heves, hev disse så sent som mulig, og senk dem så raskt som mulig (Principle of least privilege).

5.4.2 Håndtering av hemmeligheter

- Hemmeligheter, som passord, konfigurasjonsfiler, kredentialer, o.l., skal aldri ligge i koden, og aldri pushes til git-repo.
- Ettersom oppdragsgivers infrastruktur og deployment-behov avdekkes skal det utarbeides en mer robust strategi for å håndtere hemmeligheter.

6. Fremdriftsplan for Gjennomføring



Figur 6, Gantt-chart for prosjektet

Notater til fremdriftsplanen

- Flere av oppgavene vil gå parallelt gjennom hele prosjektperioden, eksempelvis vil det alltid være fokus på sluttrappportskriving. Derfor må Gantt-chartet tolkes slik at periode-inndelingen indikerer hovedfokus i gitt perioder, snarere enn utelukkende fokus.
- Disse periodene er veiledende: det er forventet og ønskelig at milepæler, datoer og oppgaveinndeling kan justeres etter behov.

Tabell 6, milepæler.

	MVP	MRF	RC
Dato	14 Feb 2024	13 Mar 2024	17 Apr 2024
Versjon	v1.0	v2.0	v3.0
Funksjonalitet	• Gjenskapelse av nåværende kostnadsformel • UX/UI Design v1	• Dynamisk formel-innlesning • Integrasjon av database • Brukertesting med oppdatert UX/UI Design	• Administrasjons-konsoll • Utrulling av nye formler

7. Terminologi

Tabell 7, begrepsforklaring.

Begrep	Beskrivelse
PO - Product Owner	Representant for organisasjoner som har eierskap i prosjektet. Koordinerer med Scrum Master og øvrige utviklere for å klargjøre prosjektets mål, utarbeide PBI samt prioritering av disse.
SM - Scrum Master	Koordinator for utviklingsteamet. Fasliliterer kommunikasjon mellom utviklere og produkteier/interessenter. Leder og organiserer scrum-aktiviteter, som PO-møte, sprint-planlegging osv.
Backlog	Prioritert liste med arbeidsoppgaver for utviklingsteamet, som beskriver hva som må implementeres for å forbedre produktet. Oppdeles i: • Sprint backlog - Oppgaver som jobbes med i nåværende sprint • Produkt backlog - oppgaver som venter for fremtidige sprints
Issueboard	Visualiseringsverktøy for issues. Presenterer issuene på en tavle, som forteller hvilken sprint issuet tilhører, hvilken status det har (påbegynt, under vurdering, ferdigstilt), tidsestimat for gjennomføring, m.m.
Issue	Konkret oppgave som utgjør hele eller deler av et PBI
PBI - Product Backlog Item	Et element i Backlog som beskriver et stykke arbeid som skal utføres. Jo høyere på listen, jo mer granulært bør PBI være. Se **DOD** og **DOR**

Sprint	Én utviklingssyklus, som oftest én eller to uker lang. Utviklingsteamet velger PBI-er fra backloggen og forsøker å implementere disse. Ved endt sprint vurderes status til disse, og om de ikke er ferdigstilt blir de som regel del av neste sprint.
DOD - Definition of Done	Et sett med karaktertrekk et PBI må møte for å kunne sies å være ferdigstilt. Møter PBI ikke DOD legges den tilbake i Backlog ved sprintens slutt.
DOR - Definition of Ready	Et sett med karaktertrekk et PBI må møte for å vøre klar til å trekkes inn i en sprint. Møter PBI ikke DOR, men har høy prioritet, må det bearbeides videre til det møter DOR.

8. Referanseliste

8.1 Nettsider

- Hutchison, Charles. (5. desember 2018). *Blog: Successful Scrum Acceptance Critera*. Scrum Adventures. Hentet 19/1-2024 frahttps://scrumadventures.com/successful-scrum-acceptance-criteria/
- Schwaber, Ken & Sutherland, Jeff. (2020). *The Scrum Guide*. Scrum Guides. Hentet 29/1-2024 fra https://scrumguides.org/scrum-guide.html
- *Skogkurs*. Skogkurs. Hentet 29/1-2024 fra https://skogkurs.no/
- *Risk Poker*. TMAP. Hentet 29/1-2024 frahttps://www.tmap.net/wiki/risk-poker

8.2 Bilder og figurer

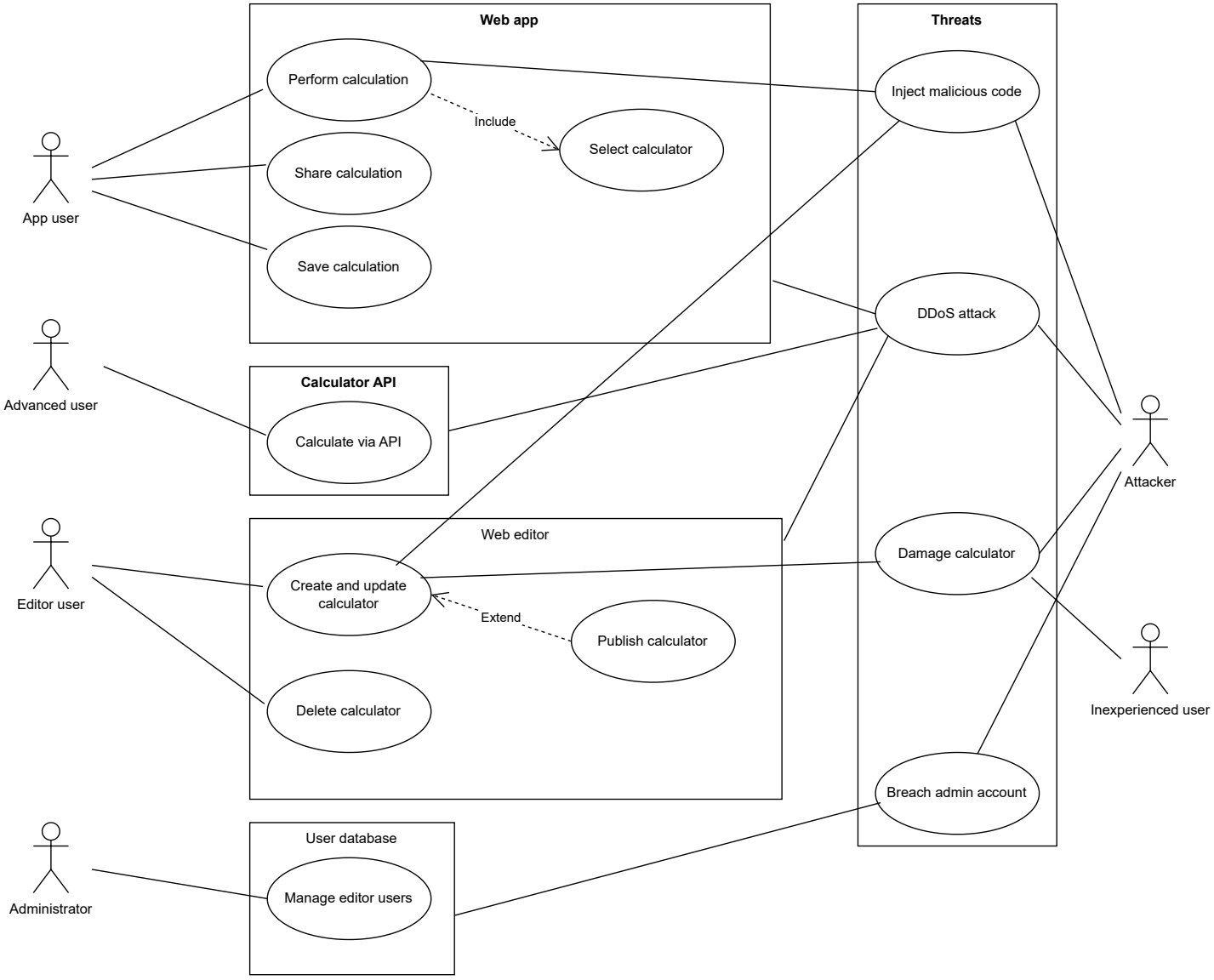
- Gudzianowski, Konrad. *Trunk-based Development vs. Git Flow*. Toptal. https://www.toptal.com/software/trunk-based-development-git-flow
- *Why Test-Driven Development*. Marsner.https://marsner.com/blog/why-test-driven-development-tdd/

Appendix F

Use Case Diagram

This document contains the use case diagram, complete with descriptions of all use/abuse cases.

Use case/abuse case diagram



Use case	Perform calculation
Actor	App user
Goal	Calculate cost of forestry operations
Description	App user fills in own values in some or all input fields. Main results are viewed in real time. More comprehensive results are viewed on user's request.

Use case	Share calculation
Actor	App user
Goal	Share calculation with another person
Description	App user clicks "share" button. A system specific share menu appears, and lets user send calculation as e,g. e-mail to another person.

Use case	Save calculation
Actor	App user
Goal	Save calculations for later retrieval
Description	The app user clicks the "save" button. In the corresponding menu the user may save a calculations with a identifying name. The user may also retrieve and delete from a list of previously saved calculations.

Use case	Select calculator
Actor	App user
Goal	Select the appropriate calculator for a specific need
Description	App user may choose among all available calculators. Calculators should have descriptive names for the user to select the best suited one.

Use case	Calculate via API
Actor	Advanced user
Goal	Perform calculation using an external system
Description	User utilizes https requests containing calculator version and all necessary input data to perform calculation. Results are returned in a machine readable format.

Use case	Delete calculator
Actor	Editor user
Goal	Delete existing calculator
Description	Via a calculator editor, the Editor user can delete an existing calculator. Editor operations require that the user is logged in.

Use case	Create and update calculator
Actor	Editor user
Goal	Create new and update existing calculator
Description	Via a calculator editor, the Editor user can create a new calculator. A set of math operations, input types and result displays are available. Editor user may choose to save calculator, or save and publish. For previously saved calculators, Editor user may apply changes and updates. At update, the calculator should be given a new version number. All Editor user operations require that the user is logged in.

Use case	Manage editor users
Actor	Administrator
Goal	Add or remove editor users for access to calculator database
Description	Administrator may add new or remove editor users for the calculator editor.

Abuse case	Inject malicious code
Actor	Attacker
Goal	Run malicious code in the app
Description	Attacker may try to inject scripts or other malicious code through input fields.

Abuse case	DDoS attack
Actor	Attacker
Goal	Make service unavailable
Description	Attacker may send massive amounts of requests to the app to iverload servers and make service unavailable to intended users.

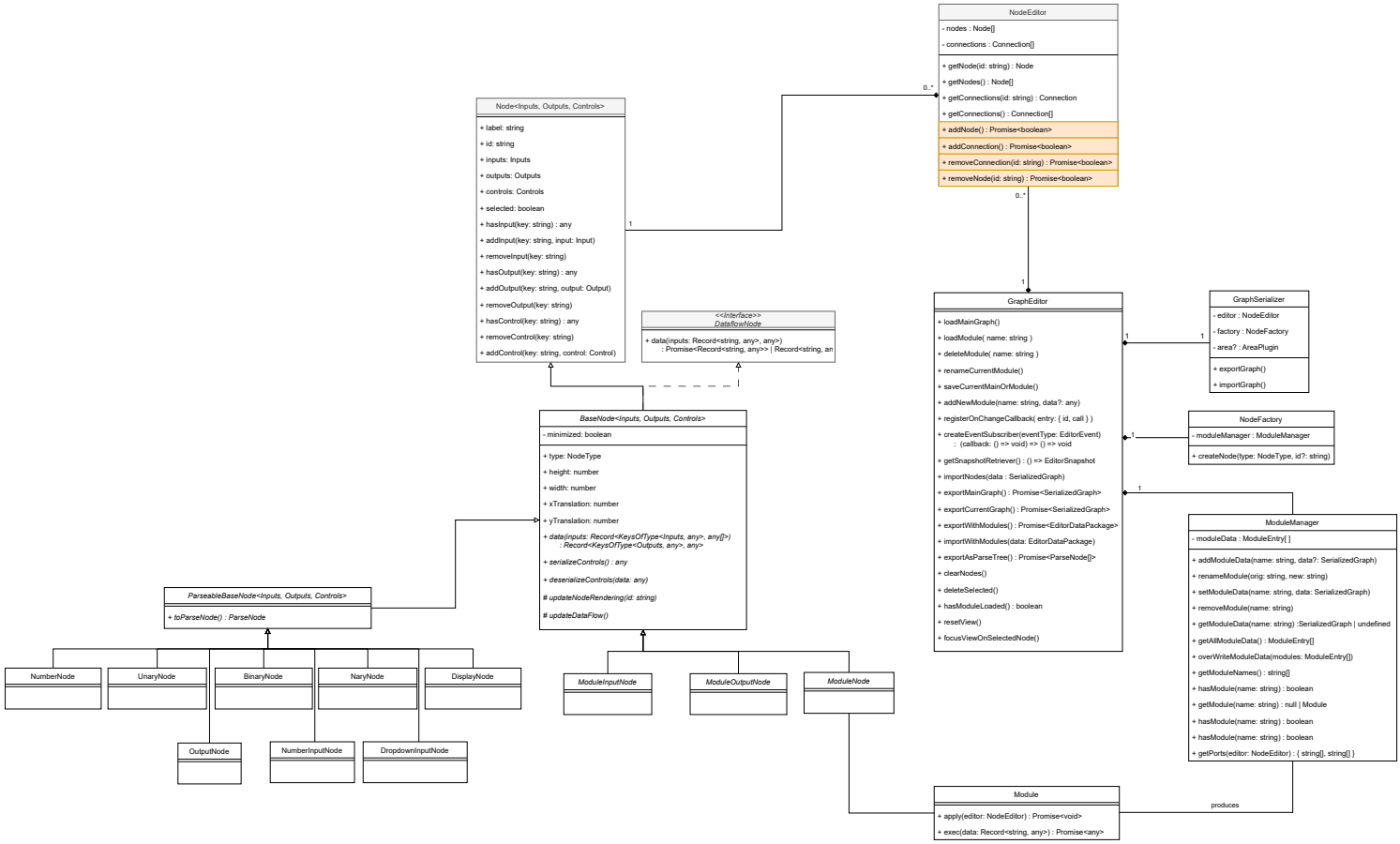
Abuse case	Damage calculator
Actor	Attacker, inexperienced user
Goal	Overwrite stored calculators
Description	Attacker may get write access in calculator editor client, and change existing calculators on purpose to make them erroneous or unusable. An inexperienced user may overwrite a previously stored calculator by accident.

Abuse case	Breach admin account
Actor	Attacker
Goal	Get control over user database
Description	Attacker may get access to admin account, thereby the ability to add or remove approved users. Legitimate users can be banned, an malicious users can be added.

Appendix G

Editor Class Diagram

This document contains a somewhat simplified class diagram. Some private properties and methods have been removed from the diagram for the purpose of brevity, while public ones have been kept intact.


NodeEditor
- nodes : Node[]
- connections : Connection[]
+ getNode(id: string) : Node
+ getNodes() : Node[]
+ getConnections(id: string) : Connection
+ getConnections() : Connection[]
+ addNode() : Promise<boolean>
+ addConnection() : Promise<boolean>
+ removeConnection(id: string) : Promise<boolean>
+ removeNode(id: string) : Promise<boolean>
0..*
0..*
Node<Inputs, Outputs, Controls>
+ label: string
+ id: string
+ inputs: Inputs
+ outputs: Outputs
+ controls: Controls
+ selected: boolean
+ hasInput(key: string) : any
+ addInput(key: string, input: Input)
+ removeInput(key: string)
+ hasOutput(key: string) : any
+ addOutput(key: string, output: Output)
+ removeOutput(key: string)
+ hasControl(key: string) : any
+ removeControl(key: string)
+ addControl(key: string, control: Control)
1
1
<<Interface>>
DataflowNode
+ data(inputs: Record<string, any>, any>)
: Promise<Record<string, any>> | Record<string, an
GraphEditor
+ loadMainGraph()
+ loadModule(name: string)
+ deleteModule(name: string)
+ renameCurrentModule()
+ saveCurrentMainOrModule()
+ addNewModule(name: string, data?: any)
+ registerOnChangeCallback(entry: { id, call })
+ createEventSubscriber(eventType: EditorEvent)
: (callback: () => void) => () => void
+ getSnapshotRetriever() : () => EditorSnapshot
+ importNodes(data : SerializedGraph)
+ exportMainGraph() : Promise<SerializedGraph>
+ exportCurrentGraph() : Promise<SerializedGraph>
+ exportWithModules() : Promise<EditorDataPackage>
+ importWithModules(data: EditorDataPackage)
+ exportAsParseTree() : Promise<ParseNode[]>
+ clearNodes()
+ deleteSelected()
+ hasModuleLoaded() : boolean
+ resetView()
+ focusViewOnSelectedNode()
1
1
GraphSerializer
- editor : NodeEditor
- factory : NodeFactory
- area? : AreaPlugin
+ exportGraph()
+ importGraph()
1
1
NodeFactory
- moduleManager : ModuleManager
+ createNode(type: NodeType, id?: string)
1
1
BaseNode<Inputs, Outputs, Controls>
- minimized: boolean
+ type: NodeType
+ height: number
+ width: number
+ xTranslation: number
+ yTranslation: number
+ data(inputs: Record<KeysOfType<Inputs, any>, any[]>)
: Record<KeysOfType<Outputs, any>, any>
+ serializeControls() : any
+ deserializeControls(data: any)
# updateNodeRendering(id: string)
# updateDataFlow()
ParseableBaseNode<Inputs, Outputs, Controls>
+ toParseNode() : ParseNode
ModuleManager
- moduleData : ModuleEntry[ ]
+ addModuleData(name: string, data?: SerializedGraph)
+ renameModule(orig: string, new: string)
+ setModuleData(name: string, data: SerializedGraph)
+ removeModule(name: string)
+ getModuleData(name: string) :SerializedGraph | undefined
+ getAllModuleData() : ModuleEntry[]
+ overWriteModuleData(modules: ModuleEntry[])
+ getModuleNames() : string[]
+ hasModule(name: string) : boolean
+ getModule(name: string) : null | Module
+ hasModule(name: string) : boolean
+ hasModule(name: string) : boolean
+ getPorts(editor: NodeEditor) : { string[], string[] }
NumberNode
UnaryNode
BinaryNode
NaryNode
DisplayNode
OutputNode
NumberInputNode
DropdownInputNode
ModuleInputNode
ModuleOutputNode
ModuleNode
Module
+ apply(editor: NodeEditor) : Promise<void>
+ exec(data: Record<string, any>) : Promise<any>
produces

Appendix H

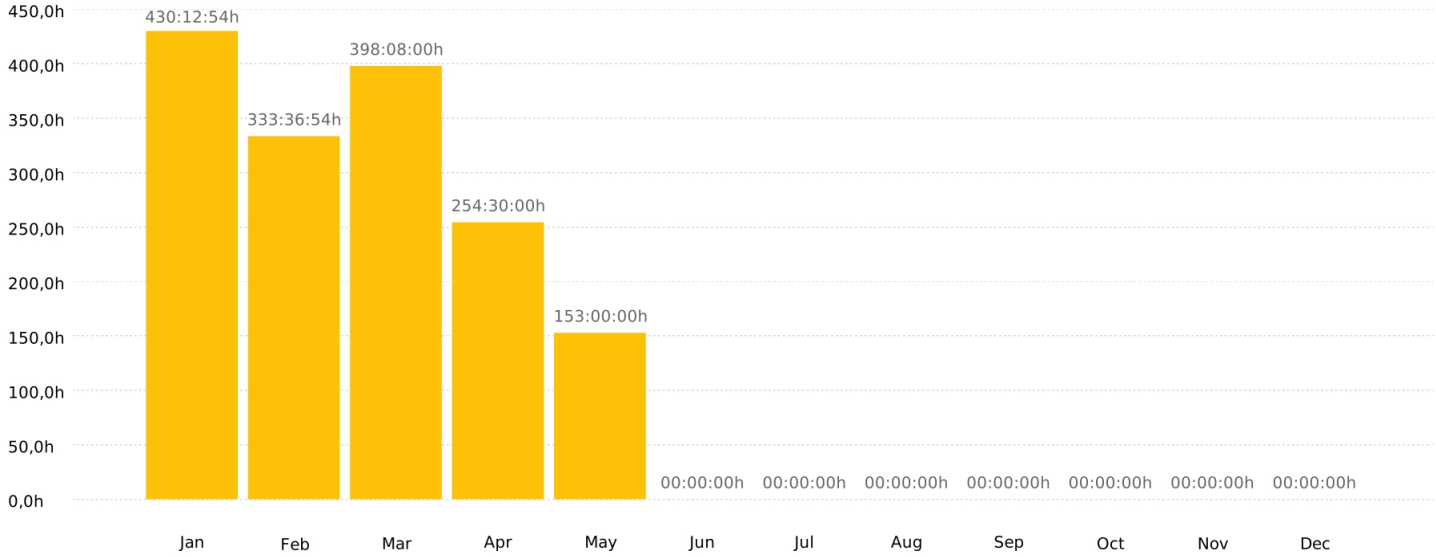
Time Report

This document contains summary graphs of the time recorded in Clockify, the time tracking software used for the project. It should be noted that time entries have been entered sporadically, leading to periods where not all time is accounted for, particularly towards the end of the project.

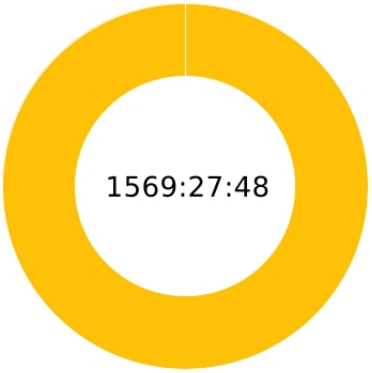
Time Report Skogkurs

01/01/2024 - 31/12/2024

Total: 1569:27:48 Billable: 1569:27:48 Amount: 0,00 USD

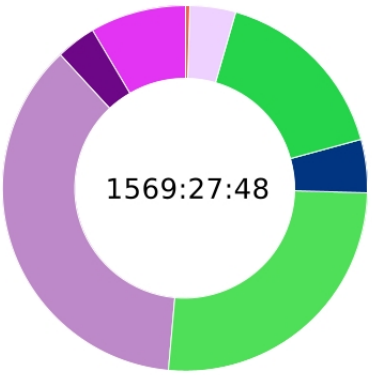


Project



Skogkurs	1569:27:48	100,00%

Task



Dokumentasjon - Skogkurs	132:03:00	8,41%
Egenlæring - Skogkurs	56:05:00	3,57%
Koding - Skogkurs	573:12:37	36,52%
Møter - Skogkurs	408:49:53	26,05%
Organisering - Skogkurs	73:30:00	4,68%
Rapport - Skogkurs	255:57:00	16,31%
Research - Skogkurs	63:20:18	4,04%

●	**UI design** - Skogkurs	06:30:00	0,41%

Project / Task	Amount
Skogkurs	0,00 USD
Dokumentasjon	0,00 USD
Egenlæring	0,00 USD
Koding	0,00 USD
Møter	0,00 USD
Organisering	0,00 USD
Rapport	0,00 USD
Research	0,00 USD
UI design	0,00 USD

Appendix I

Product and Sprint Backlogs

This document contains the Sprint Backlogs (SP) for Sprints 1 through 11. Occasionally, Product Backlog Items (PBIs) was carried over from the previous sprint, because it could not be completed within the time. As a result, these PBIs are listed in the sprint they were last moved to. Additionally, the Product Backlog includes PBIs with a status of 'Done' due to overlap with other PBIs or because they were worked on after all assigned PBIs for the sprint were completed, similar to Kanban.

Sprint Backlogs

Summary	Issue key	Issue Type	Assignee	Sprint	SP	Status
Som utviklere ønsker vi å beskrive prosjektets sikkerhetsstrategi	SKOG-12	Story	Paul Schiager	SKOG Sprint 01	10	Done
Som utviklere ønsker vi å beskrive prosjektets oppgavebeskrivelse	SKOG-16	Story	André Marthinsen	SKOG Sprint 01	12	Done
Som utviklere ønsker vi å analysere nåværende formel brukt i eksisterende kalkulator	SKOG-14	Story	André Marthinsen	SKOG Sprint 01	12	Done
Som scrum master ønsker jeg å utarbeide egen struktur/rutiner	SKOG-18	Story	Øystein Qvigstad	SKOG Sprint 01	14	Done
Som utviklere ønsker vi å beskrive kildekodens branchingstrategi og pipeline	SKOG-11	Story	Øystein Qvigstad	SKOG Sprint 01	6	Done
Som utviklere ønsker vi å beskrive prosjektets fagområde	SKOG-17	Story	Paul Schiager	SKOG Sprint 01	8	Done
Som utvikler ønsker vi å beskrive prosjektets bakgrunn og mål	SKOG-13	Story	Even Stetrud	SKOG Sprint 01	9	Done
Som utvikler ønsker vi å beskrive prosjektets mål	SKOG-37	Story	Even Stetrud	SKOG Sprint 02	1	Done
Som UX/UI designer ønsker jeg å opprette wireframe for webskjemaet	SKOG-46	Story	Paul Schiager	SKOG Sprint 02	2	Done
Som studenter ønsker vi å utforme en gruppekontrakt som skal signeres	SKOG-43	Story	Øystein Qvigstad	SKOG Sprint 02	2	Done
Som utviklere ønsker vi å beskrive prosjektets rammer og avgrensning	SKOG-21	Story	Even Stetrud	SKOG Sprint 02	2	Done
Som studenter ønsker vi at bacheloravtalen signeres	SKOG-20	Story	Øystein Qvigstad	SKOG Sprint 02	2	Done
Som utviklere ønsker vi å beskrive prosjektets testing og inspeksjonsstrategi	SKOG-42	Story	André Marthinsen	SKOG Sprint 02	4	Done
Som utviklere ønsker vi å beskrive prosjektets teknologianvendelse	SKOG-9	Story	Øystein Qvigstad	SKOG Sprint 02	8	Done
Som utviklere ønsker vi å beskrive prosjektets risiko	SKOG-8	Story	Paul Schiager	SKOG Sprint 02	8	Done
Som student ønsker jeg å reorganisere seksjoner til ny plass i prosjektplanen	SKOG-62	Documentation	Even Stetrud	SKOG Sprint 03	1	Done
Som studenter ønsker vi å konvertere prosjektplanen til Word og levere den inn	SKOG-60	Documentation	Even Stetrud	SKOG Sprint 03	1	Done
Som utvikler ønsker vi å definere en datastruktur for det statiske webskjemaet	SKOG-38	Documentation	André Marthinsen	SKOG Sprint 03	1	Done
Som student ønsker jeg å argumentere for ulike teknologivalg i sluttrapporten	SKOG-59	Documentation	André Marthinsen	SKOG Sprint 03	3	Done
Som utvikler ønsker jeg å skrive en klasse/funksjon for utregning av formel	SKOG-61	Story	André Marthinsen	SKOG Sprint 03	2	Done
Som sluttbruker ønsker jeg at variabler som skal oppgis er godt forklart	SKOG-49	Story	André Marthinsen	SKOG Sprint 03	2	Done
Som sluttbruker ønsker jeg å kunne skrive inn feltene iht. hogstmaskin	SKOG-63	Story	Øystein Qvigstad	SKOG Sprint 03	4	Done
Som sluttbruker ønsker jeg at skjemaet er brutt opp i sider fremfor å få alt på en gang	SKOG-52	Story	Even Stetrud	SKOG Sprint 03	4	Done
Som sluttbruker ønsker jeg å kunne se resultatene for hogstmaskin fra utregning	SKOG-47	Story	Paul Schiager	SKOG Sprint 03	4	Done
Som utrullingsansvarlig ønsker jeg å feilrette cachingmekanisme	SKOG-80	Bug	Øystein Qvigstad	SKOG Sprint 04	2	Done
Som student ønsker jeg å kartlegge behov/muligheter til sluttrapport	SKOG-90	Documentation	Even Stetrud	SKOG Sprint 04	2	Done
Som studenter ønsker vi å lage domenediagram til sluttrapporten	SKOG-89	Documentation	Øystein Qvigstad	SKOG Sprint 04	2	Done
Som utvikler ønsker jeg å researche browsersupport for caching/localstore/firebase og begrensninger	SKOG-83	Documentation	Øystein Qvigstad	SKOG Sprint 04	2	Done
Som studenter ønsker vi å lage et dokument for usecase-diagram med use-case for nåværende funksjonalitet	SKOG-81	Documentation	Even Stetrud	SKOG Sprint 04	2	Done
Som utvikler vil jeg utforme struktur på MVP brukertest	SKOG-79	Documentation	Paul Schiager	SKOG Sprint 04	2	Done
Som utvikler ønsker jeg å researche browsersupport for PWA	SKOG-82	Documentation	André Marthinsen	SKOG Sprint 04	3	Done
Som utvikler ønsker jeg å lage en wireframe på tab-navigering	SKOG-88	Documentation	Paul Schiager	SKOG Sprint 04	4	Done
Som sluttbruker ønsker jeg å kunne skrive inn feltene iht. lastbærer	SKOG-65	Story	Øystein Qvigstad	SKOG Sprint 04	4	Done
Som sluttbruker ønsker jeg å kunne se resultatene for lastbærer fra utregning	SKOG-64	Story	Øystein Qvigstad	SKOG Sprint 04	6	Done
Som produkteier ønsker jeg at tallgrunnlag på egen side	SKOG-102	Story	Øystein Qvigstad	SKOG Sprint 05	1	Done
Som produkteier ønsker jeg ikoner på sideindikatoren og evne. navn på siden	SKOG-99	Story	Paul Schiager	SKOG Sprint 05	1	Done
Som produkteier ønsker jeg å endre ordlyden i noen av feltene	SKOG-98	Story	Paul Schiager	SKOG Sprint 05	1	Done
Som produkteier ønsker jeg å endre grupperingen på sidene	SKOG-97	Story	Paul Schiager	SKOG Sprint 05	1	Done
Som produkteier ønsker jeg en forside på kalkulatoren	SKOG-96	Story	Even Stetrud	SKOG Sprint 05	2	Done
Som sluttbruker ønsker jeg å få feilmelding dersom utregningen ga ugyldig resultat	SKOG-87	Story	Øystein Qvigstad	SKOG Sprint 05	2	Done
Som utrullingsansvarlig ønsker jeg å researche hvordan secrets typisk håndteres i PaaS	SKOG-44	Documentation	Øystein Qvigstad	SKOG Sprint 06	1	Done
Som utvikler ønsker jeg å ferdigstille en mal for brukertesting	SKOG-111	Documentation	Paul Schiager	SKOG Sprint 06	2	Done
Som sluttbruker ønsker jeg at sideindikator blir rød hvis det er ugyldige feltverdier	SKOG-50	Story	Even Stetrud	SKOG Sprint 06	1	Done
Som utvikler ønsker jeg å flytte informasjon til egen sideindikator, og mer generell info på frontsiden	SKOG-119	Story	Even Stetrud	SKOG Sprint 06	2	Done
Som utvikler ønsker jeg å sette opp en forbindelse til API-endepunkt fra frontend	SKOG-117	Story	Øystein Qvigstad	SKOG Sprint 06	2	Done
Som utvikler ønsker jeg å sette opp et API-endepunkt med connection til database	SKOG-116	Story	Øystein Qvigstad	SKOG Sprint 06	2	Done
Som utviklere ønsker vi å utføre brukertesting av MVP	SKOG-114	Story	Paul Schiager	SKOG Sprint 06	2	Done
Som utvikler ønsker jeg å sette opp en dokumentdatabase	SKOG-113	Story	Øystein Qvigstad	SKOG Sprint 06	2	Done
Som utvikler ønsker jeg å forberede noen spørsmål til UX-designer (Merete)	SKOG-110	Story	Paul Schiager	SKOG Sprint 06	2	Done
Som produkteier ønsker jeg en tilbakeknapp i navbaren	SKOG-109	Story	Even Stetrud	SKOG Sprint 06	2	Done
Som utvikler ønsker jeg å kartlegge hva som skal til for at applikasjonen skal fungere med skjermleser	SKOG-95	Story	Even Stetrud	SKOG Sprint 06	2	Done
Som utvikler ønsker jeg å teste precaching/offline-funksjonalitet og PWA av frontendapplikasjon	SKOG-41	Story	Øystein Qvigstad	SKOG Sprint 06	2	Done
Som produkteier vil jeg at brukere skal se om et felt inneholder en standardverdi og ikke noe de har lagt til selv, og knapp for nullstilling	SKOG-85	Story	Øystein Qvigstad	SKOG Sprint 06	3	Done
Som utvikler ønsker jeg å definere en datastruktur for formel	SKOG-112	Story	André Marthinsen	SKOG Sprint 06	4	Done
Som student ønsker jeg å skrive om designprosessen i sluttrapporten	SKOG-130	Documentation	Paul Schiager	SKOG Sprint 07	1	Done
Som student ønsker jeg å skrive om utviklingsmetodikk i sluttrapporten	SKOG-131	Documentation	Even Stetrud	SKOG Sprint 07	4	Done
Som utvikler ønsker jeg å skrive rapport på brukertesting, og legge inn i sluttrapport	SKOG-129	Documentation	Paul Schiager	SKOG Sprint 07	8	Done

Summary	Issue key	Issue Type	Assignee	Sprint	SP	Status
Som utvikler ønsker jeg å opprette testnoder for dynamisk innlesning	SKOG-133	Story	André Marthinsen	SKOG Sprint 07	2	Done
Som UI-ansvarlig ønsker jeg skjemanavigeringslinje i bunnen (sticky) når viewport er liten, ellers i toppen.	SKOG-121	Story	Øystein Qvigstad	SKOG Sprint 07	2	Done
Som utvikler ønsker jeg å importere delte react-komponenter mellom klienter	SKOG-137	Story	Øystein Qvigstad	SKOG Sprint 07	3	Done
Som utvikler ønsker jeg å lage en prioriteringsliste over hva som kan dynamisk innleses	SKOG-135	Story	André Marthinsen	SKOG Sprint 07	4	Done
Som utvikler ønsker jeg å skissere UI wireframe for dynamisk kalkulator	SKOG-132	Story	André Marthinsen	SKOG Sprint 07	4	Done
Som sluttbruker ønsker jeg å kunne hente opp igjen resultatet på senere tidspunkt	SKOG-54	Story	Øystein Qvigstad	SKOG Sprint 07	4	Done
Som utvikler ønsker jeg å innlese testnoder for dynamisk innlesning	SKOG-134	Story	Øystein Qvigstad	SKOG Sprint 07	8	Done
Som student ønsker jeg å skrive om API-et i sluttrapporten	SKOG-143	Documentation	Øystein Qvigstad	SKOG Sprint 08	2	Done
Som student ønsker jeg å skrive deployment-kapittel i sluttrapport	SKOG-164	Documentation	Øystein Qvigstad	SKOG Sprint 08	3	Done
Som student ønsker jeg å drøfte bærekraft i sluttrapportavslutningen	SKOG-166	Documentation	Even Stetrud	SKOG Sprint 08	4	Done
Som utvikler ønsker jeg å skrive om admin-consollet i sluttrapporten	SKOG-154	Documentation	André Marthinsen	SKOG Sprint 08	4	Done
Som utvikler ønsker jeg å ferdigstille React-komponent for dropdown-noden i admin-consollet	SKOG-146	Story	Paul Schiager	SKOG Sprint 08	1	Done
som utvikler ønsker jeg komponent for å legge til html infotekst for noder	SKOG-177	Story	Even Stetrud	SKOG Sprint 08	2	Done
Som utvikler ønsker jeg at resultatsiden skal hente kalkulatorversjon som er definert i URL/lagring	SKOG-175	Story	Øystein Qvigstad	SKOG Sprint 08	2	Done
Som utvikler ønsker jeg å opprette API endpoints for versjonering og hente nyeste versjon	SKOG-173	Story	Øystein Qvigstad	SKOG Sprint 08	2	Done
Som utvikler ønsker jeg at redux oppdaterer TreeState når noe endrer seg i Rete	SKOG-172	Story	André Marthinsen	SKOG Sprint 08	2	Done
Som utvikler ønsker jeg å lage React-komponent for graf-display i admin-consollet	SKOG-151	Story	André Marthinsen	SKOG Sprint 08	2	Done
Som utvikler ønsker jeg å lage React-komponent for stolpe-display i admin-consollet	SKOG-147	Story	Øystein Qvigstad	SKOG Sprint 08	2	Done
Som utvikler ønsker jeg å ferdigstille React-komponent for number-input-noden i admin-consollet	SKOG-145	Story	Even Stetrud	SKOG Sprint 08	2	Done
Som utvikler ønsker jeg å opprette cachingmekanisme med service worker for API kall	SKOG-174	Story	Øystein Qvigstad	SKOG Sprint 08	3	Done
Som utvikler ønsker jeg å kunne velge rekkefølge på inputs innen en side	SKOG-162	Story	Paul Schiager	SKOG Sprint 08	4	Done
Som utvikler ønsker jeg moduler å gruppere undergrafer i	SKOG-160	Story	André Marthinsen	SKOG Sprint 08	4	Done
Som utvikler ønsker jeg å lage React-komponent for choose-noden i admin-consollet	SKOG-144	Story	André Marthinsen	SKOG Sprint 08	4	Done
Som driftsansvarlig ønsker jeg å migrere drift til Skogkurs	SKOG-127	Story	Øystein Qvigstad	SKOG Sprint 08	4	Done
Som student ønsker jeg å lage use case diagram til krav-spek	SKOG-191	Documentation	Paul Schiager	SKOG Sprint 09	2	Done
Som student ønsker jeg å skrive om frontend-implementasjon i rapporten	SKOG-190	Documentation	Øystein Qvigstad	SKOG Sprint 09	8	Done
Som utvikler vil jeg ha en forenklet text editor for å lage units for input og output	SKOG-180	Story	Even Stetrud	SKOG Sprint 09	1	Done
Som student ønsker jeg å flytte innhold fra den forrige Latex malen til den nye	SKOG-185	Story	Even Stetrud	SKOG Sprint 09	2	Done
Som utvikler ønsker jeg å lage en min / max node i editoren	SKOG-182	Story	Paul Schiager	SKOG Sprint 09	2	Done
Som utvikler ønsker jeg at bruker enten må lagre eller forkaste endringer gjort i infotext. Krysses vindu ut med endringer skal de få advarsel	SKOG-181	Story	Even Stetrud	SKOG Sprint 09	2	Done
Som utvikler ønsker jeg at vi kan definere number input som enten heltall eller desimaltall	SKOG-138	Story	Paul Schiager	SKOG Sprint 09	2	Done
Som utvikler ønsker jeg å kunne skrive info-tekst for resultatdisplayene	SKOG-189	Story	Paul Schiager	SKOG Sprint 09	3	Done
Som utvikler ønsker jeg å lage React-komponent for tabell-display i admin-consollet	SKOG-148	Story	Øystein Qvigstad	SKOG Sprint 09	3	Done
Som utvikler vil jeg fikse algoritme for å utvide moduler til fulle grafer	SKOG-188	Story	André Marthinsen	SKOG Sprint 09	4	Done
Som utvikler ønsker jeg å lage logikk for plassering/størrelse på visualisering for ulike størrelser	SKOG-171	Story	Øystein Qvigstad	SKOG Sprint 09	4	Done
Som student ønsker jeg å skrive om utviklingsprosessen	SKOG-196	Documentation	Even Stetrud	SKOG Sprint 10	3	Done
Som student ønsker jeg å skrive om trær i rapporten	SKOG-194	Documentation	André Marthinsen	SKOG Sprint 10	3	Done
Som utvikler ønsker jeg å skrive om PWA og service worker i rapporten	SKOG-195	Documentation	Øystein Qvigstad	SKOG Sprint 10	6	Done
Som utvikler ønsker jeg å gjenskape kalkulatoren i editoren	SKOG-212	Story	Øystein Qvigstad	SKOG Sprint 10	1	Done
Som utvikler ønsker jeg å endre fra objekt til string ved lagring av kalkulator	SKOG-192	Story	Øystein Qvigstad	SKOG Sprint 10	2	Done
Som student ønsker jeg å skrive om kode testing i sluttrapport	SKOG-211	Story	André Marthinsen	SKOG Sprint 10	3	Done
Som student ønsker jeg å lage et abuse-case-diagram	SKOG-251	Documentation	Paul Schiager	SKOG Sprint 11	1	Done
Som student ønsker jeg å oppdatere domenemodell	SKOG-252	Documentation	Øystein Qvigstad	SKOG Sprint 11	2	Done
Som student ønsker jeg å skrive om resultatene fra admin-consolldemonstrasjonbrukertest	SKOG-241	Documentation	Even Stetrud	SKOG Sprint 11	2	Done
Som student ønsker jeg å utforme enkle brukertestspørsmål til admin-consoll-demonstrasjonen	SKOG-240	Documentation	Even Stetrud	SKOG Sprint 11	2	Done
Som student ønsker jeg å drøfte KI-bruk (gruppas bruk av verktøy, maskinlæring for formelverk) i rapporten (kap 4)	SKOG-247	Documentation	Øystein Qvigstad	SKOG Sprint 11	4	Done
Som student ønsker jeg å drøfte alternative utforminger (dynamisk vs statisk, programmeringsspråk og platform (shinyapps), liknende applikasjoner)	SKOG-246	Documentation	André Marthinsen	SKOG Sprint 11	4	Done
Som student ønsker jeg å skrive en evaluering av gruppas arbeid i sluttrapportavslutningen	SKOG-168	Documentation	Even Stetrud	SKOG Sprint 11	4	Done
Som student ønsker jeg å drøfte resultatmål og produkt i rapporten (kap 4)	SKOG-243	Documentation	Paul Schiager	SKOG Sprint 11	1	In Progress
Som student ønsker jeg å drøfte effektmål i rapporten (kap 4)	SKOG-245	Documentation	Paul Schiager	SKOG Sprint 11	2	In Progress
Som student ønsker jeg å drøfte læringsmål i rapporten (kap 4)	SKOG-244	Documentation	Paul Schiager	SKOG Sprint 11	2	In Progress
Som student ønsker jeg å skrive innledningen i sluttrapporten	SKOG-155	Documentation	Even Stetrud	SKOG Sprint 11	3	In Progress
Som student ønsker jeg å skrive om videreutvikling av produktet i sluttrapportavslutningen	SKOG-169	Documentation	Paul Schiager	SKOG Sprint 11	3	In Progress
Som student ønsker jeg å skrive om bærekraft i rapporten (kap 4)	SKOG-250	Documentation	Even Stetrud	SKOG Sprint 11	4	In Progress
Som student ønsker jeg å skrive om sikkerhet i rapporten	SKOG-193	Documentation	Paul Schiager	SKOG Sprint 11	6	In Progress
Som student ønsker jeg å skrive krav-spek i sluttrapporten	SKOG-158	Documentation	Paul Schiager	SKOG Sprint 11	6	In Progress
Som student ønsker jeg å skrive abstract for rapporten	SKOG-242	Documentation	André Marthinsen	SKOG Sprint 11	2	To Do
Som student ønsker jeg å skrive om formelverket (Brunberg) i teorikapittelet	SKOG-239	Documentation	André Marthinsen	SKOG Sprint 11	2	To Do

Product Backlog

Summary	Issue key	Issue Type	Sprint	Status
Som bruker ønsker jeg info om G15 på resultatene	SKOG-142	Story	Backlog	Done
API /calculate/ endpoint gir feil resultat på hogstmaskin, antakeligvis en bug i dropdown-valgene	SKOG-238	Bug	Backlog	Done
Kategoriske kostnadsdrivere i graf-(display)noden har ikke "din verdi"	SKOG-231	Bug	Backlog	Done
Blank knapp appendes til graf-(display)noden ved endringer i treet, eller navigering mellom moduler. Får ikke fjernet siste blanke knapp	SKOG-229	Bug	Backlog	Done
Offcanvas lukker seg ikke når man navigerer til ny side	SKOG-105	Bug	Backlog	Done
Som student ønsker jeg å skrive om alternativer til vår løsning i admin konsoll	SKOG-94	Documentation	Backlog	Done
Som utviklere ønsker vi å lage usecase for administrasjon/vedlikehold av kalkulatoren	SKOG-128	Documentation	Backlog	Done
Som produkteier ønsker jeg synlig side om forskningsgrunnlag	SKOG-103	Story	Backlog	Done
Som utvikler ønsker jeg å lage en sqrt (samt. nth sqrt) node i editoren	SKOG-184	Story	Backlog	Done
Som bruker ønsker jeg å kunne lage link fra lagrede-resultat siden	SKOG-140	Story	Backlog	Done
Som bruker ønsker jeg å se produktivitet som søylediagram	SKOG-139	Story	Backlog	Done
Som utvikler ønsker jeg innloggingsmekanisme på admin-consollet	SKOG-136	Story	Backlog	Done
Som UI-ansvarlig ønsker jeg å sette en max-bredde på navbarinnhold	SKOG-126	Story	Backlog	Done
Som UI-ansvarlig ønsker jeg at at linker til forsknings- og tallgrunnlag som er i navbar pr i dag legges i infoside	SKOG-125	Story	Backlog	Done
Som UI-ansvarlig ønsker jeg at skjemanavigeringslinjen også har tekst	SKOG-124	Story	Backlog	Done
Som UI-ansvarlig ønsker jeg en infoside inne i selve skjemaet	SKOG-123	Story	Backlog	Done
Som UI-ansvarlig ønsker jeg å fjerne pilknapper i navigeringslinjen fordi det ikke er så mange sider	SKOG-122	Story	Backlog	Done
Som interessent ønsker jeg resultatet synlig når man taster inn verdi	SKOG-100	Story	Backlog	Done
Som produkteier ønsker jeg et API for å kalkulere med andre systemer	SKOG-78	Story	Backlog	Done
Som sluttbruker ønsker jeg visualisering av kostnadsdrivere på resultat	SKOG-56	Story	Backlog	Done
Som sluttbruker ønsker jeg å kunne få innblikk i hvordan enkelte kostnadsdrivere påvirker totalkostnaden	SKOG-48	Story	Backlog	Done
Som utviklere ønsker vi å lage en mer detaljert fremdriftsplan	SKOG-15	Story	Backlog	Done
Som utviklere ønsker vi å beskrive prosjektets ressursbehov	SKOG-10	Story	Backlog	Done
Som utvikler ønsker jeg å organisere tekster i internationalisation-bibliotek	SKOG-115	Story	Backlog	In Progress
Service Worker henter ikke kalkulatorer fra cache dersom handler returnerer 5xx feilkode	SKOG-237	Bug	Backlog	To Do
Kan ikke se alle inputfeltene for å sjekke rekkefølge dersom det er mange på samme side	SKOG-235	Bug	Backlog	To Do
Reset alle felter knapp er ikke implementert	SKOG-233	Bug	Backlog	To Do
Units-definisjoner (listen) blir nullstilt, hvorav alle forsvinner	SKOG-230	Bug	Backlog	To Do
Rekkefølge på display-noder blir i blant feil og må justeres og lagres på nytt	SKOG-228	Bug	Backlog	To Do
Rekkefølge på inputfelt blir feil - virker som endring av rekkefølge på én side også endrer rekkefølgen på en annen	SKOG-227	Bug	Backlog	To Do
Som student ønsker jeg å skrive en kritikk av oppgaven/produktet (hva kunne vært bedre, annerledes) i rapporten (kap 4)	SKOG-248	Documentation	Backlog	To Do
Som student ønsker jeg å skrive konklusjon i sluttrapportavslutningen	SKOG-170	Documentation	Backlog	To Do
Som student ønsker jeg å skrive en kritikk av bacheloroppgaven i sluttrapportavslutningen	SKOG-167	Documentation	Backlog	To Do
Som student ønsker jeg å drøfte prosjektresultatene i sluttrapportavslutningen	SKOG-165	Documentation	Backlog	To Do
Som student ønsker å skrive testing-kapittel i sluttrapport	SKOG-163	Documentation	Backlog	To Do
Som student ønsker jeg å skrive implementering-kapittel i sluttrapport	SKOG-161	Documentation	Backlog	To Do
Som student ønsker jeg å skrive design-kapittel (GUI og teknisk-utforming) i sluttrapporten	SKOG-159	Documentation	Backlog	To Do
Som student ønsker jeg å skrive prosess-kapittelet i sluttrapporten	SKOG-157	Documentation	Backlog	To Do
Som student ønsker jeg å skrive teori-seksjon i sluttrapporten	SKOG-156	Documentation	Backlog	To Do
Som produkteier ønsker jeg at applikasjonen følger universal utforming	SKOG-91	Documentation	Backlog	To Do
Som produkteier ønsker jeg både produktivitet og kostnad i preview-noden	SKOG-234	Story	Backlog	To Do
Som produkteier ønsker jeg lenker i infotekst	SKOG-232	Story	Backlog	To Do
Som utvikler ønsker jeg å legge til et varsel dersom man er i ferd med å overskrive en lagret kalkulatorversjon	SKOG-210	Story	Backlog	To Do
Som utvikler ønsker jeg å kunne slette lagrede kalkulatorer	SKOG-197	Story	Backlog	To Do
Som utvikler ønsker jeg å endre endre kalkulatorversjonering fra nummer til string	SKOG-187	Story	Backlog	To Do
Som utvikler ønsker jeg å pynte/redesigne inputfeltene i rete-nodene	SKOG-186	Story	Backlog	To Do
Som utvikler ønsker jeg å lage en ternary (clamp) node i editoren	SKOG-183	Story	Backlog	To Do
Som utvikler ønsker jeg å tilpasse iht. skjermleser	SKOG-179	Story	Backlog	To Do
som bruker ønsker jeg å kunne bytte mellom kubikkpris og pris i kostnadsoverslag	SKOG-141	Story	Backlog	To Do
Som produkteier ønsker jeg mekanisme for å motta tilbakemelding fra brukere av kalkulatoren	SKOG-77	Story	Backlog	To Do

Appendix J

Frontend Research Notes

This document includes notes compiled during the research of libraries and technologies for frontend use. It features among other things, comparisons of core libraries, examines browser compatibility in relation to PWAs, and explores the functionality of service workers.

Research

Forms

Rammeverk	Pros	Cons
React Hook Form	• Har fokus på effektiv rendering • Støtter øvrige valideringsrammeverk som Yup, Zod, Joi, Superstruct og selvlagede. • lettvekt størrelse • lav til medium lærekurve • 0 avhengigheter • God dokumentasjon	• Hook basert. Ikke nødvendigvis en bakdel, men ikke noe jeg har kjennskap til @André Marthinsen • Ikke noen Form komponent
Formik	• Deklarativt, angivelig enkelt å jobbe med mtp state • components og hooks • Bruker basic React, så lett å sette i gang om man kjenner React. • medium lærekurve • God dokumentasjon	• State håndteres “lokalt og kortvarig”. Usikker på hva som ligger i dette. • Virker å hovedsakelig støtte Yup for validering

Datarepresentasjon

D3.js

D3

Bibliotek for dynamisk visualiering av data i form av grafer, pie-charts og lignende ved bruk av svg

charts.js

Chart.js | Chart.js

Enkelt, men lite customizable. Muligens et godt valg for prototyping.

nivo.js

Home

D3 + React abstraksjon

Grensesnitt for formel-endring

Rete.js

Examples overview - Rete.js

Bibliotek for visuell programmering. Tillater konfigurering av noder med inn og ut, samt hva disse nodene gjør funksjonellt.

CSS Rammeverk

CSS Framework	Kjennetegn	Pros	Cons
Bootstrap	• Dekleravitvt	• Mobile først • Veldig god dokumentasjon • Man kan finne temaer på nett ◦ Men mange koster penger	• Vanskeligere å tilpasse • Utseende messing ligner bootstrap-prosjekter hverandre • Relativt store filer
Foundation		• Mobile first • Katalog med “building blocks”	• Litt vanskeligere å lære • Mindre dokumentasjon
Materialize	• Basert på Material Design	• God UX • Kanskje bedre for mobilbruk?	• Litt kjedelig/standardist UI • Ekstremt lite fleksibel for tilpasning • Heavy bruk av JS
Tailwind CSS	• Imperativt • Bruker ferdiglagde CSS	• Veldig tilpasselig	• Høy læringskurve • Bloated markup

	klasser istedenfor å skrive ut hele CSS properties.		
Bulma		• Enkel å lære • Rask (basert på flexbox) • Renere syntaks	• Pure CSS (ingen JS) • Mindre dokumentasjon

↰ Vi bruker bootstrap som CSS rammeverk fordi lite behov for customization og tidligere erfaring. Dessuten gir det mindre markup bloat i JS koden vår.

- **Typescript**
  - For
    - Mer forutsigbart da typen av en variabel er kjennt ved transpile tid
    - Erfaring med det fra integrasjon
  - Mot
    - Til tider kronglete å finne den konkrete typen et bibliotek bruker som returtype eller parameter.
    - Må kompileres. Øker tid på pipeline osv. Sikkerhetshensyn.
    - Avhengig at transpiler oversetter koden effektivt.
    - Ekstra dependencies; type declaration files
  - Årsaken er behov for deling av kode knyttet til formelbehandling.
    - Ønske om API fra Allskog
    - Frontend SPA skal ha offline support
  - **Node.js 20.x** (siste LTS) brukes som runtime- og pakkesystem.
- **Frontend** avhengigheter/rammeverk:
  - **Bootstrap** brukes som CSS rammeverk
    - Behøver ikke store style-tilpasninger i prosjektet
    - Mindre bloat
    - Kjennskap fra før av
    - God støtte for forms
  - **React** brukes som frontend komponentbibliotek
    - Lett å skape dynamisk innhold
    - Deklarativt og komponentbasert
    - Tidligere brukt i integrasjon
    - Bred support i andre biblioteker og rammeverk
  - **Redux** brukes for state management
    - Forenkler håndtering av state med en struktur som ligner MVC arkitektur.
  - **Rete** brukes for visualisering av matematisk formel i administrasjonskonsollet.
    - Intuitiv måte å endre en formel samtidig som man kan endre andre egenskaper ved presentasjon som gruppering av input, output, osv.

- **Backend** avhengigheter/rammeverk
  - **Express** brukes som web-service rammeverk
    - Abstraksjon/forenkling av REST
- **PaaS/BaaS**
  - **Firebase** eller **MongoDB** brukes som database
    - Firebase
      - For
        - Google integrasjon
        - Gir støtte for autentiseringstjenester
          - Forenkler vesentlig autentiseringsbiten vi skal gjøre senere i prosjektet
        - Høy sannsynlighet for at kunden skal bruke google som PaaS, forenkler drift/betaling å samle begge under samme leverandør
        - Antakelig bedre støtte for offline-bruk og caching. Støtter også å synkronisere så fort nettverk blir tilgjengelig igjen (scheduling), som HumongousDB antakeligvis ikke støtter natively
        - Vi har brukt før i cloud
        - BaaS → Ingen administrasjonsbehov ift. skalering og backup
    - HumongousDB
      - For
        - Tillater self-hosting av DB
        - Brukt i integrasjonsprosjekt i form av MongoDB Atlas
  - **Google App Engine** brukes som vertsplatform i utviklingsfasen.

PWA: Browser support

Informasjon om app installasjon: Progressive Web Apps Compatibility – firt.dev

Kilde for bruksdata: Statcounter Global Stats - Browser, OS, Search Engine including Mobile Usage Share

NB: Markedsandel må tas med en klype salt. Usikkert i hvilken grad statcounter kan måle bruken av eksempelvis firefox pga beskyttelse mot sporing, men Norge virker i betydelig grad å ha en større andel iOS brukere en globalt.

Support for manifest felt: "PWA" | Can I use... Support tables for HTML5, CSS3, etc

iOS mobil og pad info: iOS PWA Compatibility – firt.dev

i**OS støtter egendefinerte felter i manifest** som tillater at PWA virker mer som en tradisjonell app som full screen osv.

Desktop

Browser	Bruk	Installasjon	Offline Workers
Chrome	62.21%	Ja	Ja
Safari	15.61%	Delvis. Siste Safari variant, Sonoma, har en funksjon som tillater at alle sider lagres som en app,	Ja

		men ikke tidligere versjoner.	
Edge	12.31%	Ja	
Firefox	5%	Nei	Nei
Opera	4.34%	Nei	Nei

Pad

Browser	Bruk	Installasjon	Offline Workers
Safari	48.69%	Delvis. Kan lastes ned og leggs blant apper og fungerer tilsynelatende, men ingen spesiell PWA installasjonsprompt.	Ja
Chrome	47.01%	Ja	Ja
Android	2.53%		
Opera	0.77%	Nei	Nei
Edge	0.72%	Ja	Ja

Mobil

På iOS enheter kan kun Safari legge websider til forsiden

Browser	**%**	**Installasjon**	Offline Workers	Notater
Safari	57.65%	Delvis. Kan lastes ned og leggs blant apper og fungerer tilsynelatende, men ingen spesiell PWA installasjonsprompt.	Ja	• Ikke felles minne mellom browser og installasjon.
Chrome	33.48%	Ja	Ja	
Samsung Internet	7.3%	Ja	Ja	
Opera	0.44%	Nei	Nei	
Firefox	0.63%	• Android: Ja • iOS: Nei		
Edge	0.3	Ja		

HTML parsing og sanitering

- npm: react-html-parser - pakke for å parse string til HTML
- npm: dompurify - pakke for sanitering av HTML

Lokallagring PWA

Lagring

- **IndexDB** brukes hovedsaklig til **strukturert data**.
- **Cache Storage** brukes hovedsaklig til **URL requests**.
- **Web Storage** er generelt uegnet til caching.

Type	Kjennetegn	Annet
Web Storage • localStorage • sessionStorage	• Dårligste valg pga API-en ikke er asynkron, og key/value lagring har begrensninger • Er IKKE tilgjengelig via service worker context	
IndexedDB	• God ytelse • NoSQL med støtte for lagring av strukturert og binærdata ◦ Bedre query-mekanisme enn Web Storage • Støtte for å lagre endringer i kø for opplasting når nettverk blir online	API: IndexedDB API - Web APIs | MDN
Cache Storage	• Cacher HTTP requests/responses helt identisk fra tidligere. ◦ Vil kunne gi cachet/simulert response når nettverk er offline. • Man kan også cache index.html,	API: CacheStorage - Web APIs | MDN Støtte fra Chrome 43, edge 16, Firefox 41, Safari 11.1 Tilgjengelig fra: • PWA main thread • Service Worker

	CSS, JS, fonts, data	• Andre workers(?)

Caching approach		
Precaching	• Man velger på forhånd hvilket innhold som skal lastes inn ved kald oppstart	
Cache as needed	• Når en request gjøres så legges respons i cache ◦ Typisk innhold uavhengig av appen, og som man forventer vil kunne endre seg • Finnes flere sub-strategier	
API + web service	Caching av API response, enten i Cache API eller IndexDB	

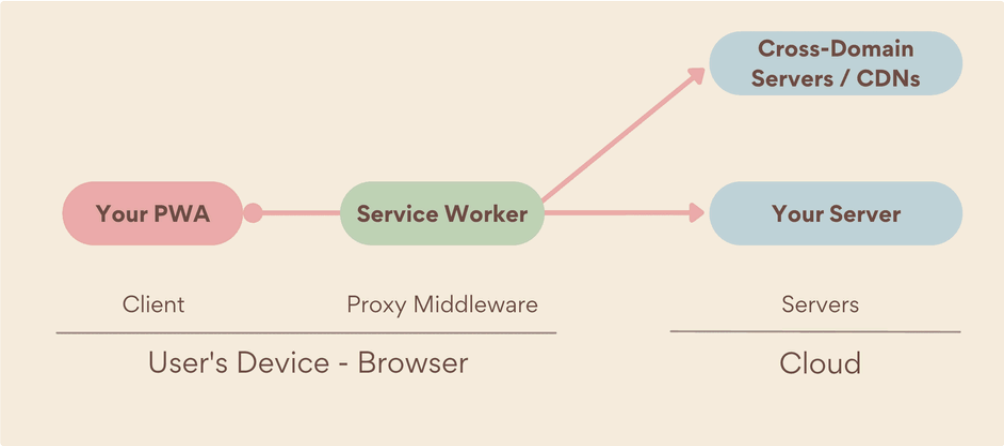
Caching strategi	Beskrivelse
Cache first	Gi respons fra cache, og hvis ikke funnet så send network request.
Network first	Gi respons fra nettverk, og hvis ikke svar så ta respons fra cache
Stale while revalidate	Gi respons fra cache, og i bakgrunnen last ned nyere versjon som brukes på fremtidige request.
Network-only	Ingen caching
Cache-only	Alltid bruk cache. Ingen nettverk • Typisk hvis man cacher alt under installasjon

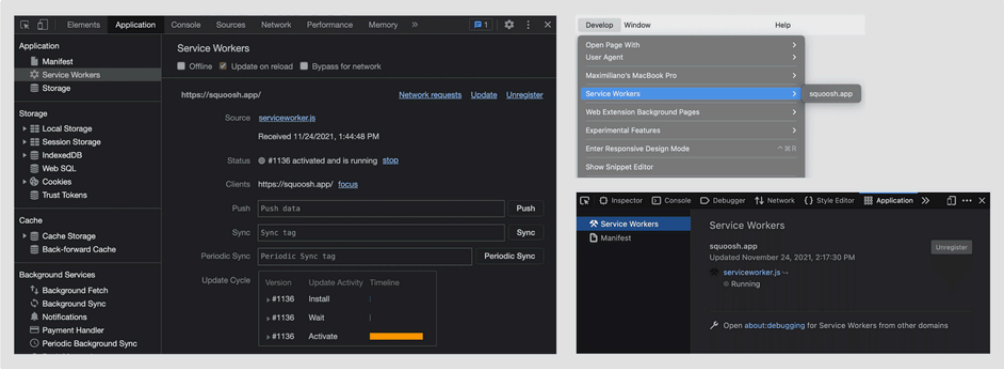
Kodesnutter og diagrammer for de øvrige strategiene: Serving | web.dev

Det er også andre strategier man kan bruke, beskrevet senere i Update | web.dev

Service Workers

- Bruksområder:
  - Fetche data fra cache når nettverk er offline
  - Sende notifications
  - Legge til “badge” på PWA ikon
  - Oppdatere data i bakgrunnen
  - …..Generelt få PWA-en til å fungere offline.
  - Annet: Capabilities | web.dev
- Hvordan den fungerer:
  - Man registrerer den via API
    - Limit på 1 registrert Service Worker pr PWA
    - Enkel if-statement sjekker om browser støtter serviceworker
    - Må ta hensyn til “scope” (implementaojonsdetaljer)
    - Browser parser lifecycle-filen og executer `install` (silent)
      - `install` er Ikke det samme som installasjon av PWA
      - Serviceworker kan kjøre på enheter som ikke engang støtter PWA (Desktop Safari/Firefox)
      - Oppdatering skjer når nettsiden endres (byte-different), men aktiveres kun når man har lukket ned alle vinduer, fordi den forrige workeren ikke slås av automatisk.
  - Den “fanger” og videresender requests/responses mellom PWA og API
    - Kjører i egen tråd, uavhengig av om siden er åpen.
      - In-memory state er ikke konsistent. State må lagres i feks IndexedDB
      - Service Worker kjører ikke hele tiden, og kan stenges ned etter noen sekunder





Service Worker I Developer Mode (Chrome, Firefox, Safari)

Workbox

- Bibliotek som ernkapsulerer low-level APIer som service worker og cache storage.
- Veldig populær. Brukes av 54% av tjenester med service-worker.
- Inkludert i mange byggeverktøy feks Create-React-App. Kan bygge manifest-fil osv.
- Alternativer: sw-tools og UpUp.

Rask tutorial: Workbox | web.dev

Material Design ikoner

- Ikoner lagt til som Font i index.html
- MUI: wrapper for material ikoner i React

Dette fungerte kun for enkelte ikoner.

React Icon Component - Material UI

Nytt forsøk, installere ikoner som svg (importer enkeltikoner):

React Icon Component - Material UI

Fortsatt er det ikke alle material-ikoner som finnes, men søkbart liste finnes:

https://mui.com/material-ui/material-icons/?theme=Outlined&query=un

Endret til React-icons for å kun importere icons, ikke helt helt rammeverk:

npm: react-icons

Søkbar liste over ikoner:

React Icons

Appendix K

Accessibility report

A report of accessibility issues in the web app. The focus is predominantly on making the application function with screen readers, but it also touches on accessibility best practices more broadly.

Tilgjengelighet

Status på siden per 19/3

Testet med WAVE-verktøy.

Utfordringer

- En del knapper uten tekst som skjermleser kan finne - for eksempel i navbaren, og i input-feltene (både info og reset)
  - Kan være et bootstrap-problem (å legge til “type={“input”} funker ikke)
- Tomme headere som pakker inn iconene i produktivitet, kostnad, kostnadsoverslag og lignende.
- Missing form-label og aria-label på number-input-felter
  - Endre aria-describedby til aria-label
- m3 blir lest som superscript 3, ikke som kubikkmeter

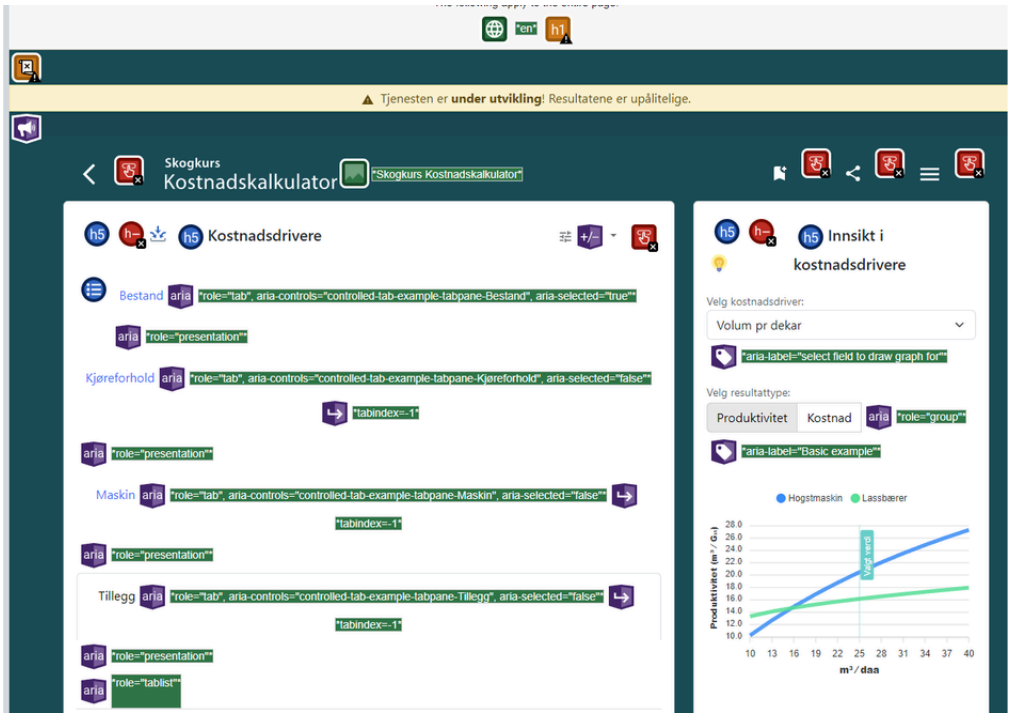
Symbolforklaring

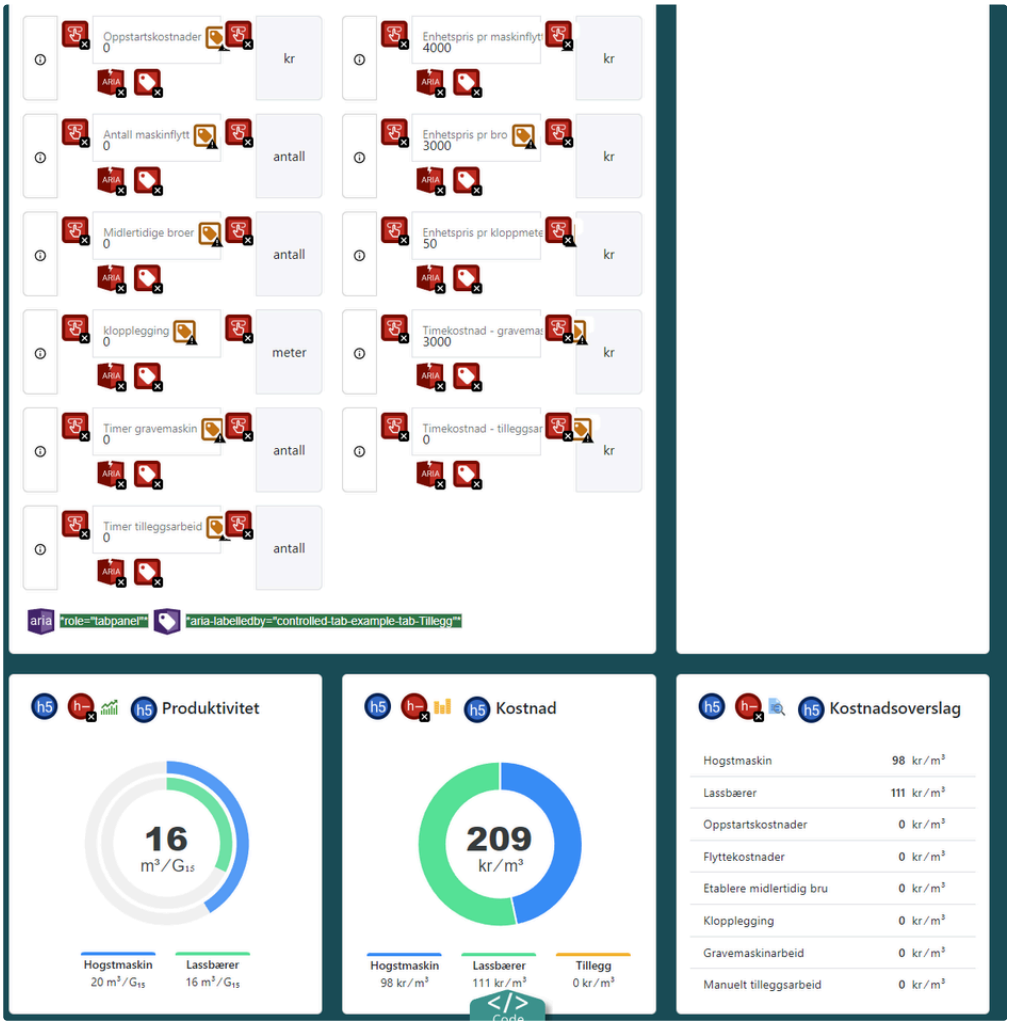
- Missing form label - input-formen har ingen tittel/navn-verdi.
- Empty heading - heading uten innhold, i vårt tilfelle er det bare ikoner og ingen tekst som screen reader kan se.
- Empty button - button uten tilhørende navn/tittel.
- Broken ARIA-reference - problemer med aria-label og aria-describedby. Aria-label legger på navn, mens aria-describedby brukes vanligvis til å legge til “instructions, format requirements or an error message” (What are the differences between label, aria-labelledby and aria-describedby? | Accessible Web). Tiltak: bytt ut describedby med label

- [x] 104 Errors
  - [x] 21 X Missing form label
  - [x] 5 X Empty heading
  - [x] 57 X Empty button
  - [x] 21 X Broken ARIA reference

- [x] 28 Alerts
  - [x] 26 X Orphaned form label
  - [x] 1 X Missing first level heading
  - [x] 1 X Noscript element

Skjermbilder




Oppstartskostnader
0
kr
Enhetspris pr maskinflytt
4000
kr
Antall maskinflytt
0
antall
Enhetspris pr bro
3000
kr
Midlertidige broer
0
antall
Enhetspris pr kloppmete
50
kr
klopplegging
0
meter
Timekostnad - gravema
3000
kr
Timer gravemaskin
0
antall
Timekostnad - tilleggsar
0
kr
Timer tilleggsarbeid
0
antall
"role="tabpanel""
"aria-labelledby="controlled-tab-example-tab-Tillegg""
Produktivitet
16
m³/G15
Hogstmaskin
20 m³/G15
Lassbærer
16 m³/G15
Kostnad
209
kr/m³
Hogstmaskin
98 kr/m³
Lassbærer
111 kr/m³
Tillegg
0 kr/m³
Code
Kostnadsoverslag
Hogstmaskin 98 kr/m³
Lassbærer 111 kr/m³
Oppstartskostnader 0 kr/m³
Flyttekostnader 0 kr/m³
Etablere midlertidig bru 0 kr/m³
Klopplegging 0 kr/m³
Gravemaskinarbeid 0 kr/m³
Manuelt tilleggsarbeid 0 kr/m³

Appendix L

Meeting Minutes

This document consists of the minutes for all the formal meetings held throughout the project. They are grouped by which sprint they took place in, and are elaborated by screenshots of the sprint backlog and the application.

Sprint 1

Start dato	**10 Jan 2024**
Slutt dato	16 Jan 2024

Møtereferat - Produkt Backlog Raffinering

Dato	10 Jan 2024
Tilstede	• Øystein Qvigstad • André Marthinsen • Even Stetrud • Paul Schiager • Mikael Fønus, Skogkurs • Brede Lauten, Skogkurs

Agenda	• Fastslå forretningsmål
	• Prioritere PBIer for de to neste sprintene
	  ◦ Bryte ned store PBIer
	  ◦ Definere akseptkriterier

Forretningsmål	Utarbeide plan og rutiner for prosjektet
Notater	• Raffineringsmøte skal vanligvis foregå med Mikael (bortreist) og/eller Brede. ◦ Vi er i oppstartsfasen, og ikke startet med utvikling. • Utviklingsteamet brukte tiden til å sette opp user stories for prosjektplanen og andre aktiviteter som skjer i oppstartsfasen. • Forhåpentligvis får vi til et ordentlig PB Raffineringsmøte neste uke.

Møtereferat - Sprint Planning

Dato	10 Jan 2024
Tilstede	• Øystein Qvigstad • André Marthinsen • Even Stetrud • Paul Schiager
Agenda	• Tidsestimere team-kapasitet • Tildele oppgaver ○ Tidsestimere PBIer • Starte ny sprint
Notater	• Størrelsesenhet for Story Points dene uken er antall timer. Definisjon vil opprettes senere

Team kapasitet	Prosent kapasitet (100% = 18 timer)
Øystein Qvigstad	110%
André Marthinsen	120%
Even Stetrud	100%
Paul Schiager	100%
Estimert kapasitet	

Møtereferat - Sprint Review

- Sprint Review møter avholdes ikke i oppstartsfasen av prosjektet ettersom vi ikke har noe å vise til interessenter enda. Når møtevirksomhet for Sprint Review starter opp forventes det å bli avholdt på tirsdager kl 13.

Eventuell intern agenda flyttes til internkursingsmøtet.

Møtereferat - Sprint Retrospective

Dato	16 Jan 2024

Til stede	• Øystein Qvigstad • André Marthinsen • Even Stetrud • Paul Schiager
Agenda	• Generelt: Hva gikk bra? Hva gikk dårlig? Hva kan vi forbedre?
Notater	• Litt for tidlig i prosessen å komme med konkrete forslag/tiltak

Fortsett med	**Start med**	**Slutt med**
Bruke kontorene i Brumunddal		
Skrive timer på slutten av hver dag		

Møtereferat - Intern Review

Dato	**16 Jan 2024**
Tilstede	• Øystein Qvigstad • André Marthinsen • Even Stetrud • Paul Schiager
Agenda	• Utviklere viser frem kode utviklet under sprinten seg i mellom. • Utviklere gir status på fortrolighet med biblioteker og kodebase, behov for kunnskapsdeling luftes.
Notater	• André viste frem bruk av enkle noder i rete.js • Øystein viste frem GitHub Action/Pipeline og noe smått i Google App Engine • Det er ikke så mye å vise frem på dette tidspunktet

Møtereferat - Veiledningsmøte

Dato	**10 Jan 2024**

Tilstede	• Øystein Qvigstad • André Marthinsen • Even Stetrud • Paul Schiager • Peter Stefan Nussbaum, Rådgiver NTNU
Agenda	• Gjennomgang av spørsmålsliste • Gjennomgang av prosjektplan og internsystemer sålangt

Notater	Diskusjoner:
	
	• Bør vi ha en sluttdato for endringer/ønsker til nye funksjoner?
	• Signering av gruppekontrakt.
	• Er mål og oppgavebeskrivelse i samsvar med kunden? Kanskje lage et møte/prosess for godkjenning.
	• Soft deadline for innlevering. Tar diskusjon på konkret tidspunkt senere.
	• Har naboland eller lignende industrier tilsvarende løsning eller gått gjennom samme prosess?
	  ○ Granberg?
	  ○ Hvordan fungerer de kalkylene som allskog og Glommen bruker?

Spørsmålsstiller	Spørsmål	Resultat
Alle	Hva menes med “ressursbehov”, ref. prosjektplan	Sannsynligvis hardwarekrav
Alle	Hvor mye kan vi sakse fra integrasjonsprosjektet? Må vi referere til eget arbeid? Hvor mye bør ordlyden endres?	Vi kan eventuelt legge inn henvisning til tidligere prosjektplan om at noe av ordlyden kan være lik. Enten som referanse eller fotnote
Øystein Qvigstad	Kan vi sende møteinnkallelser til deg, eller blir det tungtvindt om tidene endret	Kan skrive agenda dagen før. Øystein sender møteinnkallelse
Alle	Punkt i prosjektplanen: “Oppfølging og rapportering underveis”. Hva legges i dette?	Dette faller under “øvrige forhold”, og ikke påkrevd. Kan sikkert besvares på mange ulike måter.
Alle	Punkt i prosjektplanen: ”krav til statusmøter og beslutningspunkter”. Avklaring av dette.	Dekkes av svaret ovenfor

Ekstraordinære møter

Dato	09 Jan 2024
Tilstede	• Øystein Qvigstad • André Marthinsen • Even Stetrud • Paul Schiager
Agenda	• Rollefordeling

Notater	• Sekretær, notering og møtereferater - den som sier noe, noterer noe (etter at man har diskutert seg fram til konklusjon)
	• Øystein tar rollen som Scrum Master og Deployment-ansvarlig
	• André Tar ansvar som Arkitektur og Test ansvarlig
	• Even tar ansvar som dokument- og rapportettersynansvarlig
	• Paul tar ansvar som UX/UI Designer
	• Even og Paul deler på ansvar for sikkerhet
	
	• **Scrum Master** Øystein Qvigstad
	  ○ Overordnet ansvar for gjennomføring av Scrum
	  ▪ Organisere og lede scrum-aktiviteter som sprint-review, -planlegging osv.
	  ○ Fasilitere kommunikasjon mellom utviklingsteam og PO/andre involverte
	• **UI/UX Design** Paul Schiager
	  ○ Organisere brukertester
	  ○ Analayse/innhenting av PO/interessenters grensesnittkrav
	  ○ Tilgjengelighetsanalyse
	  ○ Figma-eierskap
	• **Arkitekt** André Marthinsen
	  ○ Utarbeide arkitektur
	  ▪ Kodearkitektur
	  ▪ Systemarkitektur
	• **Sikkerhet** Even Stetrud Paul Schiager
	  ○ Verdianalyse
	  ○ Kartlegge risikoer og tiltak
	  ▪ Deployment
	  ▪ Systemarkitektur
	  ▪ Kode
	  ○ Etterse oppfyllelse av sikkerhetskrav
	• **Dokument/rapportettersyn** Even Stetrud
	  ○ Påse at diagrammer for kode lages
	  ○ Identifisere tidlige rapportskrivingsmuligheter
	  ○ Pådriver for rapportskriving
	• **Deployment / Pipeline** Øystein Qvigstad
	  ○ Utforme pipeline og git arbeidsflyt
	  ○ Kordinere oppsett / struktur av utrulling og hosting
	• **Testing** André Marthinsen
	  ○ Oppfølge utviklere på testing
	  ○ Vurdere om tester er hensiktsmessig utformet

Dato	**12 Jan 2024**

Tilstede	• Øystein Qvigstad • André Marthinsen • Even Stetrud • Paul Schiager • Kjetil Finsrud
Agenda	• Innledende prat om utrulling og tilgjengelig ressurser

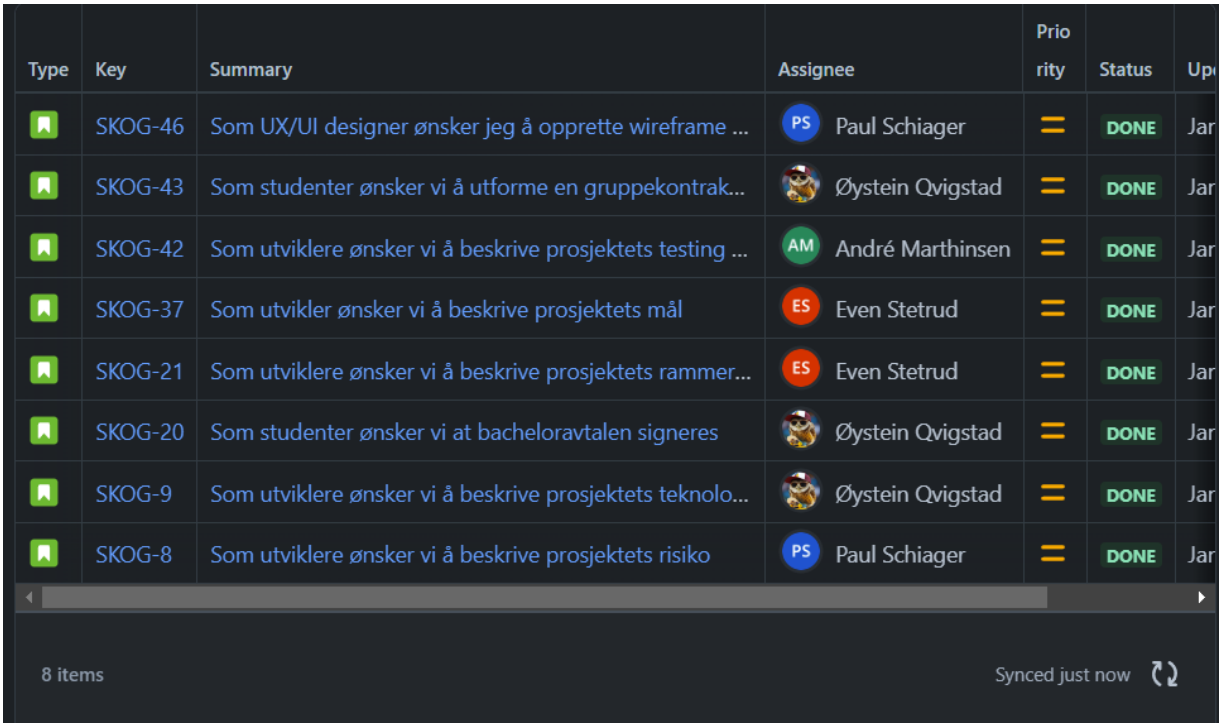
Notater	• Skogkurs (v/ Kjetil) kan anskaffe subdomene og tilhørende TLS sertifikat
	• Skogkurs er villig til å betale lisenspenger for ekstra tjenester.
	• Skogkurs har ingen åpen dialog med tjenesteleverandører som kan være aktuelle
	  ◦ Nettsiden http://Skogkurs.no er laget i wordpress
	  ◦ Skogfondkalkulatoren benytter en distribusjonsmappe (FTP) som Kjetil laster opp filene til. Dette virker kun som statiske filer - mens vi har behov for tjenere i tillegg.
	  ◦ Skogkurs har egen selskapskonto hos Google for typ AdSense. Dette kan gjøre Google Cloud aktuell.
	  ◦ Skogkurs har Azureabonnement for interndrifting
	• Kjetil har ekstra ansvar ovenfor Web og GPDR
	• Øystein Syversen har ansvar for intern IT-drift, inkludert Office 365.

Dato	**12 Jan 2024**
Tilstede	• Øystein Qvigstad • André Marthinsen • Even Stetrud • Paul Schiager • Kjetil Finsrud
Agenda	• Standup • Demo formelanalyse • Kort gjennomgang av pipeline og deployment • Snakke om budsjetting av reiseutgifter • Punkt 5 i kontrakten, hvem som vil ha eierskap til resultatet • Gjennomgang av timeline og milestones

Notater	• Ta kontakt med Brede
	  o Organisere interne møter mellom PO og Interessenter?
	  ▪ Brede mener det bør hovedsaklig foregår gjennom Sprint Review
	  o Har han tid fredag til PB raffineringsmøte?
	  ▪ Uken i utgangspunktet full
	  ▪ Mandager er beste dagen, fra morgenen kl 9.
	  ▪ 29. går ikke
	  o Punkter i planen som han kan se over:
	  ▪ Oppgavebeskrivelsen, fagområde/domene.
	  ▪ Våre tanker om admin grensesnitt
	• Ta kontakt med Mikael
	  o Når blir Mikael borte? Kapasitet?
	  ▪ Han håper å få operasjon i løpet av uken.
	  ▪ Blir legge på intensiv overvåkning noen uker.
	  o Kan fint få flere nøkkelkort
	  o Budsjett kjøring
	  ▪ Loggbok i excel, realistisk tall på budsjett
	
	• Foreløpig prosjektplanen sendes som PDF til Mikael og Brede
	• Vi må snakke med Peter ang pkt. 5A/B i kontrakten. Skogkurs skal kunne redigere og tilpasse kildekode. Dersom dette ikke kommer tydelig frem i 5A. Skal vi skrive fotnote/kommentar om hva som er ment?
	• Reelle reisekostnader skal loggføres eksakt til slutten av prosjektet. Budsjett er kun et estimat for intern prosjektøkonomi. Hvert medlem må budsjettere for sine kostnader.
	• Mandager kl 9 er beste tidspunkt å ha PO-møter med Brede mens Mikael er sykemeldt.

Sprint 2

Start dato	**17 Jan 2024**
Slutt dato	23 Jan 2024



Type	Key	Summary	Assignee	Priority	Status	Up
	SKOG-46	Som UX/UI designer ønsker jeg å opprette wireframe ...	Paul Schiager	=	DONE	Jar
	SKOG-43	Som studenter ønsker vi å utforme en gruppekontrak...	Øystein Qvigstad	=	DONE	Jar
	SKOG-42	Som utviklere ønsker vi å beskrive prosjektets testing ...	André Marthinsen	=	DONE	Jar
	SKOG-37	Som utvikler ønsker vi å beskrive prosjektets mål	Even Stetrud	=	DONE	Jar
	SKOG-21	Som utviklere ønsker vi å beskrive prosjektets rammer...	Even Stetrud	=	DONE	Jar
	SKOG-20	Som studenter ønsker vi at bacheloravtalen signeres	Øystein Qvigstad	=	DONE	Jar
	SKOG-9	Som utviklere ønsker vi å beskrive prosjektets teknolo...	Øystein Qvigstad	=	DONE	Jar
	SKOG-8	Som utviklere ønsker vi å beskrive prosjektets risiko	Paul Schiager	=	DONE	Jar

8 items Synced just now

- Møtereferat - Produkt Backlog Raffinering
- Møtereferat - Sprint Planning
- Møtereferat - Sprint Review
- Møtereferat - Sprint Retrospective
- Møtereferat - Intern Review
- Møtereferat - Veiledningsmøte
- Ekstraordinære møter

Møtereferat - Produkt Backlog Raffinering

Dato	**17 Jan 2024**

Tilstede	• Øystein Qvigstad • André Marthinsen • Even Stetrud • Paul Schiager • Mikael Fønus, Skogkurs • Brede Lauten, Skogkurs
Agenda	• Oppretting, oppstykking og prioritering av gjennomførbare oppgaver.
Forretningsmål	
Notater	

notater til senere

- Snakke løst om «produkt backlog»-prosessen og hvordan sikre ulike interesser (forretningsmessige, interessenter, tekniske, akademiske).
-
- Eventuell gjennomgang av andre dokumenter/spørsmål.
    - Oppgavebeskrivelse/fagområde/domene i prosjektplan
    - Våre tanker om grensesnitt for formeltilpasning

Møtereferat - Sprint Planning

Dato	**17 Jan 2024**
Tilstede	• Øystein Qvigstad • André Marthinsen • Even Stetrud • Paul Schiager

Agenda	• Tilgjengelig tid estimeres for neste sprint basert på forrige sprint og timeplan
	• PBI-er tidsestimeres av utviklerne i fellesskap
	• PBI-ere velges ut i tråd med prioritering og tid
	  ◦ Utviklere plukker fra de utvalgte oppgavene basert på lyst og kompetanse.
	  ◦ Der mulig deles utviklere inn to og to med et gitt ansvarsområde for å promotere par-programmering og kunskapsdeling
	  ◦ Hver utvikler deler sine PBI-er opp i “tasks” og estimerer tid for gjennomføring.

Notater	

Team kapasitet	Prosent kapasitet (100% = 18 timer)
Øystein Qvigstad	160%
André Marthinsen	120%
Even Stetrud	100%
Paul Schiager	100%
Estimert kapasitet	29

Møtereferat - Sprint Review

- Sprint Review møter avholdes ikke i oppstartsfasen av prosjektet ettersom vi ikke har noe å vise til interessenter enda. Når møtevirksomhet for Sprint Review starter opp forventes det å bli avholdt på tirsdager kl 13.

Eventuell intern agenda flyttes til internkursingsmøtet.

Møtereferat - Sprint Retrospective

Dato	23 Jan 2024
Til stede	• Øystein Qvigstad • André Marthinsen • Even Stetrud • Paul Schiager
Agenda	• Generelt: Hva gikk bra? Hva gikk dårlig? Hva kan vi forbedre?
Notater	

Fortsett med	Start med	Slutt med
Gi god informasjon i forkant av møter osv.	Planlegge review-møter for interessenter i forkant.	
	• Det skal lage ny kategori i Clockify for UX-design	

Møtereferat - Intern Review

Dato	23 Jan 2024
Tilstede	• Øystein Qvigstad • André Marthinsen • Even Stetrud • Paul Schiager
Agenda	• Gjennomgang og lukking av nåværende sprint • Utviklere viser frem kode utviklet under sprinten seg i mellom. • Utviklere gir status på fortrolighet med biblioteker og kodebase, behov for kunnskapsdeling luftes.
Notater	• Det var ikke mye å vise frem. Vi har jobbet med prosjektplan for det meste ◦ Paul viste frem sin UX/UI-prototype i forkant • Alle burde undersøke grunnleggende Redux • Vi burde tenke på browsersupport iht PWA • Man burde se på fordeler ift. Form-rammeverk

Møtereferat - Veiledningsmøte

Dato	17 Jan 2024
Tilstede	• Øystein Qvigstad • André Marthinsen • Even Stetrud • Paul Schiager • Peter Stefan Nussbaum, Rådgiver NTNU
Agenda	• Gjennomgang av spørsmålsliste • Gjennomgang av prosjektplan og internsystemer sålangt
Notater	• Det viktigste er at målene er gjensidige mellom studenter og Skogkurs • Noen refleksjoner om integrasjon av KI i produktet kan være fint å ha.

Ekstraordinære møter

Dato	17 Jan 2024
Tilstede	• Øystein Qvigstad • André Marthinsen • Even Stetrud • Paul Schiager • Mikael Fønhus
Agenda	• Tilbakemelding prosjektplan
Notater	• Domeneseksjon ser bra ut, Mikael svarer på spørsmål til fagperson som notert i planen ○ Mikael noterer kommentarer direkte i PDF-en og sender tilbake til oss. • Bærekraft avklares noe senere. Mikael tar kontakt med bærekraft-trainee (Camilla) på Skogkurs, vi kan enten delta på møte om inkorporering av bærekraft i kursmateriell, eller få foilsett som blir utarbeidet på bakgrunn av dette. • Mikael ønsker å holdes i loopen mens han er sykemeldt, gitt at det enkelt innebærer å sette han på "cc" på e-post. • Oppsummeringsmøte med Mikael når han er tilbake.

Sprint 3

Start dato	**24 Jan 2024**
Slutt dato	30 Jan 2024

Møtereferat - Produkt Backlog Raffinering

Dato	22 Jan 2024
Tilstede	• Øystein Qvigstad • André Marthinsen • Even Stetrud • Paul Schiager • Mikael Fønus, Skogkurs • Brede Lauten, Skogkurs

Agenda	• Snakke løst om «produkt backlog»-prosessen og hvordan sikre ulike interesser (forretningsmessige, interessenter, tekniske, akademiske).
	• Eventuell gjennomgang av andre dokumenter/spørsmål.
	  ◦ Oppgavebeskrivelse/fagområde/domene i prosjektplan
	  ◦ Våre tanker om grensesnitt for formeltilpasning

Forretningsmål	• Se formelen fungere i en nettleser

Notater	• Kontroll av urimelige (gyldige, men “usannsynlige”) verdier (“Er du sikker på at dette er riktig?”) er ikke prioritert - vanskelig å implementere, stor forskjell i rimelige verdier fra region til region • skogdriftsledere/entreprenører ønsker ikke at sine utførte kostnadskalkulasjoner skal være tilgjengelig for sine konkurrenter, men de vil selv kunne hente ut igjen gamle kalkulasjoner. Innlogging/sikkerhet? - Brede skal snakke om dette med interessenter • lagring av data diskuteres med interessenter i sprint review. Sett dette på agendaen i innkallingen. • Undersøke om tilgang til forskningsdata som kalkulatoren er basert på - Nibio

---

Møtereferat - Sprint Planning

Dato	**24 Jan 2024**
Tilstede	• Øystein Qvigstad • André Marthinsen • Even Stetrud • Paul Schiager

Agenda	• Tilgjengelig tid estimeres for neste sprint basert på forrige sprint og timeplan
	• PBI-er tidsestimeres av utviklerne i fellesskap
	• PBI-ere velges ut i tråd med prioritering og tid
	  ○ Utviklere plukker fra de utvalgte oppgavene basert på lyst og kompetanse.
	  ○ Der mulig deles utviklere inn to og to med et gitt ansvarsområde for å promotere par-programmering og kunskapsdeling
	  ○ Hver utvikler deler sine PBI-er opp i “tasks” og estimerer tid for gjennomføring.

Notater	

Team kapasitet	**Prosent kapasitet (100% = 18 timer)**
Øystein Qvigstad	100%
André Marthinsen	100%
Even Stetrud	100%
Paul Schiager	100%
Estimert kapasitet	24

Møtereferat - Sprint Review

Møtereferat - Sprint Retrospective

Dato	30 Jan 2024
Til stede	• Øystein Qvigstad • André Marthinsen • Even Stetrud • Paul Schiager
Agenda	• Generelt: Hva gikk bra? Hva gikk dårlig? Hva kan vi forbedre?

Notater	• Erfaringer med parprogrammering
	  ◦ Har generelt fungert bra
	  ◦ Bytte på “driver” og “observer”

Fortsett med	Start med	Slutt med
Alle deltar med å skrive referater		
Parprogrammering		
TDD		

Møtereferat - Intern Review

Dato	30 Jan 2024
Tilstede	• Øystein Qvigstad • André Marthinsen • Even Stetrud • Paul Schiager
Agenda	• Utviklere viser frem kode utviklet under sprinten seg i mellom. • Utviklere gir status på fortrolighet med biblioteker og kodebase, behov for kunnskapsdeling luftes.

Notater	• Vist frem Redux-implementasjon og ulike React-komponenter

Møtereferat - Veiledningsmøte

Dato	**24 Jan 2024**
Tilstede	• Øystein Qvigstad • André Marthinsen • Even Stetrud • Paul Schiager • Peter Stefan Nussbaum, Rådgiver NTNU
Agenda	• Gjennomgang av spørsmålsliste • Gjennomgang av prosjektplan og internsystemer sålangt
Notater	• Ingen spørsmål var på agendaen • Vi gikk gjennom prosjektplanen på storskjerm med Peter. ○ Peter vil skrive noen kommentarer på planen vår • Peter signerte prosjektavtale

Sprint 4

Start dato	**31 Jan 2024**
Slutt dato	06 Feb 2024

Møtereferat - Produkt Backlog Raffinering

Dato	29 Jan 2024
Tilstede	• Øystein Qvigstad • André Marthinsen • Even Stetrud • Paul Schiager • Mikael Fønus, Skogkurs • Brede Lauten, Skogkurs
Agenda	• Gjennomgang Confluence/Jira ◦ Samtale om backlog items • Demo av prototype vi har jobbet med siste uke
Forretningsmål	Forrige sprint: Se formelen fungere i en nettleser Neste sprint: Se formelen fungere i en nettleser
Notater	• Ragnhild: Lufte ønske om å kunne låne Merete (UX designer) for rådgivning/tips på nåværende prototype

---

Møtereferat - Sprint Planning

Dato	31 Jan 2024
Tilstede	• Øystein Qvigstad • André Marthinsen • Even Stetrud • Paul Schiager

Agenda	• Opprettelse/omprioritering av utvikler-spesifikke PBI-er
	• Tilgjengelig tid estimeres for neste sprint basert på forrige sprint og timeplan
	• PBI-er tidsestimeres av utviklerne i fellesskap
	• PBI-ere velges ut i tråd med prioritering og tid
	  ◦ Utviklere plukker fra de utvalgte oppgavene basert på lyst og kompetanse.
	  ◦ Der mulig deles utviklere inn to og to med et gitt ansvarsområde for å promotere par-programmering og kunskapsdeling
	  ◦ Hver utvikler deler sine PBI-er opp i “tasks” og estimerer tid for gjennomføring.

Notater	• Uken designeres til dokumentasjon, design og testing. ◦ Unntaket er fullføring av kode for feltinformasjon på nettsiden, og bugfixing av cacheproblem på nettsiden • Brukertesting er delt ansvar mellom Paul og André • Fullføring av feltinformasjon er delt ansvar mellom Even og André

Team kapasitet	Prosent kapasitet (100% = 18 timer = 22 SP)
Øystein Qvigstad	100%
André Marthinsen	100%
Even Stetrud	100%
Paul Schiager	70%
Estimert kapasitet	22

Møtereferat - Sprint Review

Dato	06 Feb 2024

Til stede	• Øystein Qvigstad • André Marthinsen • Even Stetrud • Paul Schiager • Jon Sigurd Leine • Erling Perstolen • Bjørn Lauritzen • Kjetil Finsrud • Brede Lauten • Helmer Belbo • Mikael Fønhus
Agenda	• Kort info om formål • Demo av nåværende status • Notere feedback/kommentarer/forslag • Spørsmål: Interesse for andre språk på nettsiden?

Notater	Faguttrykk oppdateres Oversikt over totalt antall parametere, så man ikke glemmer noen. Infoknapper: Forside med enkle instrukser. Vise oppdateringsdato. Parameter-gruppering: Side 1 ok (Skogbestand) Side 2 vei og terreng Side 3 lassbærer og hogstmaskin, sortiment (timekostnader vi være ganske konstant?) Defaultverdier basert på tidligere inntastet verdi? Eventuelt innstilling for å velge mellom default fra skogkurs og lagret verdi? Overflatestruktur på traktorvei - endres til: Basvei Overflatestruktur på terreng - endres til: Hogstfelt Vinter/sommerdrift - finnes det forskningsdata på dette? Skogtype - informasjon ok Legge til kulturskog? Må i så fall ha forskningsgrunnlag for dette. Mer bakgrunnsinfo for å vite hvilken skogtype man skal velge? Parameter for forhåndsrydding? Blir vel beskrevet av antall ryddetrær per dekar? Forenklet versjon - kommer senere i prosessen. Ang. gjennomsnittsverdier for overflatestruker og hellning - Allskog lager flere kalkyler for sprikende verdier for en hogst og volumveier disse mot hverandre, da snittet blir feil, G/M kategoriserer hele som lett, normal, vanskelig. Mangler parameter for motkjøring (ligger tømmeret nederst eller øverst?) Generelt om å stykke opp parameter/gjøre parameter mer granulær - G/M er skeptisk til veldig granulære parametre da det vil gjøre det vanskelig å finne riktig verdi

Ensidig/tosidig avlegg ? Glommen/Mjøsen har tall på dette. Målbart eller veldig godt forklart i kalkulatoren.

Ordlyd i inndatafelt:

Tømmertrær - endres til treantall? Eller skal man heller bruke middeldimensjon?

Vise resultat på hver enkelt side? Graf som viser endringen er ønsket.

Ønske om å vise utregnet middeldimensjon på side 1. Ryddetrær nederst.

I meny: Tallgrunnlag i tillegg til forskningsgrunnlag

Videreutvikling:

Parametere for rigging, graving, transport +++

Parameter for høyde over havet på hogstfelt og velteplass?

Flere driftstyper - lukket hogst. Modeller kommer fra Helmer, nokså klargjort. I tillegg kommer oppfølging av drift, for å bekrefte.

Nytt input-felt for hogstformer: Lukket hogst (Flatehogst, bledning, tynning), flatehogst … ?

Ønske om ukentlige møter for å holdes i loopen. I utgangspunktet 1 time 10:30.

Til neste møte:

- Lagring av beregning, med navn/beskrivelse
- Sende resultat på mail

Ønsket gruppering av felter:

Side 1	Side 2	Side 3	Side 4
Låglandskog	Lengde basvei	Timekostnad hogstmaskin	Produksjon og kostnad Hogstmaskin
Virkesuttak i tre/da	Helling basvei	Timekostnad lassbærer	Produksjon og kostnad Lassbærer
Ryddetre / da	Overflatestruktur basvei	Lass størrelse	Middeldimensjon
Volum m3/da	Lengde kjøring hogstfelt	Antall sortiment	
Driftsstørrelse	Helling kjøring hogstfelt		
	Overflatestruktur hogstfelt		

Møtereferat - Sprint Retrospective

Dato	06 Feb 2024
Til stede	• Øystein Qvigstad • André Marthinsen • Even Stetrud • Paul Schiager
Agenda	• Generelt: Hva gikk bra? Hva gikk dårlig? Hva kan vi forbedre?
Notater	• Sykdom og fravær påvirket framdrift • Vi justerer ekstern review prosedyrer iht punkter under

Fortsett med	Start med	Slutt med
	Litt mer konkret agenda	
	Opprette spørsmål til review dokument	

Møtereferat - Intern Review

Dato	06 Feb 2024
Tilstede	• Øystein Qvigstad • André Marthinsen • Even Stetrud • Paul Schiager
Agenda	• Utviklere viser frem kode utviklet under sprinten seg i mellom. • Utviklere gir status på fortrolighet med biblioteker og kodebase, behov for kunnskapsdeling luftes.
Notater	

Møtereferat - Veiledningsmøte

Dato	01 Feb 2024

Tilstede	• Øystein Qvigstad • André Marthinsen • Even Stetrud • Paul Schiager • Peter Stefan Nussbaum, Rådgiver NTNU
Agenda	• Demo av nåværende tjeneste • Gjennomgang av prosjektplan og internsystemer sålangt
Notater	• Ingen planlagte spørsmål denne uken • Researche lignende kalkulatorer fra andre land? Kan være fint å ha med i prosjektrapporten, før vi snevrer inn mot Skogkurs spesielt. • Vise til teori som er lært i tidligere emner ○ Evaluater setter pris på at vi har fått noe ut av tidligere emner - refleksjon over dette kan være lurt

Sprint 5

Start dato	**07 Feb 2024**
Slutt dato	13 Feb 2024

Møtereferat - Produkt Backlog Raffinering

Dato	29 Jan 2024
Tilstede	• Øystein Qvigstad • André Marthinsen • Even Stetrud • Paul Schiager • Mikael Fønus, Skogkurs • Brede Lauten, Skogkurs
Agenda	• Gjennomgang Confluence/Jira ◦ Samtale om backlog items • Demo av prototype vi har jobbet med siste uke
Forretningsmål	Forrige sprint: Se formelen fungere i en nettleser
Notater	Språkinnstilling tas opp på møte tirsdag (med øvrige interessenter). Se nettkurs om universell utforming. Ta kontakt med Dag Fjeld Sende ut litt info i forkant til Sprint Review i morgen

Møtereferat - Sprint Planning

Dato	07 Feb 2024
Tilstede	• Øystein Qvigstad • André Marthinsen • Even Stetrud • Paul Schiager

Agenda	• Opprettelse/omprioritering av utvikler-spesifikke PBI-er
	• Tilgjengelig tid estimeres for neste sprint basert på forrige sprint og timeplan
	• PBI-er tidsestimeres av utviklerne i fellesskap
	• PBI-ere velges ut i tråd med prioritering og tid
	  ◦ Utviklere plukker fra de utvalgte oppgavene basert på lyst og kompetanse.
	  ◦ Der mulig deles utviklere inn to og to med et gitt ansvarsområde for å promotere par-programmering og kunskapsdeling
	  ◦ Hver utvikler deler sine PBI-er opp i "tasks" og estimerer tid for gjennomføring.

Notater	• Be om innhold til "Tallgrunnlag"-side • Kopiere inn tekst fra regneark på "Forskningsgrunnlag"-side • Beregning av middelstamme settes i boks sammen med antall trær og antall kubikk. • Be om innhold til forside • Returnere error fra kalkulator i tilfelle NaN på ett av resultatene ?

Team kapasitet	Prosent kapasitet (100% = 18 timer = 22 SP)
Øystein Qvigstad	100%
André Marthinsen	70%
Even Stetrud	70%
Paul Schiager	100%
Estimert kapasitet	20

Møtereferat - Sprint Review

Møtereferat - Sprint Review

Dato	13 Feb 2024
Til stede	• Øystein Qvigstad • André Marthinsen • Even Stetrud • Paul Schiager • Jon Sigurd Leine • Erling Perstolen • Bjørn Lauritzen • Kjetil Finsrud • Brede Lauten • Helmer Belbo • Jon Harby (GM) • Mikael Fønhus
Agenda	• Gjennomgang av timeline • Plan for adminkosnoll • Informasjon om brukertesting
Notater	• Plan for brukertesting - noe senere så vi får med flere features? • Riggekostnader ? Ikke så relevant for oss, kan løses via grafisk formelbygger • Ulike input for ulike aktører - fint med felksibel løsning • Ikke en priskalkulator! • Tekst på hover over symboler

---

Møtereferat - Sprint Retrospective

Dato	13 Feb 2024
Til stede	• Øystein Qvigstad • André Marthinsen • Even Stetrud • Paul Schiager
Agenda	• Generelt: Hva gikk bra? Hva gikk dårlig? Hva kan vi forbedre?
Notater	• Review møtet med eksterne gikk bra i dag. Fortsette med det.

Fortsett med	Start med	Slutt med
Tydelig agenda på review møte	Huske å påminne om å skrive møteferat	

Møtereferat - Intern Review

Dato	
Tilstede	• Øystein Qvigstad • André Marthinsen • Even Stetrud • Paul Schiager
Agenda	• Utviklere viser frem kode utviklet under sprinten seg i mellom. • Utviklere gir status på fortrolighet med biblioteker og kodebase, behov for kunnskapsdeling luftes.
Notater	Brukertest: Undersøke hva som brukes i dag

Møtereferat - Veiledningsmøte

Dato	**07 Feb 2024**
Tilstede	• Øystein Qvigstad • André Marthinsen • Even Stetrud • Paul Schiager • Peter Stefan Nussbaum, Rådgiver NTNU
Agenda	•

Notater	• Ingen planlagte spørsmål denne uken
	• Punkter som ble tatt opp underveis
	  ◦ Resultater fra ekstern-review, funksjonalitets- og utformingsønsker kontra skogfaglig innspill som faller utenfor vårt domene
	  ◦ Resultatvisning på hver input-side, rotete på mobil-enheter, kanskje bedre på større enhetstyper

Sprint 6

Start dato	**14 Feb 2024**
Slutt dato	27 Feb 2024

Møtereferat - Produkt Backlog Raffinering

Dato	12 Feb 2024
Tilstede	• Øystein Qvigstad • André Marthinsen • Even Stetrud • Paul Schiager • Mikael Fønus, Skogkurs • Brede Lauten, Skogkurs
Agenda	• Gjennomgang Confluence/Jira ◦ Samtale om backlog items • Demo av prototype vi har jobbet med siste uke
Forretningsmål	Forrige sprint: Se formelen fungere i en nettleser
Notater	

Møtereferat - Sprint Planning

Dato	14 Feb 2024
Tilstede	• Øystein Qvigstad • André Marthinsen • Even Stetrud • Paul Schiager

Agenda	• Opprettelse/omprioritering av utvikler-spesifikke PBI-er
	• Tilgjengelig tid estimeres for neste sprint basert på forrige sprint og timeplan
	• PBI-er tidsestimeres av utviklerne i fellesskap
	• PBI-ere velges ut i tråd med prioritering og tid
	  ◦ Utviklere plukker fra de utvalgte oppgavene basert på lyst og kompetanse.
	  ◦ Der mulig deles utviklere inn to og to med et gitt ansvarsområde for å promotere par-programmering og kunskapsdeling
	  ◦ Hver utvikler deler sine PBI-er opp i “tasks” og estimerer tid for gjennomføring.

Notater	

Team kapasitet	Prosent kapasitet (100% = 36 timer = 12 SP)
Øystein Qvigstad	150%
André Marthinsen	100%
Even Stetrud	100%
Paul Schiager	100%
Estimert kapasitet	54

Møtereferat - Sprint Review

Dato	27 Feb 2024

Til stede	• Øystein Qvigstad • André Marthinsen • Even Stetrud • Paul Schiager • Jon Sigurd Leine • Jon Harby • Erling Perstolen • Bjørn Lauritzen • Kjetil Finsrud • Brede Lauten • Helmer Belbo • Mikael Fønhus
Agenda	• Resultatside - spesielt graf • Nevne såvidt tilleggene - vi legger inn en ny side

Notater	• Oppdatert resultatside
	  ◦ Graf ok - kan være litt krevende å skjønne. Grafen kan virke distraherende fra hoveddisplay - kanskje en synlighetsvalg for grafen.
	  ◦ Det mest interessante er produktivitet, kostnad og kostnadsoverslagene (som de vises i dag).
	  ◦ Fortsatt ønkelig at resultatene vises sammen med input-ene.
	  ◦ Å vise laveste kostnad, ofte mer interesssert i hogstmaskin. Hogstmaskin er ofte den mest kostbare av maskinene (60-40, 65-35 i hogstmaskins favør?).
	  ◦ Ønsker å se resultatet av endring av parameter i realtime (hva skjer hvis jeg endrer denne?)
	• På større skjerm - mye frem og tilbake når man må veksle mellom input-sider
	  ◦ ønske om input på samme side som resultat
	  ▪ Også på mindre enheter - kanskje vanskelig å implementere?
	• På input-sider: smal presentasjon av resultat
	• Brukes av mange ulike typer brukere og bruksområder (før, under og etter hogst)

Møtereferat - Sprint Retrospective

Dato	**13 Feb 2024**

Til stede	• Øystein Qvigstad • André Marthinsen • Even Stetrud • Paul Schiager
Agenda	• Generelt: Hva gikk bra? Hva gikk dårlig? Hva kan vi forbedre?
Notater	

Fortsett med	Start med	Slutt med

Møtereferat - Intern Review

Dato	
Tilstede	• Øystein Qvigstad • André Marthinsen • Even Stetrud • Paul Schiager
Agenda	• Utviklere viser frem kode utviklet under sprinten seg i mellom. • Utviklere gir status på fortrolighet med biblioteker og kodebase, behov for kunnskapsdeling luftes.
Notater	

Møtereferat - Veiledningsmøte

Dato	14 Feb 2024
Tilstede	• Øystein Qvigstad • André Marthinsen • Even Stetrud • Paul Schiager • Peter Stefan Nussbaum, Rådgiver NTNU
Agenda	• Demo av nåværende tjeneste • Gjennomgang av prosjektplan og internsystemer sålangt

Notater	• Mye kartlegging/planlegging for overgang til dynamisk kalkulator • Bra å bruke designer hos Skogkurs for innspill; vis dette i rapporten • Observasjoner fra uformelle møter inn i rapporten ?

Møtereferat - Veiledningsmøte

Dato	**21 Feb 2024**
Tilstede	• Øystein Qvigstad • André Marthinsen • Even Stetrud • Paul Schiager • Peter Stefan Nussbaum, Rådgiver NTNU
Agenda	• Demo av nåværende tjeneste • Gjennomgang av prosjektplan og internsystemer sålangt
Notater	• Få med egne vurderinger i rapporten rundt diskusjoner som skjer innad i gruppen ○ Vi blir i stor grad vurdert på evnen til å reflektere over ulike valg/diskusjoner

Ekstraordinære møter

Dato	**23 Feb 2024**
Tilstede	• Øystein Qvigstad • André Marthinsen • Even Stetrud • Paul Schiager • Peter Stefan Nussbaum, Rådgiver NTNU
Agenda	• Diskusjon av logo og profil • Avklare omfang mtp prosess videre •
Notater	

Sprint 7

Start dato	**28 Feb 2024**
Slutt dato	12 Mar 2024

Møtereferat - Produkt Backlog Raffinering

Dato	**03 Mar 2024**
Tilstede	• Øystein Qvigstad • André Marthinsen • Even Stetrud • Paul Schiager • Mikael Fønus, Skogkurs • Brede Lauten, Skogkurs
Agenda	• Vise middelstamme på inputfelt? • Vise antall trær pr m3
Forretningsmål	Forrige sprint: Se formelen fungere i en nettleser

Notater	• Endre til placeholdertekst; lite brukervennlig at man må fjerne 0 manuelt
	• Grafen beholdes, Skogkurs mener den er svært nyttig
	• Detalj: “Avstand klopplegging” endres til “Distanse”
	• Sette sammen felter med enhet og antall i samme felt ?
	• Revidere infotekster per felt
	• Landingsside for alle kalkulatorer opprettes av Skogkurs, med link til hver kalkulator (linker ikke til flere kalkulatorer fra forside i appen).
	• Må lande på hvilken funksjonalitet som skal prioriteres
	  ◦ Hogstformer - eget møte med Nibio?
	  ▪ Formel fra Helmer, tilleggskostnader.
	• Setter frist 22. mars for ønsker om funksjonalitet
	  ◦ Alle innspill til statisk versjon ferdig til denne dato
	  ◦ Mikael skriver tekster som skal legges inn i infosider (forskring, tallgrunnlag, info til parametere…) - kommer som som Word-dokument
	• Evaluering av prosjektet med Skogkurs når det nærmer seg slutten
	  ◦ Samarbeid med oppdragsgiver
	  ◦ Eventuelt flere NTNU-prosjekter senere?
	• Statisk kalkulator er delt ut i branch “release-v1.1”

Møtereferat - Sprint Planning

Dato	**28 Feb 2024**
Tilstede	• Øystein Qvigstad • André Marthinsen • Even Stetrud • Paul Schiager

Agenda	• Opprettelse/omprioritering av utvikler-spesifikke PBI-er
	• Tilgjengelig tid estimeres for neste sprint basert på forrige sprint og timeplan
	• PBI-er tidsestimeres av utviklerne i fellesskap
	• PBI-ere velges ut i tråd med prioritering og tid
	  ◦ Utviklere plukker fra de utvalgte oppgavene basert på lyst og kompetanse.
	  ◦ Der mulig deles utviklere inn to og to med et gitt ansvarsområde for å promotere par-programmering og kunskapsdeling
	  ◦ Hver utvikler deler sine PBI-er opp i “tasks” og estimerer tid for gjennomføring.

Notater	• Fokus på dynamisk formel og rapportskriving

Team kapasitet	**Prosent kapasitet (100% = 36 timer = 12 SP)**
Øystein Qvigstad	130%
André Marthinsen	100%
Even Stetrud	100%
Paul Schiager	100%
Estimert kapasitet	51

Møtereferat - Sprint Review

Dato	**27 Feb 2024**

Dato	**12 Mar 2024**
Til stede	• Øystein Qvigstad • André Marthinsen • Even Stetrud • Paul Schiager • Jon Sigurd Leine • Jon Harby • Erling Perstolen • Bjørn Lauritzen • Kjetil Finsrud • Brede Lauten • Helmer Belbo • Mikael Fønhus
Agenda	• Presentere oppdatert versjon • Bestemme/avgrense funksjonalitet

Notater	• Resultatlagring: Ikke helt ferdig
	• Fjernet kostnadsdrivere fra graf (resultatside)
	• Nytt ikon i fane - sjekk om implementert
	• Lite respons fra eksterne
	  ◦ Greit inntrykk, intuitivt med resultat på samme side som input
	• Tydeligere melding om oppdatering, eks. modal ?
	• Svenske forhold: Man manipulerer timeprisen fordi man reelt ligger nærmere G0 enn G15-timer.
	• Forskningsgrunnlaget er basert på G0-timer og omregnet til G15.
	  ◦ Helmer undersøker dette videre (mulig lassbærer og hogstmaskin benytter ulike)
	• Kan få ut data i G0 og G15 fra hogstmaskin
	• Ekstra møte neste uke, for innspill
	• Bedre layout nå, men:
	  ◦ Footer eller annen info som knytter kalkulatoren opp mot Skogkurs
	  ◦ Kan også legge til en “Om”-side i hamburgermenyen
	• URL endres til mer beskrivende (eks. kostnadskaklulator)
	• Merete setter opp forlag om farger
	• Logo endres til SVG
	• Link i logo til forside

Møtereferat - Sprint Retrospective

Dato	**12 Mar 2024**
Til stede	• Øystein Qvigstad • André Marthinsen • Even Stetrud • Paul Schiager
Agenda	• Generelt: Hva gikk bra? Hva gikk dårlig? Hva kan vi forbedre?

Notater	• God gjennomføring av SCRUM • Vi må ta en avgjørelse for hvordan vi skal prioritere dynamisk/statisk versjon, og forholde oss til andre ønsker fra Skogkurs/interessenter ○ • Vi nedprioriterer andre hogstformer på bakgrunn av at vi ønsker å bruke mer tid på dynamisk innlesning, og at data/-formelgrunnlaget samt detaljene rundt dette er uklare og vi ikke oppfattet det som så omfattende.

Fortsett med	Start med	Slutt med

Møtereferat - Veiledningsmøte

Dato	06 Mar 2024
Tilstede	• Øystein Qvigstad • André Marthinsen • Even Stetrud • Paul Schiager • Peter Stefan Nussbaum, Rådgiver NTNU
Agenda	• Diskusjon om hogstformer (ref. R-script fra Nibio (Helmer Belbo))

Notater	Oppdatering av kalkulator:
	• Ekstra steg ved input for å velge lukket/åpen hogst
	• Kan bli en utfordring med indeks på sider?
	  ○ Kan gi inputs statisk indeks, og heller legge in default verdier?
	Spørsmål til Helmer:
	• Hvordan skal funksjonen vi har fått fungere med % av gran og løv
	  ○ Hvorfor? Er det for uttaksgrad? Middelsstamme?
	  ○ Resten er furu, bør det gjøres mer eksplisitt?
	• Har han noe test data, helst i csv, excel eller lignende format.
	• Hvilken skogtype er formelen basert på? I kalkulatoren påvirker skogtype resultatet i veldig stor grad.
	• Avklare utforming av parametere/kostnadsdrivere på siden.
	• Kalkulasjon for lassbærer må undersøkes videre.

Møtereferat - Produkt Backlog Raffinering

Dato	11 Mar 2024
Tilstede	• Øystein Qvigstad • André Marthinsen • Even Stetrud • Paul Schiager • Mikael Fønus, Skogkurs • Brede Lauten, Skogkurs
Agenda	• Oppfølgningspunkter Brede • Tekstbeskrivelser fra Mikael • Hovedfunn fra testrapport (vedlegg, seksjon «Konklusjon») • Forenklet kalkulatorversjon, utforming/behov • Fremside/infoside, hva skal brukeren møte? • «Lukket hogstform» – samle tanker igjen ift. spørsmål til Helmer
Forretningsmål	

Notater	• Punkter fra Brede:
	  o Filtrere kostnadsdrivere i graf - utført
	  o Konstant y-akse i graf: Kan være utfordrende
	  o Instruksjonsfilm for kalkulatoren - Skogkurs gjør dette selv
	  o Tekst til infoknapper - fått dokument fra Mikael
	  o Forenklet versjon:
	  ▪ Skjuler felter i nåværende versjon
	  ▪ Samle felter på en side?
	  ▪ Avanserte felter må falle tilbake til defaultverdi (som er typisk for skogtypen?)
	  o Hogstformer:
	  ▪ Tallene kun basert på lavlandsskog
	  ▪ Hogstform som overordnet valg - separate kalkulatorer
	  o Driftsform og ordvalg:
	  ▪ Legge inn uttaksstyrke - kun når man har lukket/rydding som hogstform
	• Tekstbeskrivelser fra Mikael
	  o Lagt inn de fleste endringer
	  o Antall maskinflytt
	  ▪ Kun flytting på bil, eller også hjulkjøring av maksin mellom oppdrag
	  ▪ Flytting av annet utstyr? Løses med informasjon på parameterne.
	  ▪ Antall som desimaltall - vi prøver å legge inn mulighet for dette.
	• Brukertesting
	  o Skifte til søylediagram for produktivitet, beholde 50 som std, men utvide y-aksen hvis resultat blir over 50
	  o Lagring: I nettleser foreløpig
	  ▪ Mister lagring om man nullstiller nettleser (men kan skje i andre situasjoner også)
	  ▪ Kommunisere at man må sende kalkulasjon for å bruke på flere enheter
	  ▪ Lagre/delefunksjon på felles knapp?
	  o Om kostnad:
	  ▪ Vise totalkostnad for hogsten - ekstra kolonne i “Kostnadsoverslag”
	  ▪ Total for alt nederst
	  ▪ Knapp for å veksle mellom totalkostnad og kostnad/kubikkmeter i Kostnad-rute? Ev. i kostnadsoverlsag-rute?
	  ▪ Det ble kostnadsoverslag.
	  o Om nøyaktighetsnivå i kalkulatoren
	  ▪ Enkel å bruke kontra nøyaktig resultat?
	  ▪ “Godt nivå nå” - Brede
	  ▪ Det viktigste er nøyaktighet

	- Om tilleggsparametere - Tre pr kubikk - ikke aktuelt å legge inn. Det blir uansett beregnet av volum pr dekar og treantall pr dekar. - Forenklet versjon: Forslag om å kunne fylle inn treantall per kubikk. Defaultverdier må velges basert på - Om G15: oppdatert informasjon lagt inn. Infoknapp eller hover på resultatvisningen også, for å oppklare hva G15 er. - Om forenkla versjon - Om felt skal skjules så bør deres verdier stilles til en defaultverdi - Landingsside med informasjon og valg mellom enkel/avansert/lukket hogst
Dato	**12 Mar 2024**
Tilstede	• Øystein Qvigstad • André Marthinsen • Even Stetrud • Paul Schiager • Helmer • Brede • Mikael
Agenda	• Diskusjon om ny formel med Helmer og Brede • Hvordan skal funksjonen vi har fått fungere med % av gran og løv • Hvorfor? Er det for uttaksgrad? Middelsstamme? • Resten er furu, bør det gjøres mer eksplisitt? • Har han noe test data, helst i csv, excel eller lignende format. • Hvilken skogtype er formelen basert på? I kalkulatoren påvirker skogtype resultatet i veldig stor grad. • Avklare utforming av parametere/kostnadsdrivere på siden. • Kalkulasjon for lassbærer må undersøkes videre.

Notater	• Formel for Blednings- og småflatehogst (patch).
	  ◦ Formel for hugstmaskin, bruker formel for lassbærer fra tynningshogst
	  ◦ Testdata
	  ▪ Sendes over til oss
	• Formel for tynningshogst kommer senere, der kommer prosent gran og løv inn.
	  ◦ Formel for både lassbærer og hugstmaskin
	  ◦ Det er for denne formelen vi bruker andel gran og løv.
	  ▪ Andelen furu er det som blir igjen
	  ▪ Andel gran påvirker sikt
	  ▪ Løv: raskere å prosessere gran og fur en løv
	• Prioritering å få inn de nye hogstformene
	• Dag Fjeld: Lavlandsskog
	  ◦ Legge inn advarsel om at de nye formelene kun er for lavlandsskog
	  ◦ Kjøreforhold trenger ikke å endres
	  ◦ Maskin trenger ikke å endres
	  ◦ Lassbærerstørrelse er ofte mindre ved tynning(?) - bare info
	  ◦ Skrive info om at hugstmaskin er stikkveigående(?)

Sprint 8

Start dato	**13 Mar 2024**
Slutt dato	26 Mar 2024

Møtereferat - Produkt Backlog Raffinering

Dato	**18 Mar 2024**
Tilstede	• Øystein Qvigstad • André Marthinsen • Even Stetrud • Paul Schiager • Mikael Fønus, Skogkurs • Brede Lauten, Skogkurs
Agenda	•
Forretningsmål	

Notater	Presentert planen om å legge nye hogstformer på is og fokusere på den dynamiske kalkulatoren, greit mottatt. Ønske om videreutvikling etter ferdig bachelor. Forenklet kalkulator: Blir ikke forskningsbasert. Vanskelig å sette fornuftige defaultverdier. Settes på vent. Møte med Skogkurs om innkomne ønsker etter møte med interessenter på fredag. Viser også demonstrasjon på fredag av det vi har i dynamisk versjon.

Møtereferat - Sprint Planning

Dato	**13 Mar 2024**
Tilstede	• Øystein Qvigstad • André Marthinsen • Even Stetrud • Paul Schiager

Agenda	• Opprettelse/omprioritering av utvikler-spesifikke PBI-er
	• Tilgjengelig tid estimeres for neste sprint basert på forrige sprint og timeplan
	• PBI-er tidsestimeres av utviklerne i fellesskap
	• PBI-ere velges ut i tråd med prioritering og tid
	  ○ Utviklere plukker fra de utvalgte oppgavene basert på lyst og kompetanse.
	  ○ Der mulig deles utviklere inn to og to med et gitt ansvarsområde for å promotere par-programmering og kunskapsdeling
	  ○ Hver utvikler deler sine PBI-er opp i “tasks” og estimerer tid for gjennomføring.

Notater	

Team kapasitet	**Prosent kapasitet (100% = 36 timer = 12 SP)**
Øystein Qvigstad	150%
André Marthinsen	150%
Even Stetrud	90%
Paul Schiager	100%
Estimert kapasitet	59

Møtereferat - Sprint Review

Dato	22 Mar 2024
Til stede	• Øystein Qvigstad • André Marthinsen • Even Stetrud • Paul Schiager • Jon Sigurd Leine • Jon Harby • Erling Perstolen • Bjørn Lauritzen • Kjetil Finsrud • Brede Lauten • Helmer Belbo • Mikael Fønhus
Agenda	• Presentere oppdatert versjon • Bestemme/avgrense funksjonalitet • Diskutere ønsker fra eksterne

Notater	• Lukket hogst satt på vent
	• Lite kommentarer tilbake fra Allskog, antar at det er positivt
	  ◦ API etterspurt av Allskog, regner med å få implementert det
	  ◦ Tall for produktivitet ønsket for hogstmaskin - planlagt endret til stolpediagram
	• Mikael: Kommentarer til grafen på produktivitet, endres til stolpediagram
	• Stadig mer lukket hogst, så det er av interesse at det blir implementert
	  ◦ Noe usikkert datagrunnlag som må undersøkes nærmere av Skogkurs
	  ◦ Nibio holder på å innhente tall fra GM
	• Lite tilbakemelding fra GM, utenom lukket hogst (enig i stolpediagram som er planlagt)
	• Forklaring på G15 må inkluderes
	• Display node: Infotext
	• Helmer blir gjerne med på å teste bruk av editoren
	• Sjekk språk som er default. Edge gir tilbud om å oversette til norsk.
	• Ordlyd i overskrift på forside: Endres til “beregn produktivitet og kostnader ved skogsdrift”
	• Undersøke om Skogkurs kan bruke kalkulatoren i kursvirksomhet før vi leverer - spør Nussbaum om dette!

Møtereferat - Sprint Retrospective

Dato	02 Apr 2024
Til stede	• Øystein Qvigstad • André Marthinsen • Even Stetrud • Paul Schiager
Agenda	• Generelt: Hva gikk bra? Hva gikk dårlig? Hva kan vi forbedre?
Notater	• Fint at vi hadde tydelig frist til interessentene, slik at de hadde tid til å tenke. • Rutiner rundt å skrive opp feedback i PB, fremfor å implementere dem med en gang. Kanban vs Scrum

Fortsett med	Start med	Slutt med
		Dra ut standup-møtene til lenger enn 15 minutter

Møtereferat - Intern Review

Dato	02 Apr 2024
Tilstede	• Øystein Qvigstad • André Marthinsen • Even Stetrud • Paul Schiager
Agenda	•
Notater	• Gikk gjennom text editor in graph editor • Gikk gjennom autentisering

Møtereferat - Veiledningsmøte

Dato	13 Mar 2024

Tilstede	• Øystein Qvigstad • André Marthinsen • Even Stetrud • Paul Schiager • Peter Stefan Nussbaum, Rådgiver NTNU
Agenda	• Beslutning om å sette ny bruksfunksjonalitet på vent
Notater	Nye hogstformer lagt på is. Prioriterer dynamisk kalkulator (større akademisk interesse). Støttes av veileder. Gjort mye detaljarbeid rundt design; kunne kanskje fokusert mer på hogstformer hvis formler hadde vært tigjengelig tidligere? Refleksjoner: • Ukjente tekniske utfordringer • Interessenters vs vår egen interesse • Ulik kultur skog/IT Kan ha en refleksjonsdel for hvert kapittel, og så en mer overordnet refleksjonsdel til slutt (evt. henvisninger tilbake til tidligere refleksjoner). Bærekraft: • Dele inn i ulike områder teknisk/sluttprodukt

Møtereferat - Veiledningsmøte

Dato	**20 Mar 2024**
Tilstede	• Øystein Qvigstad • André Marthinsen • Even Stetrud • Paul Schiager • Peter Stefan Nussbaum, Rådgiver NTNU
Agenda	• Demo av nåværende tjeneste • Gjennomgang av prosjektplan og internsystemer sålangt

Notater	• Kort gjennomgang av mandagsmøtet med Brede og Mikael hvor det ble besluttet å satse på utvikling av den dynamiske versjonen • Viste ny stoplediagram i admin-consoll

Møtereferat - Produkt Backlog Raffinering

Dato	**11 Mar 2024**
Tilstede	• Øystein Qvigstad • André Marthinsen • Even Stetrud • Paul Schiager • Mikael Fønus, Skogkurs • Brede Lauten, Skogkurs
Agenda	• Oppfølgningspunkter Brede • Tekstbeskrivelser fra Mikael • Hovedfunn fra testrapport (vedlegg, seksjon «Konklusjon») • Forenklet kalkulatorversjon, utforming/behov • Fremside/infoside, hva skal brukeren møte? • «Lukket hogstform» – samle tanker igjen ift. spørsmål til Helmer
Forretningsmål	
Notater	

Dato	**12 Mar 2024**
Tilstede	• Øystein Qvigstad • André Marthinsen • Even Stetrud • Paul Schiager • Helmer • Brede • Mikael

Agenda	• Diskusjon om ny formel med Helmer og Brede • Hvordan skal funksjonen vi har fått fungere med % av gran og løv • Hvorfor? Er det for uttaksgrad? Middelsstamme? • Resten er furu, bør det gjøres mer eksplisitt? • Har han noe test data, helst i csv, excel eller lignende format. • Hvilken skogtype er formelen basert på? I kalkulatoren påvirker skogtype resultatet i veldig stor grad. • Avklare utforming av parametere/kostnadsdrivere på siden. • Kalkulasjon for lassbærer må undersøkes videre.
Notater	

Sprint 9

Start dato	**03 Apr 2024**
Slutt dato	16 Apr 2024

Møtereferat - Produkt Backlog Raffinering

Dato	**08 Apr 2024**
Tilstede	• Øystein Qvigstad • André Marthinsen • Even Stetrud • Paul Schiager • Mikael Fønus, Skogkurs • Brede Lauten, Skogkurs
Agenda	•
Forretningsmål	
Notater	

Møtereferat - Sprint Planning

Dato	**03 Apr 2024**

Tilstede	• Øystein Qvigstad • André Marthinsen • Even Stetrud • Paul Schiager

Agenda	• Opprettelse/omprioritering av utvikler-spesifikke PBI-er
	• Tilgjengelig tid estimeres for neste sprint basert på forrige sprint og timeplan
	• PBI-er tidsestimeres av utviklerne i fellesskap
	• PBI-ere velges ut i tråd med prioritering og tid
	  ◦ Utviklere plukker fra de utvalgte oppgavene basert på lyst og kompetanse.
	  ◦ Der mulig deles utviklere inn to og to med et gitt ansvarsområde for å promotere par-programmering og kunskapsdeling
	  ◦ Hver utvikler deler sine PBI-er opp i “tasks” og estimerer tid for gjennomføring.

Notater	• Paul er ny scrum-master

Team kapasitet	**Prosent kapasitet (100% = 36 timer = 12 SP)**
Øystein Qvigstad	150%
André Marthinsen	100%
Even Stetrud	100%
Paul Schiager	100%
Estimert kapasitet	54

Møtereferat - Sprint Retrospective

Dato	**16 Apr 2024**
Til stede	• Øystein Qvigstad • André Marthinsen • Even Stetrud • Paul Schiager
Agenda	• Generelt: Hva gikk bra? Hva gikk dårlig? Hva kan vi forbedre?
Notater	Bytte av scrum master har gått bra.

Fortsett med	Start med	Slutt med

Møtereferat - Intern Review

Dato	16 Apr 2024
Tilstede	• Øystein Qvigstad • André Marthinsen • Even Stetrud • Paul Schiager
Agenda	•
Notater	Til neste sprint: Kommentere sikkerhet, spesielt i app engine

Møtereferat - Veiledningsmøte

Dato	03 Apr 2024
Tilstede	• Øystein Qvigstad • André Marthinsen • Even Stetrud • Paul Schiager • Peter Stefan Nussbaum, Rådgiver NTNU
Agenda	• Formatering av latex dokument • Spørre om tidligere bruksrettigheter
Notater	• Nussbaum sjekket med Tom, noe endring av formatering er ok. Sjekk også på biblioteket; hvilke endringer kan vi gjøre på template? • Nussbaum sjekker med Tom ang. bruksrettigheter.
	Fokus på rapport etterspørres.

Møtereferat - Veiledningsmøte

Dato	11 Apr 2024

Tilstede	• Øystein Qvigstad • André Marthinsen • Even Stetrud • Paul Schiager • Peter Stefan Nussbaum, Rådgiver NTNU
Agenda	

Notater	• Tenk over hva vi sa i prosjektplanen - målsetting
	  ◦ I hvilken grad er målene oppnådd? Hvorfor/hvorfor ikke?
	• Mer fokus på rapport!
	• Kommukasjon internt
	• Workshop positivt
	  ◦ Definer forventet utbytte - for oss og oppdragsgiver
	  ◦ Formål
	  ◦ Bughunt på forhånd?
	  ◦ Beskrive brukerne som deltar
	• Setter frist for rapport-utkast til 12. mai
	• Vurder å få utenforstående til å gjennomgå hele rapporten

Møtereferat - Produkt Backlog Raffinering

Dato	**08 Apr 2024**
Tilstede	• Øystein Qvigstad • André Marthinsen • Even Stetrud • Paul Schiager • Mikael Fønus, Skogkurs • Brede Lauten, Skogkurs
Agenda	• Tidlig bruk av kalkulatoren • Skjermleser • Status: Admin-konsoll og rapport i fokus • Dynamisk: Moduler mangler • Frontend:
Forretningsmål	

Notater	• Skriver forslag til avtale om tidlig bruk - kurs etter 1. mai • Frontend i denne sprinten • Bruke verktøyet - må kunne matte. Formel-kompetanse. Skogkurs har kometanse på dette. • Opplæring på dynamisk opprettelse av formler - Brede, Mikael, Kjetil, kanskje flere? Helmer? Sent i prosessen - så mest mulig er ferdig. Kjøre dette som en slags brukertest - Mandag 6. mai 9:30-11:30 • kostnadskalkulator.skogkurs.no ny URL • Deployment av editor på samme måte som API • Ukentlige møter reduseres - planlegg i løpet av uka

Dato	
Tilstede	• Øystein Qvigstad • André Marthinsen • Even Stetrud • Paul Schiager • Helmer • Brede • Mikael
Agenda	•
Notater	

Sprint 10

Start dato	**17 Apr 2024**
Slutt dato	29 Apr 2024

Møtereferat - Produkt Backlog Raffinering

Dato	**21 Apr 2024**
Tilstede	• Øystein Qvigstad • André Marthinsen • Even Stetrud • Paul Schiager • Mikael Fønus, Skogkurs • Brede Lauten, Skogkurs
Agenda	•
Forretningsmål	
Notater	UTGÅR?

Møtereferat - Sprint Planning

Dato	**17 Apr 2024**

Tilstede	• Øystein Qvigstad • André Marthinsen • Even Stetrud • Paul Schiager

Agenda	• Opprettelse/omprioritering av utvikler-spesifikke PBI-er
	• Tilgjengelig tid estimeres for neste sprint basert på forrige sprint og timeplan
	• PBI-er tidsestimeres av utviklerne i fellesskap
	• PBI-ere velges ut i tråd med prioritering og tid
	  ◦ Utviklere plukker fra de utvalgte oppgavene basert på lyst og kompetanse.
	  ◦ Der mulig deles utviklere inn to og to med et gitt ansvarsområde for å promotere par-programmering og kunskapsdeling
	  ◦ Hver utvikler deler sine PBI-er opp i “tasks” og estimerer tid for gjennomføring.

Notater	

Team kapasitet	**Prosent kapasitet (100% = 36 timer = 12 SP)**
Øystein Qvigstad	120%
André Marthinsen	100%
Even Stetrud	75%
Paul Schiager	90%
Estimert kapasitet	385% = 46 SP

Møtereferat - Sprint Retrospective

Dato	**30 Apr 2024**
Til stede	• Øystein Qvigstad • André Marthinsen • Even Stetrud • Paul Schiager
Agenda	• Generelt: Hva gikk bra? Hva gikk dårlig? Hva kan vi forbedre?
Notater	Prosessmessig lite endringer

Fortsett med	Start med	Slutt med
	Koordinere rapportskriving for å sørge for at alle deler behandles	

Møtereferat - Intern Review

Dato	
Tilstede	• Øystein Qvigstad • André Marthinsen • Even Stetrud • Paul Schiager
Agenda	•
Notater	

Møtereferat - Veiledningsmøte

Dato	**17 Apr 2024**
Tilstede	• Øystein Qvigstad • André Marthinsen • Even Stetrud • Paul Schiager • Peter Stefan Nussbaum, Rådgiver NTNU
Agenda	•
Notater	• Finn noen til å lese rapporten eller deler av den • Redegjør for hvorfor rapporten er skrevet som den er gjort? • Skriv noe om vurdering av maskinlæring som løsningsalternativ • Målgruppe for produktet bør skrives, målgruppe for rapporten går frem av rapporten

Møtereferat - Veiledningsmøte

Dato	**30 Apr 2024**

Tilstede	• Øystein Qvigstad • André Marthinsen • Even Stetrud • Paul Schiager • Peter Stefan Nussbaum, Rådgiver NTNU
Agenda	

Notater	• Lest gjennom rapport; kommenterer at det bærer preg av å være uferdig
	• Refleksjoner kan gjerne legges inn etter hvert
	• Forslag om summary i hvert kapittel (ikke krav)
	  ◦ Intro for hvert hovedkapittel
	  ◦ Konklusjon eller oppsummering (viktigste takeaway fra kapittel, 1/4 side)
	• Bruk gjerne kryssreferanser
	• Homogen skrivestil
	• Zoom ut så leseren forstår helheten
	• For lang rapport?
	  ◦ Unngå gjentakelser
	  ◦ Refleksjonsnotater kan ofte ha litt mye fyllstoff (presisjon!)
	• Bruk gjerne figurer (bryter opp teksten, og hjelper leseren)

Sprint 11

Start dato	**01 May 2024**
Slutt dato	20 May 2024

Møtereferat - Produkt Backlog Raffinering

Dato	**06 May 2024**
Tilstede	• Øystein Qvigstad • André Marthinsen • Even Stetrud • Paul Schiager • Mikael Fønus, Skogkurs • Brede Lauten, Skogkurs
Agenda	•
Forretningsmål	
Notater	UTGÅR

Møtereferat - Sprint Planning

Dato	**01 May 2024**

Tilstede	• Øystein Qvigstad • André Marthinsen • Even Stetrud • Paul Schiager

Agenda	• Opprettelse/omprioritering av utvikler-spesifikke PBI-er
	• Tilgjengelig tid estimeres for neste sprint basert på forrige sprint og timeplan
	• PBI-er tidsestimeres av utviklerne i fellesskap
	• PBI-ere velges ut i tråd med prioritering og tid
	  ◦ Utviklere plukker fra de utvalgte oppgavene basert på lyst og kompetanse.
	  ◦ Der mulig deles utviklere inn to og to med et gitt ansvarsområde for å promotere par-programmering og kunskapsdeling
	  ◦ Hver utvikler deler sine PBI-er opp i “tasks” og estimerer tid for gjennomføring.

Notater	

Team kapasitet	**Prosent kapasitet (100% = 54 timer = 18 SP)**
Øystein Qvigstad	150%
André Marthinsen	100%
Even Stetrud	100%
Paul Schiager	100%
Estimert kapasitet	450% / 81 SP

Møtereferat - Sprint Retrospective

Dato	
Til stede	• Øystein Qvigstad • André Marthinsen • Even Stetrud • Paul Schiager
Agenda	• Generelt: Hva gikk bra? Hva gikk dårlig? Hva kan vi forbedre?
Notater	Prosessmessig lite endringer

Fortsett med	**Start med**	**Slutt med**

Dato	06 May 2024
Tilstede	• Øystein Qvigstad • André Marthinsen • Even Stetrud • Paul Schiager • Peter Stefan Nussbaum, Rådgiver NTNU
Agenda	•
Notater	Fysisk møte utgår, tar opp eventuelle ting på mail

Møtereferat - Veiledningsmøte

Dato	15 May 2024
Tilstede	• Øystein Qvigstad • André Marthinsen • Even Stetrud • Paul Schiager • Peter Stefan Nussbaum, Rådgiver NTNU
Agenda	

Notater	• Mer fokus på drøftinger!
	• Konsekvent med kapittelinndeling og formatering + intro til kapittel
	• Avslutte hvert kapittel (kort), og vise at kapitler henger sammen
	• Hovedkapitler: Teknisk, design, implementasjon, testing
	• Diskusjon og kritikk til oppgaven veldig viktig!
	  ◦ Hva gjorde vi bra?
	  ◦ Hva kunne vi gjort annerledes?
	  ◦ Ta tråden tilbake - har vi oppnådd mål, hva har endret seg underveis?
	  ◦ Hva har vi lært
	• Konklusjon
	  ◦ Kan være fra selve produktet
	  ◦ Samarbeid internt/eksternt
	  ◦ …
	• Drøft muligheter for AI i framtidig versjon
	• Innledning: Beskriv hvordan rapporten skal leses
	  ◦ Diskusjoner underveis i hvert kapittel er ok
	  ◦ Diskuter helhet til slutt

Appendix M

Group Contract

This document outline the expectations/rules for each member of the team.

Gruppekontrakt Bacheloroppgave

Generelle

- Det forventes at man holder seg oppdatert i gjeldene rutiner og regler.
- Man oppfordres til å melde fra hvis det oppstår utfordringer med å forstå eller gjennomføre oppgaver.

Fravær og sykdom

- Fravær skal meldes fra til alle i teamet så tidlig som mulig.
- Ved sykdom holder man seg hjemme for å unngå å smitte andre gruppemedlemmer.

Møter

- Fysisk oppmøte forventes i møtevirksomhet, med unntak av "Standup"-møter.
    - Digitalt oppmøte tillates dersom det avtales på forhånd og det ikke gir negativ innvirkning.
    - Møter utenfor fastsatte tidspunkter eller med kort varsel kan avvikles digitalt.
- Man oppfordres til å dele sine meninger og ideer under møter og diskusjoner.
    - Man skal være respektfull ovenfor andres synspunkter

Frister

- Det forventes at man loggføre arbeidstimer hver dag. Senest i slutten av uken.
- Det forventes at man sjekker kommunikasjonskanaler daglig.
- Det forventes at man informere om arbeid står i fare for å ikke bli fullført innenfor forventet tid.
    - Oppgaver man er tildelt i en sprintperiode forventes å bli fullført innen "Sprint Review" dersom annen beskjed ikke gis på forhånd.
    - I tilfeller hvor arbeid gjentatte ganger ikke blir fullført, vil teamet sammen komme til en enighet om å hvorvidt en advarsel skal gis. Dersom det ikke skjer forbedringer, kan det aktuelle medlemmet risikere å bli ekskludert fra bachelorgruppen.

Øystein Qvigstad 19/1-24

Even Stetrud 24/1-24

Øystein Qvigstad

Even Stetrud

André Marthinsen 19/1-24

Paul Schiager 19/1-24

André Marthinsen

Paul Schiager

Appendix N

Future Work

This document outlines suggestions of future development of the software.

Cost calculator - Future Work

May 20, 2024

1 Suggested changes

Suggested future work, based on the remaining backlog items and new ideas we believe would be useful.

1.1 State updates

There is a bug related to the arrangements of inputs within the same page. All inputs are shown, but not necessarily in the desired order. We expect this to be caused by faulty a faulty state update, and this should be investigated further.

1.2 Performance issues

In the web editor, issues related to performance are apparent. As long as the calculators created are simple, with few nodes, immediate response can be expected. As the calculator grows in complexity, simple operations like editing a text or value may take several seconds. The application appears to hang, while processing happens in the background. The reason for this behavior is that the underlying data structure is reloaded in its entirety on every change.

1.3 Editor usability

We consider the editor to be clear and easy to learn. For a new user starting from scratch, some more guidance could be beneficial. The web editor was created with main functionality as a priority, with limited focus on usability. An effort was made to utilize already created components, to keep the design consistent with the web app. The nodes should therefore be familiar to someone who has used the web app, although there are some constraints that the user should know in order to create a calculator.

To make informed choices for the further development of the web editor, some external feedback would be useful. A process similar to that of the web app could help us develop a user friendly editor as well.

Suggested improvements to correct known problems may include:

- Info/help page, alternatively more explaining text in menus
- Clearly marked menu for "modules"
- Error message if an input is not added to a page (will not be displayed in field client otherwise)
- Mandatory name for inputs, outputs, and displays
- Warning when user tries to overwrite a previously stored calculator
- Undo/redo functionality

1.4 Code refactoring

Some code repetition is present in the code base. Specifically when working with visible react components, this has been a challenge. A refactoring job should be done in this area. Many components contain overlapping functionality. Parts of the code for component A and component B are identical, while the components also have their own specializations. Splitting the common part out into a separate reusable component, and insert into the main component would have made sense in these cases. This has also to some extent been tried, but the graphical library, Rete, in some cases made the operation more difficult than expected. It should be noted that this is an issue only in minor parts of the code base.

1.5 Calculator storage

The default is to store calculations in the browser's local store, with an option to copy or share the link. By using the default storage method, the user will only have access to the calculation on the device used when storing. The following could add more flexibility:

- **Cloud storage:** To allow access independent of device, a cloud service with user login could be used. This would make the storage more robust and available, but would also dictate user management.
- **Document export:** Letting users export the calculation in an app independent format, for example PDF, would facilitate viewing results without involving the web app, for example if the calculation is intended to be part of documentation.

1.6 Access control

The only app component that requires access control is the editor. The access control is set up with only one role. Therefore, everyone who are granted access can do all privileged operations. These operations include creating, updating, and publishing calculators to the web app. No concept of ownership of a calculator is implemented. Neither is there any activity logging. User A may thereby freely edit User B's calculator, and it would be impossible to find out who did the editing. In an improved version, access control with more roles could be implemented. Each new calculator could be owned by the user who created it, with an option of transferring ownership in case employees quit. Suggested roles:

- Test user: Access to create, update, and delete own calculators, and view other's calculators
- Editor: Same as Test user, but also publishing rights for own calculators
- Administrator: Access to edit, publish, and delete all calculators, additionally changing ownership

1.7 Calculator versioning

The web editor is set up with a system for calculator versioning, using a "major.minor.patch" pattern. This is controlled manually by the editor user. The version should be controlled at every save operation, for instance displaying a warning to the user if they are trying to overwrite a specific version. A forced incrementing of the version could be a possibility, but there are use cases where this is not desirable. For instance, if an editor user starts working on a calculator, but does not finish, then the version should not increment the next time the user continues to work on said calculator. A possible solution could be to prompt the user to choose in the storing process to continue on the old version number or increment it. That way, the editor user would still be in control, but would also be forced to make a decision. Automatically deciding which changes should trigger major/minor/patch increments could potentially be a difficult task, as it would depend on the application.

1.8 Toggle for publish/unpublish

Once a calculator is published, the user has no means of unpuublishing it. Nor is there a "delete" button. Adding a toggle button for publishing calculators would allow more flexibility, and is a necessity as more calculators are produced.

1.9 Language support

The web app only supports Norwegian language in the current version. This was a specific wish from Skogkurs based on the impression that most forestry professionals understand Norwegian. As no preference was available for the editor, this was implemented in english. We did explore the possibility of adding switchable language packs, but its implementation was not prioritized. For a future and more generalized version, language packs could be a valuable addition.

1.10 Screen readers

Setting up the web app to be available for screen readers was explored during the project. Bootstrap as CSS framework made this part more challenging than expected, especially setting labels on buttons proved difficult. More research is needed to complete this feature.

1.11 Summary

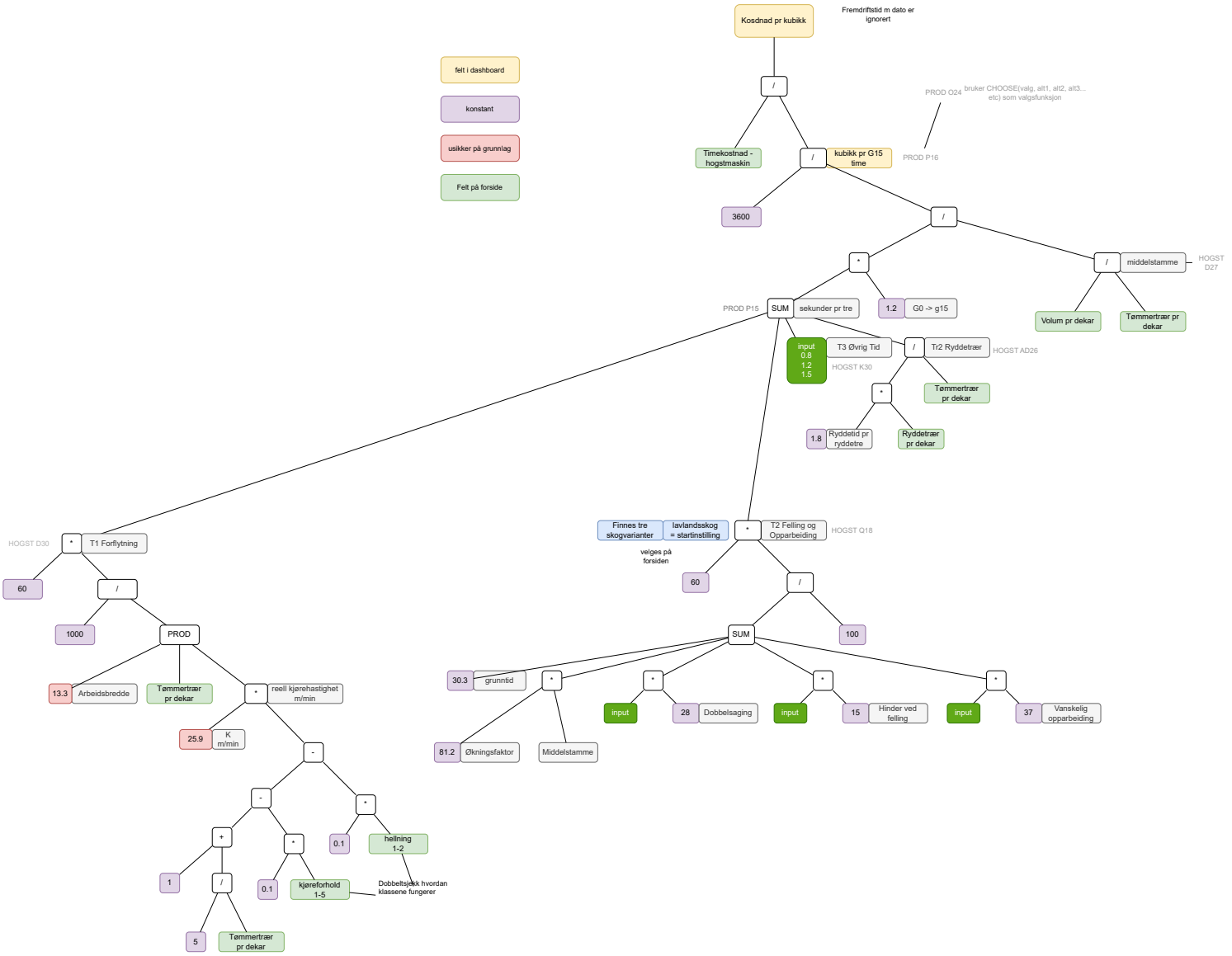
Working on a software product within a set time frame required us to prioritize which functionality to include. The focus has been on getting the main functionality up and running. Especially focusing on usability of the web app, with the largest number of potential users, other components of the software were prioritized lower. This list is meant to highlight many of the features that could be improved or added in later iterations to allow a better user experience.

Appendix O

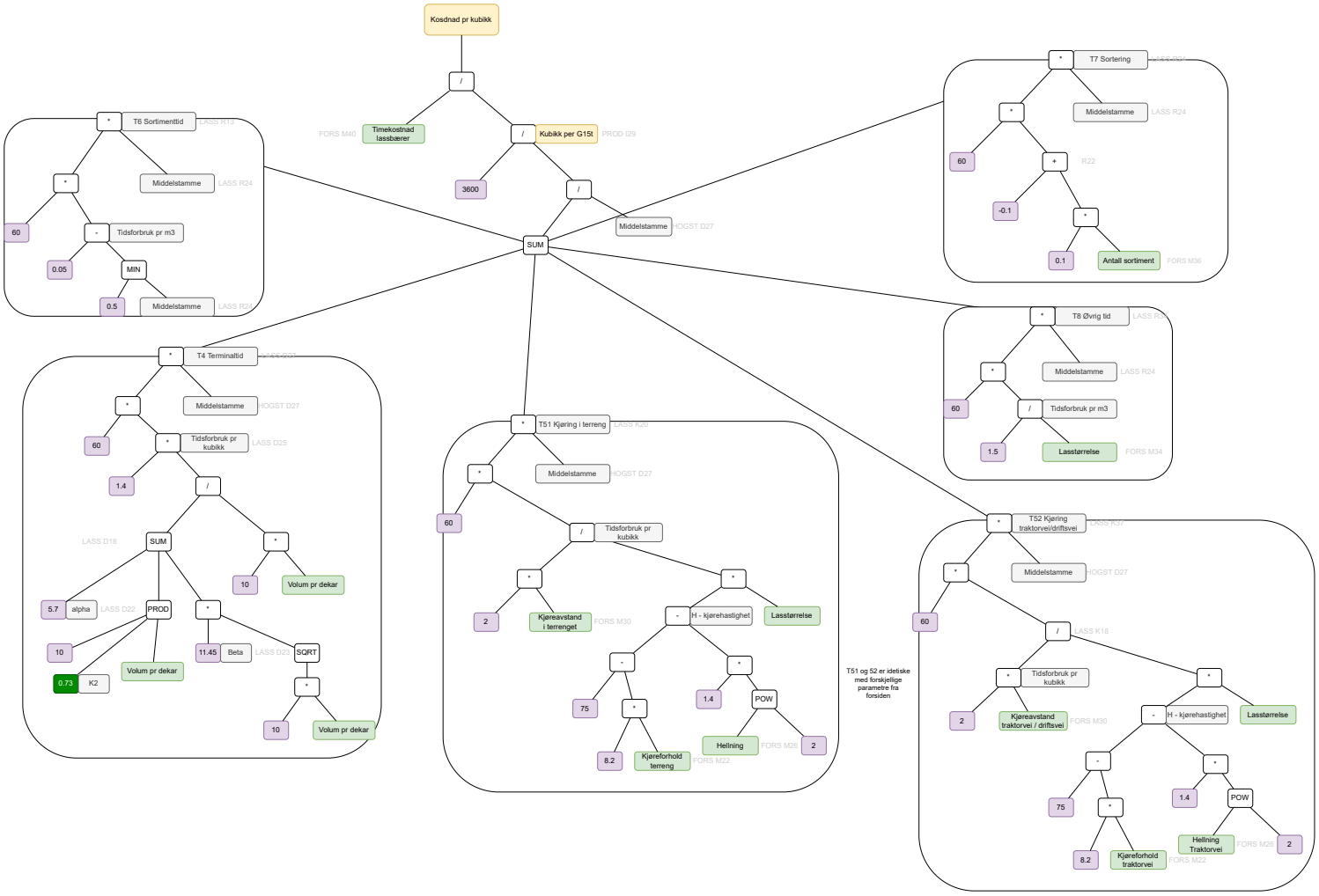
Formula Graph Representation

These diagrams were created early in the project for the purpose of analysing the formulas while also assessing just how large the formulas would be represented as expression trees.

Hogstmaskin



Lassbærer


Kosdnad pr kubikk
/
Timekostnad lassbærer
FORS M40
/
Kubikk per G15t
PROD I29
3600
/
Middelstamme
HOGST D27
SUM
* T6 Sortimenttid LASS K19
* Middelstamme LASS R24
60
- Tidsforbruk pr m3
0.05
MIN
0.5
Middelstamme LASS R24
* T7 Sortering LASS R24
* Middelstamme LASS R24
60
+ R22
-0.1
*
0.1
Antall sortiment FORS M36
* T8 Øvrig tid LASS R25
* Middelstamme LASS R24
60
/ Tidsforbruk pr m3
1.5
Lasstørrelse FORS M34
* T4 Terminaltid LASS D27
* Middelstamme HOGST D27
60
* Tidsforbruk pr kubikk LASS D25
1.4
/
LASS D18
SUM
*
5.7
alpha LASS D22
PROD
*
10
Volum pr dekar
10
Volum pr dekar
11.45
Beta LASS D23
SQRT
0.73
K2
*
10
Volum pr dekar
* T51 Kjøring i terreng LASS K20
* Middelstamme HOGST D27
60
/ Tidsforbruk pr kubikk
*
2
Kjøreavstand i terrenget FORS M30
*
- H - kjørehastighet
Lasstørrelse
-
75
*
8.2
Kjøreforhold terreng FORS M22
*
1.4
POW
Hellning FORS M26
2
T51 og 52 er identiske med forskjellige parametre fra forsiden
* T52 Kjøring traktorvei/driftsvei LASS K17
* Middelstamme HOGST D27
60
/ LASS K18
* Tidsforbruk pr kubikk
2
Kjøreavstand traktorvei / driftsvei FORS M30
*
- H - kjørehastighet
Lasstørrelse
-
75
*
8.2
Kjøreforhold traktorvei FORS M22
*
1.4
POW
Hellning Traktorvei FORS M26
2

Appendix P

Coverage

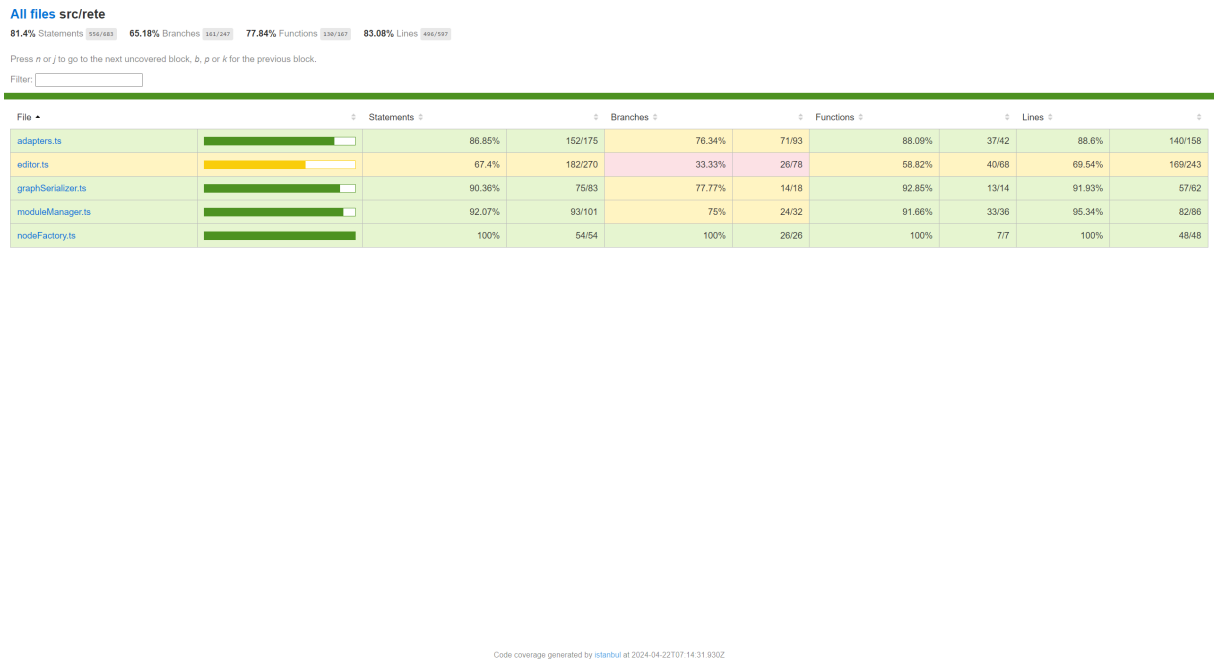


Figure P.1: Coverage of core editor classes

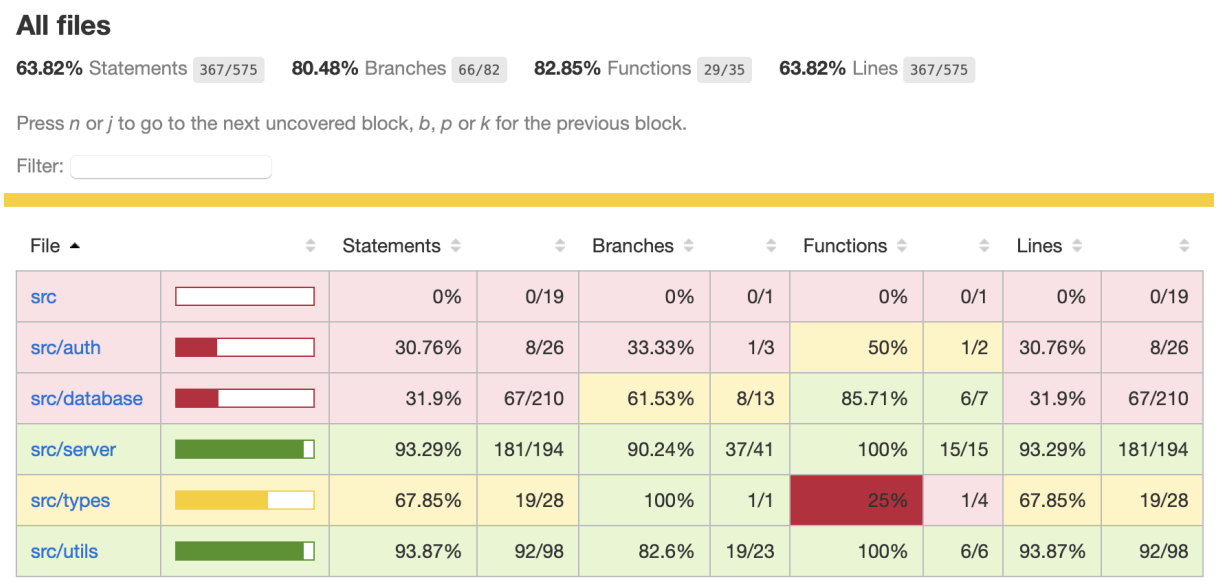
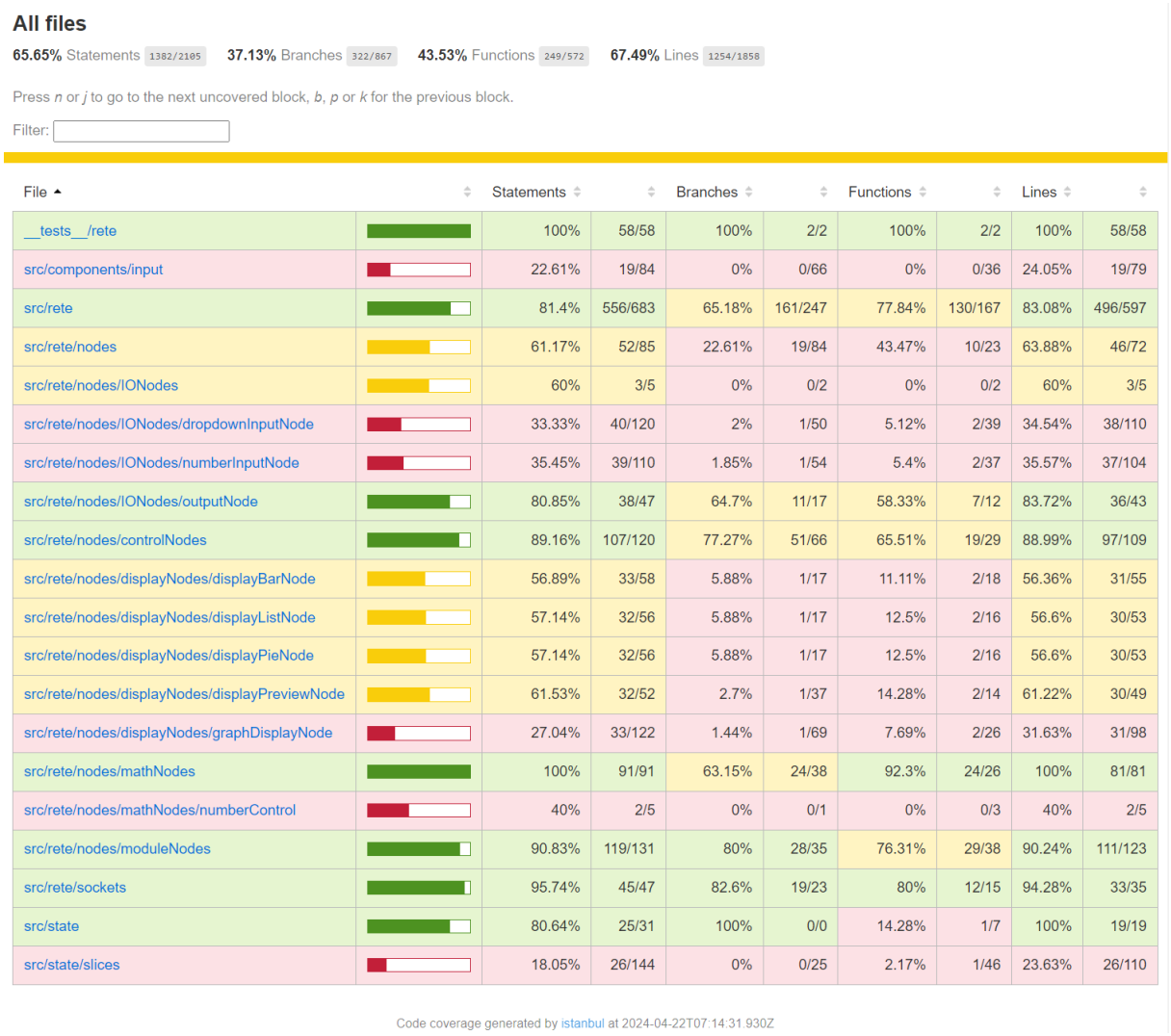


Figure P.2: API coverage



All files

65.65% Statements 1382/2105 **37.13%** Branches 322/867 **43.53%** Functions 249/572 **67.49%** Lines 1254/1858

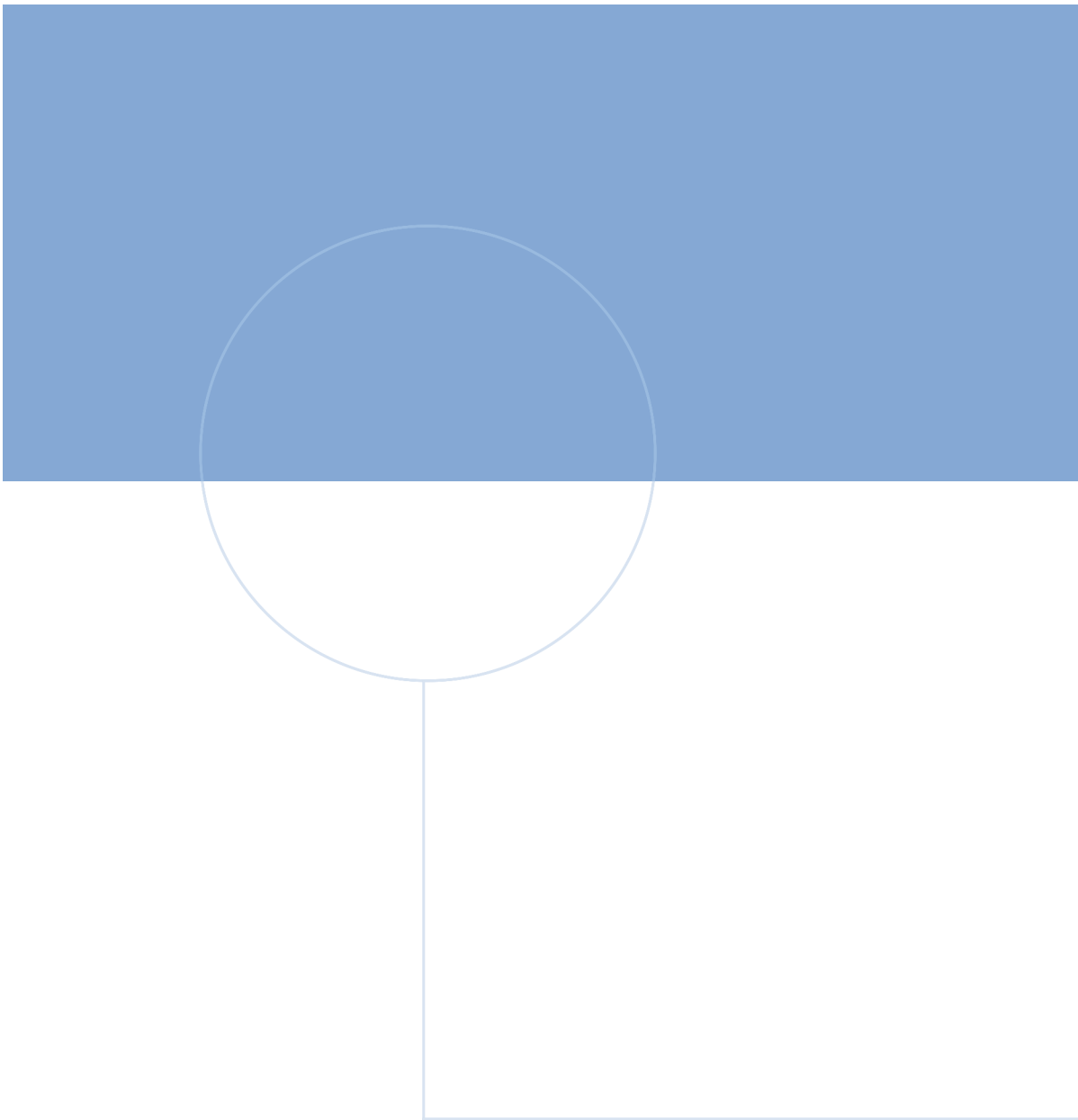
Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

Filter:

File	Statements		Branches		Functions		Lines	
__tests__/rete	100%	58/58	100%	2/2	100%	2/2	100%	58/58
src/components/input	22.61%	19/84	0%	0/66	0%	0/36	24.05%	19/79
src/rete	81.4%	556/683	65.18%	161/247	77.84%	130/167	83.08%	496/597
src/rete/nodes	61.17%	52/85	22.61%	19/84	43.47%	10/23	63.88%	46/72
src/rete/nodes/IONodes	60%	3/5	0%	0/2	0%	0/2	60%	3/5
src/rete/nodes/IONodes/dropdownInputNode	33.33%	40/120	2%	1/50	5.12%	2/39	34.54%	38/110
src/rete/nodes/IONodes/numberInputNode	35.45%	39/110	1.85%	1/54	5.4%	2/37	35.57%	37/104
src/rete/nodes/IONodes/outputNode	80.85%	38/47	64.7%	11/17	58.33%	7/12	83.72%	36/43
src/rete/nodes/controlNodes	89.16%	107/120	77.27%	51/66	65.51%	19/29	88.99%	97/109
src/rete/nodes/displayNodes/displayBarNode	56.89%	33/58	5.88%	1/17	11.11%	2/18	56.36%	31/55
src/rete/nodes/displayNodes/displayListNode	57.14%	32/56	5.88%	1/17	12.5%	2/16	56.6%	30/53
src/rete/nodes/displayNodes/displayPieNode	57.14%	32/56	5.88%	1/17	12.5%	2/16	56.6%	30/53
src/rete/nodes/displayNodes/displayPreviewNode	61.53%	32/52	2.7%	1/37	14.28%	2/14	61.22%	30/49
src/rete/nodes/displayNodes/graphDisplayNode	27.04%	33/122	1.44%	1/69	7.69%	2/26	31.63%	31/98
src/rete/nodes/mathNodes	100%	91/91	63.15%	24/38	92.3%	24/26	100%	81/81
src/rete/nodes/mathNodes/numberControl	40%	2/5	0%	0/1	0%	0/3	40%	2/5
src/rete/nodes/moduleNodes	90.83%	119/131	80%	28/35	76.31%	29/38	90.24%	111/123
src/rete/sockets	95.74%	45/47	82.6%	19/23	80%	12/15	94.28%	33/35
src/state	80.64%	25/31	100%	0/0	14.28%	1/7	100%	19/19
src/state/slices	18.05%	26/144	0%	0/25	2.17%	1/46	23.63%	26/110

Code coverage generated by istanbul at 2024-04-22T07:14:31.930Z

Figure P.3: Overall coverage of editor



NTNU

Norwegian University of
Science and Technology