NTNU
Norwegian University of
Science and Technology

Department of Computer Science (IDI)

IT2901 - Informatics project II

# TRDK03-UN Climate data

*Authors:*
Andreas Winther Moen,
Haakon Gunnarsli,
Benedicte Helen Myrvoll,
Erik Mjaaland Skår,
Per Solibakke,
Vemund Eggemoen,
Øyvind Schjerven.

2021

**Abstract**

This project revolves around the use of the Web Ontology Language (OWL) to systematize sustainable development. In collaboration with Trondheim Municipality, a prototype has been developed and showcased for *The Norwegian Digitalisation Agency*. This report explains the power of the ontologies and why this type of representation method is necessary when attempting to systematize sustainable development. The report is separated into four chapters, starting with a brief introduction in regards to motivation and objectives. Chapter two contains the preliminary research, creating a baseline towards understanding the complexity associated to both sustainability and ontologies. Chapter three provides a technical description of the developed prototype and the essential technologies used to achieve compatibility. The final chapter captures the essence of the development process.

# Contents

# Chapter 1

# Introduction

## 1.1 Motivation

In 2012, the United Nations created the 17 Sustainability Development Goals (SDGs).[1] The goals, along with their 169 targets, are objectives set to achieve a global sustainable society by 2030.[2] To get an overview of these goals and targets, along with the relations between them, is a complex task. Trondheim Municipality would like a better way to store and visualize these relations.

Let us assume that you have been working with electrical vehicles within the municipality, and need a compelling argument to apply for funding to build more charging stations. In your application, you begin to write about how building more charging stations will reduce the usage of fossil fuel and improve air quality. Furthermore, you find peer-reviewed articles that indicate a strong correlation between air quality and the sustainability goal "Good Health and Well Being". However, most people already know that clean air is important for people's health, but funding for the charging stations has not been provided. Is there a way to make a better argument regarding the effects of air quality?

You conduct more research and find sources that claim there is a strong correlation between the sustainability goal "Good Health and Well-Being" and the sustainability goal "No Poverty". According to other sources, there is also a strong correlation between "Good Health and Well-Being" and "Quality Education". As you begin to look into the strong correlation between good health and economic growth, you ask yourself: "If someone else has already researched the implied benefits of good health, why do I have to do all of this work?"

Imagine if all of these correlations were stored in one single database, where all of your research was already defined. Rather than going through an extensive research phase, this database would enable you to efficiently find the necessary connections to argue for more funding.

The system envisioned above can be established by an ontology, which is the basis for this project. A detailed description about ontologies and related topics is provided in chapter 2.

## 1.2 Vision and Goals

Leendert Wienhofen, hereinafter referred to as "the customer", is a representative from Trondheim Municipality who has been trying to convince the municipality to rely more on ontologies since his employment a few years ago. This has, however, proved rather challenging, as it is difficult to convey the potential benefits without a proof of concept. Thus, the idea for this project was formed, as an attempt to demonstrate the possibilities of utilizing ontologies in a new and engaging manner.

The vision for this project is to create a prototype based on the 17 SDGs, which conveys the power of ontologies. Today, Trondheim Municipality's work on sustainable development is stored in spreadsheets and text-based documents which lack connectivity. There is a need for a better system to optimize further progress. If ontologies could be proven to be an efficient way to systematize sustainable development, it could be a major push towards change in other areas as well. To achieve this, the customer envisions a user-friendly web application which allows users to both see and interact with a network graph based on information from an ontology.

The product will ideally help visualize the sustainability goals from a new perspective, using an intuitive interface. If the prototype is successful, there is a potential for further development into a fully fledged tool, which can aid the sustainable development within the municipality in the future. Ultimately, the goal is to use

the principles of ontologies to store and connect all of the municipality's data. The customer also has a dream to push the idea not only locally, but also nationally and hopefully internationally.

## 1.3 Objectives

Initially, the project description was a complex and somewhat vague task of creating an ontology, in which members of the municipality could explicitly relate data to actions and reports associated with sustainable development. The extensive research phase (see chapter 2) helped simplify the complexity, and provided insight towards the customer's vision. As this was meant to be a prototype, it was important to focus on quality over quantity. The overall objective was not to completely systematize and visualize sustainable development, but rather to implement a small but detailed segment.

After an initial meeting with the customer, the project was narrowed down to an appropriate scope. The project would primarily focus on the SDG *3. Good health and well-being*, and use data related to air quality measurements provided by the municipality. The data provided would be put in a database capable of storing resource description framework (RDF) data and inferencing implicit relations (see section 2.4 for more details). A web application utilizing the ontology would be made, so that members of the municipality could easily test and evaluate the product themselves. Lastly, a server would be used to connect the front-end to the database. Since the product would be a prototype, it was not necessary to choose a technology stack to optimize performance for many concurrent users.

## 1.4 Project Overview

To achieve the objectives laid out in accordance with the project description and customer meetings, the project was divided into three phases: a research phase, a design phase and a development phase. During the research phase, the sustainability goals, perspectives in Trondheim, taxonomies, the semantic web and ontologies were researched. These concepts are described in detail in chapter 2.

After the research phase, a design phase commenced. As the customer had not provided any requirements for the user interface, a thorough design phase was necessary. It was important to discover the features required in a visualization tool for ontologies. Thus, a design prototype had to be made and tested with members of the municipality. The prototyping and user testing is described in detail in section 3.5.

Following the design phase was the development phase. This was a large phase, divided into several tasks. The central part of the project was to develop an ontology using the Web Ontology Language (OWL), which was a complex task designated primarily to a few members of the group. Section 2.4.4 provides a general description of ontologies and OWL, and section 4.4.4 describes the development. Furthermore, a network graph capable of displaying an ontology had to be made. Section 3.2.2 describes the development of this graph. With the developed ontology and the graph, the two had to be connected with a REST API. The development of the back-end is described in section 3.2.1. Lastly, the rest of the front-end of website had to be developed. The choice of a design framework and development of the website's front-end is described in section 3.2.2. A summary of the product development, as well as a description of the final product is described in section 3.5. A detailed explanation of the project process can be found in chapter 4.

## 1.5 Stakeholders

The primary stakeholders for the product are the customer and the potential future development team. For the customer, this prototype aims to represent the tool necessary to convince colleagues in the municipality that ontologies are the preferable option for storing data. Additionally, the prototype will work as a foundation for further development.

Employees in the municipality working with sustainable development and other connected issues are secondary stakeholders of the product.

# Chapter 2

# Preliminary Research

Due to the complexity associated with developing a prototype for systematizing sustainable development, a comprehensive research phase was required. This chapter is a summary of the work performed during the research phase. The chapter begins with an introduction to the sustainability goals, followed by the locally oriented perspectives in Trondheim. Then, a short description of taxonomies is provided, followed by a detailed examination of a specific taxonomy related to sustainable development. Afterwards, semantic data and the associated RDF language is introduced, which creates the baseline of the Web Ontology language (OWL). The final section explains the capacity an ontology has in terms of systematizing and merging complex concepts efficiently, not possible in the general representation methods used in today's society.

## 2.1 The Sustainability Goals

The sustainability goals are objectives set to achieve a global sustainable society by 2030.[2] These goals were created in 2012 at the United Nations Conference on Sustainable Development in Rio de Janeiro.[1] Even though they were created almost ten years ago, the focus and pressure to create a sustainable society, both in the private and public sector, did not arise before a few years ago.[3] However, currently the word sustainable is used in every aspect, from the lifestyle of one person to the life-cycle chain of the world's largest companies. The goals themselves consist of 17 main goals, illustrated in figure 2.1. Each goal has specific targets, for instance the targets of *2. Zero Hunger* include:

*2.1 By 2030, end hunger and ensure access by all people, in particular the poor and people in vulnerable situations, including infants, to safe, nutritious and sufficient food all year round.*

*2.c Adopt measures to ensure the proper functioning of food commodity markets and their derivatives and facilitate timely access to market information, including on food reserves, in order to help limit extreme food price volatility.*

Targets with numerical identification values (*e.g. 2.1*) has a set time frame for completion, while targets with numerical and character identification values (*e.g. 2.a*) are targets set to be maintained and worked towards in the foreseeable future.

A common misconception for these sustainability goals and their 169 combined targets is that they are independent of each other.[4] In reality, if progression was achieved, most of the targets within a goal would positively contribute towards other goals. For example, achieving a target within *Clean Water and Sanitation* would have a positive contribution towards *Good Health and Well-being*.[5] Yet, as the sustainability goals cover a large amount of aspects with a great detail for the *social*, *economical* and *nature* environments, it is both a complex and demanding task to create approaches where all the goals are achieved by the set time frame. Especially, when some goals and their related targets would generate a trade-off to another goal. For example, progression within *9. Industry, Innovation and Infrastructure* would have a negative contribution towards *12. Responsible Consumption and Production*.[5]

The extensive articles *Towards understanding interactions between Sustainable Development Goals: the role of environment-human linkages* by J. Scharlemann *et al.*[3] and *Mapping the Sustainable Development Goals Relationships* by L.M. Fonseca *et al.*[5] provide a better understanding of how one sustainability goal affects the others. Additionally, both articles conclude that after almost ten years of determining approaches to globally reach all of the sustainability goals, an overall method to accomplish this has not yet been discovered.


Figure 2.1: The 17 sustainability goals created at United Nations Conference on Sustainable Development in Rio de Janeiro.

The complexity of the sustainability goals makes the group's given task to systematize sustainable development quite challenging. However, one notion needs to be mentioned. The goal of this project is not to discover new connections and possible approaches towards achieving sustainable development, but rather to create a representation method based on OWL, using existing connections and approaches.

## 2.2 Perspectives in Trondheim

Similar to the sustainability goals, the perspectives in Trondheim are goals created by Trondheim Municipality, in order to achieve a sustainable society. These perspectives are locally oriented towards the geographical location defined within Trondheim Municipality. The perspectives are not yet finalized, but the group has been given access to the draft for this project. There are 14 locally oriented perspectives in total, which creates a challenge. A representation method such as taxonomies, described in the next section, would struggle to maintain consistency when combining the perspectives in Trondheim and the sustainability goals.

## 2.3 Taxonomies

In earlier history, taxonomies have been used for biological classification, which is the scientific study of classifying biological organisms.[6] However, over the years the word taxonomy has become a synonym for classification. The general concept behind a taxonomy as a representation method, is to bring an order in the apparent chaos.[7] The general structure can be represented as a tree, where every concept defined within the taxonomy is related to the main class, generally called the root. From the root, sub-classes called branches are connected, where concepts defined within these branches are classified with a greater detail than that of the main class. These sub-classes can have associated branches, leading to an even greater detail, creating the so-called tree structure. In modern day, taxonomies are used to classify other aspects than that of biological organisms, such as the taxonomy described below.

### 2.3.1 *Statistisk Sentralbyrå's* (SSB) Taxonomy

The taxonomy created by SSB, is a representation method for municipalities in Norway to consistently define and structure indicators within sustainable development.[8] It was created to compare and reuse approaches, along with having a consistent naming scheme. Figure 2.2 visualizes the structure of the taxonomy and its associated tree structure, where the *Indicator* class is the root class of the taxonomy.

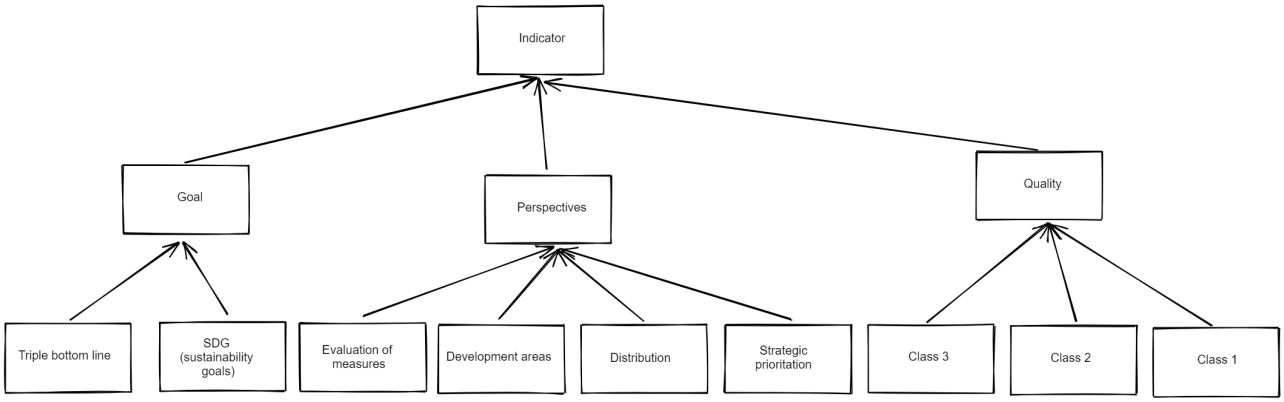


Figure 2.2: Visualization of the three first levels of SSB's taxonomy for classification of indicators for the sustainability goals.[8]

### 2.3.2 Limitations of Taxonomies

The interaction between the sustainability goals and the locally oriented perspectives in Trondheim, generate complex relationships. A representation method such as SSB's taxonomy can classify these relationships. However, not without compromising its class hierarchy or significantly reducing its efficiency.

Considering that all connections between classes within a taxonomy only point upwards, defining relationships between same level classes is not possible. Similarly, it would not be possible to define interactions between classes located in different parts of the class hierarchy. To define an indicator for sustainable development, every single relationship needs to be defined. Furthermore, each relationship may be associated to a strength. In order to explain the complexity that might occur when implementing one single indicator, an example is created: *Approach to improve air quality.*

In this example, SSB's taxonomy is used to define an approach to improve air quality. The details about the actual approach are not important in this example, but rather the corresponding effects of the implemented approach. Based on the representation method, the approach has to be defined to have a large contribution towards sustainability goal *3. Good health* and the *Social environment*, as air quality surrounding a geographical location has a large impact on the health of the inhabitants.[9] However, based on the interactions between the sustainability goals,[3, 5] progression within *3. Good health* has a strong contribution in regards to *1. No Poverty*, *2. Zero hunger* and more sustainability goals. Additionally, *3. Good health*, has a strong trade off with *13. Responsible consumption and production.* Can it be expected that the user defining this approach has the detailed knowledge to manually define these relationships?

Keep in mind that only strong contributions and trade offs are mentioned in this example. Mathematically, this definition of a strong contribution corresponds to the following:

$$Moving\ 1\ step\ forward\ within\ X. \tag{2.1}$$

$$X\ has\ a\ strong\ contribution\ towards\ Y. \tag{2.2}$$

$$Y\ moves\ between\ 0.65\ and\ 0.95\ steps\ forward. \tag{2.3}$$

Defining relationships of lesser strength requires even more detailed knowledge. However, the combination of all factors would reflect why this particular approach was successful or not. This leads to an essential part of sustainable development: reproducibility. How can one municipality reuse an approach from another, if all these relationships are not defined during the classification of the actual approach?

Taxonomies do not offer any sensible way of visualizing these relations. Thus, SSB's taxonomy is not an optimal representation method to systematize sustainable development or to reproduce successful approaches.

## 2.4 Semantic data

### 2.4.1 The Semantic Web

In 2001, Berners-Lee *et al.* published their proposition of an extension of the current web called the Semantic Web.[10] The basic principle of the Semantic Web is that not only web pages, but also pieces of data have unique identifiers, called Universal Resource Identifier (URI).

Table 2.1: Examples of the triplet structure in the RDF language.

| **Subject** | **Predicate** | **Object** |
|---|---|---|
| Human | has common ancestor with | Chimpanzees |
| 1 | is a | Integer |
| A sustainability goal | is a | Goal |

Table 2.2: Examples of the RDFS extension, along with their class definition.

| **Subject** | **Predicate** | **Object** | **Definition** |
|---|---|---|---|
| Has strong correlation to | rdfs:range | Sustainability goals | Properties |
| 3. Good health | rdfs:isDefinedBy | "" (String) | Utility properties |
| Sustainability goals | rdfs:SubClassOf | Goals | Classes |

### 2.4.2 The Resource Description Framework (RDF)

Semantic data is based on Resource Description Framework (RDF), defined by W3C in 1999.[11] RDF data is encoded in sets of triples, containing a *subject, predicate, object* - statement. Translated into natural language, the triplet structure can be defined as follows: The concept in question (*subject*), has a connection (*predicate*), with the following concept (*object*). Table 2.1 illustrates some specific examples of the triplet structure, where each is identified by a URI.[10]

In their book *Semantic Web for the Working Ontologist: Effective Modelling in RDFS and OWL*,[12] Allemang and Hendler refer to regular data as "dumb" data. By dumb they mean inconsistent, out of sync and disconnected data. When the data changes, someone must manually update associated references to correctly represent the data. With semantic data, however, the interconnectivity automatically updates the references. This is referred to by Allemang and Hendler as "smart data".

The possibilities arising from the relatively simple triplet structure drastically surpass representation methods such as taxonomies. Concepts within a taxonomy only point upwards towards the main class. The triple structure removes this limitation since relationships between concepts can be defined independently of their related placement.

### 2.4.3 Resource Description Framework Schema (RDFS)

RDFS is an extension of the RDF language, providing sets of classes with associated properties.[13] Table 2.2, shows some examples of standardized predicates defined within the extension, along with their definitions. The extension provides compatibility and consistency to each unique collection of data written in RDF. Properties asserted based on RDFS would contain a consistent definition across the the Semantic Web. For example, taxonomies can be converted to the RDF language using only the *rdfs:SubClassOf* predicate.

### 2.4.4 The Web Ontology Language (OWL)

Web Ontology Language (OWL) is a family of knowledge representation languages, including RDF and RDFS, providing functionalities to express complex relationships.[14] Every ontology represented by OWL also contains its associated functionalities. Each ontology is defined with a set of specific URIs, converted into name tags. For example, the RDFS extension has the name tag *rfds*. To illustrate the extensive definitions occurring in an ontology, table 2.3 shows the subject *Good health* and its connections. The *SDG:* name tag represents the ontology specified for this project. Other name tags are knowledge representation languages.

Keep in mind that *SDG:Good_health*, based on the class hierarchy, also has several *rdf:type* connections and several connections with the same object property. As a matter of conciseness, table 2.3 only shows unique predicates.

Table 2.3: Subject *Good health* and connections accompanying it.

| **Subject** | **Predicate** | **Object** | **Definition** |
|---|---|---|---|
| SDG: Good_health | rdfs:label | Good health$^{String}$ | Annotation |
| SDG: Good_health | SDG:HasHighContributionTo | SDG:Zero_Hunger | Object property |
| SDG: Good_health | SDG:HasTargets | 13$^{Integer}$ | Data property |
| SDG: Good_health | rdf:type | SDG:Sustainability_go... | Classes |
| SDG: Good_health | SDG:HasHighTradeOffTo | SDG:Responsiable_pro... | Object property |
| SDG: Good_health | rdfs:IsDefinedBy | https://sdgs.un.org/goal... | Annotation |
| SDG: Good_health | schema:icon | https://www.un.org/sus... | Schema |

### Critical Mass

One reason why ontologies are not common in today's society, is the extensive definitions necessary to define concepts.[15] Whereas a taxonomy is straight forward to develop, an ontology often requires a lot more manual work initially. The critical mass of a representation method is defined as the amount of structural expansion associated to its development before it provides useful information.[16] When systematizing sustainable development, an ontology has a substantially larger critical mass than representation methods such as taxonomies. This may be an important reason why no large-scale databases use ontologies.

### Explicit and Implicit Relationships

The main reason why representation methods are used, is to generate implicit relationships. This reduces the amount of intuitive connections that need to be defined when incorporating a new concept. For example, in SSB's taxonomy in figure 2.2, placing a concept within *Development areas* would implicitly connect the concept to *Perspectives* and *Indicator*. This also applies to ontologies. However, with ontologies, implicit relations may be drawn across the data set, not just upwards in the class hierarchy. This drastically increases the amount of implicit connections possible with one single explicit connection. An ontology can typically be portrayed as a spider web, where the size of the web depends on the population and structural expansion of the ontology.

In the initial development of an ontology, every new concept has to be defined based on the triple structure a significant amount of times. As the population and structure expand, one new connection starts to lead to implicit relationships between other concepts. At a certain stage, the critical mass is reached and one explicit connection leads to a significant amount of implicit relationships. After this point, the expansion of the ontology would generate a larger and larger spider web, without the need of extensive manual work.

### Object Properties

Object properties are built-in properties within *OWL*, which help reduce the manual work developing an ontology.[17] For example, an object property such as *hasContributionTo* can be defined as a symmetric object property. Thus, by explicitly defining the connection:

$$X \quad hasContributionTo \quad Y, \tag{2.4}$$

another connection would implicitly be defined:

$$Y \quad hasContributionTo \quad X. \tag{2.5}$$

Similarly, if *hasContributionTo* is defined an asymmetric object property, explicitly defining the connection:

$$X \quad hasContributionTo \quad Y, \tag{2.6}$$

would result in:

$$Y \quad !hasContributionTo \quad X, \tag{2.7}$$

where !*hasContributionTo* corresponds to *does not contribute to* in natural language.

In *OWL* there are more functional object properties which use the same principle to reduce the amount of explicit definitions, along with defining rules and restrictions to maintain a consistent ontology.

```
PREFIX SDG: <http://www.semanticweb.org/aga/ontologies/2017/9/SDG#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
    SELECT *
    WHERE {
        {
            SDG:B8 SDG:harModeratKorrelasjon ?Object .
            OPTIONAL {?Object rdfs:label ?ObjectLabel}
        }
        UNION {
            ?Subject SDG:harModeratKorrelasjon SDG:B8 .
            OPTIONAL {?Subject rdfs:label ?SubjectLabel}
        }
    }
```

Figure 2.3: An example SPARQL query, visualized by *GraphDB*.

Table 2.4: The data retrieved from the query in figure 2.3.

| Object | ObjectLabel | Subject | SubjectLabel |
|---|---|---|---|
| SDG:B1 | ”Utrydde fattigdom” | | |
| SDG:B11 | ”Bærekraftige byer og lokalsamfunn” | | |
| | | SDG:B5 | ”Likestilling mellom kjønn” |
| | | SDG:B6 | ”Rent vann og gode sanitærforhold” |

#### Incorporation of Scientific Resources

In order for the ontology to provide trustworthy implicit relations, it's important that every class, object property and instance are documented. The use of *Annotations* provides detailed information about a specific concept. For example, *rdfs:IsDefinedBy* makes it possible to define where the definition comes from.

#### SPARQL

An ontology is stored in a database as a set of triplets. To be able to retrieve specific data from a database, *SPARQL* was defined.[18] Syntactically, *SPARQL* is reminiscent of the more familiar *SQL*. Figure 2.3 shows an example of a *SPARQL* query. As the figure shows, there is a *SELECT* clause which, similar to *SQL*, specifies which values to retrieve from the query. The *WHERE* clause specifies the filter for which data should be retrieved, also similar to *SQL*. The *UNION* operator combines two sets of data provided by *WHERE* clauses, similar to the *FULL OUTER JOIN* operator in *SQL*.

Unlike *SQL*, however, variables in *SPARQL* are uniquely identified with a URI. Because the data is unique, it is not necessary to group data, and a *FROM* clause specifying which table to search is not necessary. Instead, the entire database is searched. Figure 2.3 retrieves all triplets where *SDG:B8* is either the subject or object, and where the predicate is *SDG:harModeratKorrelasjon*. Here, *SDG* is an abbreviation of ⟨http://www.semanticweb.org/aga/ontologies/2017/9/SDG#⟩. To avoid specifying the full URI every time, an abbreviation can be specified at the beginning of the query with the *PREFIX* clause, as shown in the figure. The two *OPTIONAL* lines add the label of the subject and object to the query result. The *OPTIONAL* clause specifies that the label is not a requirement, *i.e.* a subject or object which doesn't have a label is still retrieved. Table 2.4 shows the data retrieved from the query in figure 2.3.

## 2.5 The Power of Ontologies

The functionalities introduced in the previous section enable the combination of the concepts *The sustainability goals*, *Perspectives in Trondheim* and *SSB's* taxonomy into one representation method. Furthermore, scientific

resources associated with the sustainability goals can be defined and incorporated with the use of predicates. The class hierarchy of the taxonomy can be directly implemented with the RDFS extension.

It is important to note, that creating a well-defined ontology of sustainable development would require collaboration between experts from almost every sustainability goal and perspective in Trondheim. The ontology developed for this project is made by informatics students. Although the data is based on expert sources, the ontology would not be sufficiently well-defined for usage beyond a prototype.

To visualize the capacity within an ontology systematizing sustainable development, the same example *Approach to improve air quality* from 2.3.2 will be used. In the ontology, unlike *SSB's* taxonomy (see figure 2.2), the relationships between concepts have already been defined. This means that the user implementing the approach into the ontology does not need to research all the associated side effects. A single explicit connection towards the concept which is directly linked to the approach would be enough.

### 2.5.1 Are there Alternative Solutions?

Native graph databases have the possibilities of creating detailed data points as nodes and connect these data points by relationships. *Neo4j* is the self-proclaimed leader of graphical databases and can furthermore be described as a major actor within representing graphical databases.[19] However, *Neo4j*, as other graph databases, does not use the RDF language.

#### Key Differences

In a short summary, *Neo4j* is a graph database which captures data, whereas an ontology captures the structure.[20] In cases where the main objective is to deal with actual data and structure the data accordingly, a knowledge graph would be superior to an ontology. However, if the main objective is rather to connect and structure a representation of metadata, an ontology is preferred.

A general saying is that an ontology is always able to integrate a graph database, while a graph database can't always integrate an ontology. As introduced before, the language associated to ontologies is a well defined logical language. It is important to note that a graph database such as *Neo4j*, is able to recreate similar entities and its explicit relationships.[20] However, *Neo4j* is not able to compete with the inferencing associated with the set of restrictions and rules within an ontology. This is the crucial difference between ontologies and graph database.

Specifically, in this development, *Neo4j* would not be able to capture the essence of the task at hand. While it would be able to produce a similar graphical representation, the lack of capacities within inferencing would easily make the graph database inconsistent. Additionally, the associated population expansion would not reduce the amount of manual work compared to ontologies.

### 2.5.2 Essential Sources

Three essential sources need to be highlighted in accordance with the development of the ontology.

*SSB's* taxonomy for classification of indicator for the sustainability goals has been used as the core for the developed ontology.[8] Without this resource the structural expansion of the ontology would be severely reduced. Additionally, the customer had access to an early draft of this taxonomy, which was one of the reasons for this project.

Mapping the Sustainable Development Goals relationships has been used to define object properties such as *HasHighContributionTo* and *HasHighTradeOffTo*, based on their work in discovering interactions between sustainability goals.[5]

The class hierarchy of Metoder [Nor] in the currently developed ontology is based on *An Ontology-Based Knowledge Modelling for a Sustainability Assessment Domain.*[21] This ontology has provided a better insight during the structural expansion.

# Chapter 3

# Product

In this chapter, the development of the product will be explained. First, the requirements provided by the customer, both functional and non-functional, will be described. Then an overview of the application's technology stack and architecture is explained, followed by an overview of testing and continuous integration. Afterwards, the development process will be presented, from initial design prototypes and user testing to the final product. As this product is meant to be a prototype, the chapter is finished with a discussion of possible further work.

## 3.1 Requirements

In accordance with the preliminary research, developing an ontology based on OWL was a prioritized requirement. The customer also had recommendations regarding accompanying technologies, but the decisions regarding design, coding practices and accompanying technologies were left for the group to make. During the first few customer meetings, a set of functional and non-functional requirements for the product were established. The following sections, provide a description of these requirements, along with a summary in table 3.1.

### 3.1.1 Functional Requirements

In order to make the prototype easily accessible by other employees of the municipality, the customer recommended a website interface. Furthermore, the user interface would be in Norwegian, as the given data sets are made in Norwegian and Norwegian is the native language of the the employees of the municipality.

The purpose of the prototype was to provide information about sustainable development and discover connections between concepts based on a graphical representation associated to the implicit connections within an ontology. The extracted data needed to be visualized in a user-friendly manner, and the navigation of the graphical representation needed to be clear. Additionally, the ontology would provide data that is relevant for Trondheim Municipality, resulting in easier and quicker understanding of the data provided.

As the website was meant as a showcase for ontologies, it was not a requirement to let the user edit the ontology. Changes to the ontology must be made by project administrators.

As explained above, the decisions regarding design were left for the group to make, and no color schemes or other design details were specified by the customer.

### 3.1.2 Non-Functional Requirements

In accordance with the only requirement from the customer and the purpose of the project, the ontology must be be written using OWL.

During the first customer meetings, the customer was asked about requirements regarding authentication and security. At a later stage, the data received from the costumer might contain sensitive information. Thus, authentication with user tokens and a log-in page may be implemented in the future. Nevertheless, for this prototype, authentication is not necessary. It is important, however, that the ontology is stored in a secure database which does not let users of the website change or remove the ontology.

The data should be backed up whenever changes are made to ensure changes can be reverted. Consequently, maintainability should be implemented in the case of data loss or any other unfortunate event. Another requirement from the customer is that the code should be open source, so that other developers may expand

Table 3.1: Functional and non-functional requirements.

| Functional requirements | Non-functional requirements |
|---|---|
| The language for the end-user must be Norwegian | The ontology must be written using OWL |
| The graph must be easy to use | The code should be open-source |
| The user must be able to discover explicit and implicit connections regarding sustainable development | Data must be backed up to ensure data control, leading to better maintainability. |
| The ontology needs to provide relevant data for Trondheim Municipality | The website must be secure and prevent users from changing, adding or removing data without authorization. |
| | The website must run at an acceptable frame-rate |

the prototype or develop a new product based on it. Thus, *Git*[1] should be used for version control, to ensure that others can both contribute and roll back unwanted changes to the code.

As the prototype is meant to be used only by a few users, scalability is not a concern. However, to be able to demonstrate the concept convincingly, the client performance, *i.e.* the performance of the website, is an important requirement. The prototype should be able to run at an acceptable frame rate without performance issues that diverts attention away from the content.

## 3.2 Technology Stack

The compatibility of the technology stack is an essential part of this project. The ontology has been developed in accordance to the preliminary research (see section 2.4). Since there is no point in reinventing the wheel, several third-party components and technologies have been used throughout the project.

Although using a third-party library improves the development speed, there are possible issues with compatibility, security, lacking maintainability and infrequent updates. Before including each library in the project, the group discussed if the library was needed, and checked the license to ensure that it was open source and met the licensing criteria from the customer.

### 3.2.1 Back-end

According to the customer's requirements (see section 3.1), scalability is not a concern of the prototype. Thus, performance in the back-end was not of major importance when selecting a technology stack. The main priority was rather to use technologies and libraries that are easily accessible and that the group has experience with, in order to optimize the development speed.

#### Triplestore

The developed ontology naturally had to be stored in a database. As ontologies consist of triples (see section 2.4), a triplestore had to be used. A triplestore is a purpose-built database for the storage and retrieval of triples.[22]

Two popular *triplestores* were considered, namely *OntoText's GraphDB*[2] and *Cambridge Semantics' AnzoGraph*,[3] based on recommendation from the customer. Both of the databases required a registered account to use their services. However, as *Cambridge Semantics* also required an application to be sent and reviewed to gain access to *AnzoGraph*, *GraphDB* was used initially. The group sent an application to *Cambridge Semantics*, but after several weeks without response, *GraphDB* was chosen due to the limited time aspect of the project.

After a reminder was sent to *Cambridge Semantics*, they eventually responded to the group's inquiries. To gain access to the software the group would have to join a video consultation meeting with them and explain the need for the software. At this point the group had worked with *GraphDB* for weeks and was content with its performance, and decided against further inquiries as the development was almost at an end. For a future project with more simultaneous users and a higher required server capacity, *AnzoGraph* may be considered. A more detailed comparison between *AnzoGraph* and *GraphDB* is provided in section 3.6.2.

[1] https://git-scm.com/
[2] https://www.ontotext.com/products/graphdb/
[3] https://www.cambridgesemantics.com/anzograph/

#### Enapso GraphDB Client

Although the group has realized the potential of triplestore databases, their popularity is still far lower than traditional *SQL* and *NoSQL* databases. This was clear when a library had to be chosen to connect the server to *GraphDB* (see section 3.2.1).

There are some libraries to compose and send *SPARQL* queries, such as the *npm* package *SPARQL HTTP Client*[4]. However, this library receives data as a primitive line-by-line byte-stream. To use this library, the data would have to be parsed and transformed into *JSON* manually. Error handling would also have to be implemented. To avoid unnecessary work, the group decided to use a wrapper library made specifically for *GraphDB*, called *Enapso GraphDB Client*[5].

A potential issue with *Enapso GraphDB Client* is its low usage, *i.e.* that it has few weekly downloads on its *npm* page.[23] Since anyone can submit *npm* packages quite easily,[24] there is no guarantee that libraries do not contain malicious code, and it is generally discouraged to use uncommon libraries. However, *Enapso GraphDB Client* is an open-source project, and although it is developed by a small group, it is fairly frequently updated and well-maintained. After some considerations, the group decided to use this library, which simplified database access to a single line of code.

#### Express.js

*GraphDB* is written in *Java*.[25] Hence, the group assumed that a *Java* REST server would be the most compatible, based on the *GraphDB* documentation.[26] A *Java* server would also let the group use *Kotlin* if preferred, as they are interchangeable and compile to the same *JVM* bytecode.[27]

Initially, the plan was to develop a REST server using *Kotlin* and *Ktor* with a custom *GraphDB* instance. However, during the initial development stages, the group realized that there was no need for modifying a *GraphDB* instance, but rather use it exclusively as a database that could be queried.

Several third-party components for *GraphDB* written in *JavaScript* with *Node.js*[6] were discovered. These provided better methods for sending *SPARQL* queries to *GraphDB*. Additionally, using a *JavaScript*-based back-end and front-end would make the entire application more uniform. Thus, the group ultimately decided to develop the server with *Express.js*[7] and *Node.js*.

#### Communication between *Express* and *GraphDB*

As explained in section 2.4.4, the queries from back-end to *GraphDB* need to accommodate the RDF language. The group decided to use *SPARQL*, as this syntax is recommended by World Wide Web Consortium (W3C) and is the standard query language for *GraphDB*.[28, 29] Other RDF query languages were looked into, such as *XQUERY*[8], but as *SPARQL* seemed the most intuitive and met all necessary requirements for the technology stack, it was decided to continue using it.

### 3.2.2 Front-end

#### React

According to the customer, Trondheim Municipality has previous experience with the *JavaScript* framework *React*. Since this product may be further developed by Trondheim Municipality, the customer preferred this as the front-end framework. Most members of the group also have experience with *React*.

*React* allows creation of modular components which would be re-usable and scalable throughout the project. As *React* is the second most popular *JavaScript* framework on *GitHub* with over 160,000 stars,[30] there is a lot of community driven documentation. Most of the questions that arose during development had already been asked and answered by the community, which was valuable to the group.

#### Redux

As the front-end expanded, the group realized that a global state was necessary. In essence, a global state is a way to allow several components to update, and be updated by, a variable without having to pass the variable up and down the component tree. Most of the group members were familiar with *Redux*[9], a commonly used state

[4] https://www.npmjs.com/package/sparql-http-client
[5] https://www.npmjs.com/package/@innotrade/enapso-graphdb-client
[6] https://nodejs.org/en/
[7] https://expressjs.com/
[8] https://www.w3.org/XML/Query/
[9] https://redux.js.org/

management library for *React*. However, based on experience, *Redux* is time-consuming due to its verboseness. In another project, some group members were introduced to *Recoil*[10], which is a more recent and less verbose state management library. Initially, the group decided to try *Recoil* as the state management library for the application.

A problem with *Recoil* was discovered when implement error handling, as it does not allow modifying the state from outside the *DOM*. Thus, whenever a network request failed, the error message would have to be passed down to the relevant component and dispatched from there.

To prevent unnecessary passing of variables through the *DOM*, the group decided to replace *Recoil* with *Redux*. With *Redux*, whenever a network request failed an error could be dispatched directly from the network module. The change from *Recoil* to *Redux* saved a lot of time during development as the API error handling only had to be defined once in the network module. While *Redux* requires more code, this was not a major issue because only some of the data was put in the global state. The global state diagram can be seen in figure 3.2 and shows which components access the global state.

### Chakra UI

To make a uniform design across the entire website, and to reduce the time spent on the design process, the group quickly decided to use a third-party component library. The group decided to use *Chakra UI*,[11] as it is a simple, modular and accessible component library, which is easy to set up and use out of the box. *Chakra UI* components are also highly customizable and composable, and they are easily made responsive.

Other component libraries, such as *Material UI*[12] and *Semantic UI*[13], were considered as well, but were ultimately not chosen for the following reasons. *Chakra UI* is made specifically for React, resulting in less compatibility issues.[31] It is also easy to learn with experience in *React*, considering the prop based model of styling components.

### D3.js

An easy and intuitive graph visualization of the ontology is one of the main requirements of the project, as described in section 3.1. As none of the group members had made a similar graph visualization before, careful considerations were required. The OWL-based structure requires a non-linear network graph which is able to display nodes and edges in different directions. A library was needed to create a customizable force-directed graph.

Several libraries for drawing graphs were considered. One of the libraries was *Visx*[14], a collection of expressive, low-level visualization primitives for *React*, which is commonly used.[32] The problem with *Visx* for this particular project is that the graphs are not sufficiently customizable. Another library that the group considered is *G6*,[15] which is more customizable than *Visx*. However, when evaluating *G6*, it was discovered that the library does not have much community-driven documentation and tutorials relevant to the development of the ontology graph. Most of the community-driven content that does exist is not made in English.

Based on recommendations from fellow students, the group decided to try *D3.js*. It has a low-level approach, focusing on composable primitives such as shapes and scales rather than configurable models.[33] The library gives more control and customizability than the other libraries considered, and is popular and well-documented with more than *150 million* downloads.[34]

*D3.js* makes it possible to create a graph with dynamically positioned nodes and edges, and has support for simulating forces to make the graph interactable and dynamic. The strength of the forces and how they interfere with the graph can easily be configured. The simulation divides each calculation into a specific number of ticks. Every tick calls a callback function and the result determines what is rendered in the *DOM*. As *D3.js* is data-driven, the *DOM* elements do not need to re-render if the data is unchanged between ticks. *D3.js* also offers the possibility of manually choosing how to update, delete and add new nodes and edges to the DOM. This level of control is important when working with large graphs to maintain high performance. This process is visualized in figure 3.1.

There are still some downsides to *D3.js*. The library is fairly extensive and requires a vast amount of time to learn compared to more lightweight libraries. However, due to the complexity of the task, the group felt that

[10] https://recoiljs.org/
[11] https://chakra-ui.com/
[12] https://material-ui.com/
[13] https://semantic-ui.com/
[14] https://airbnb.io/visx/
[15] https://g6.antv.vision/en

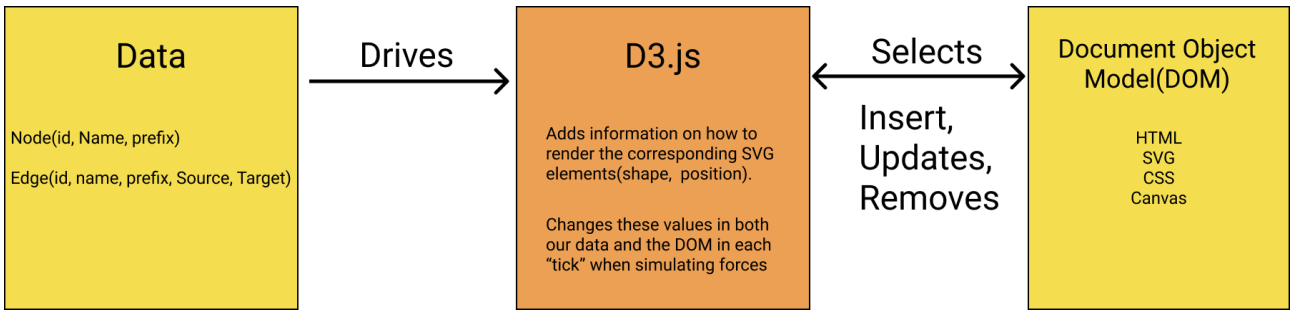


Figure 3.1: *D3.js*' data driven process, manipulating objects and applying them to the DOM.

the customizability provided by *D3.js* was worth the additional time spent learning a such an extensive library.

### 3.2.3 Programming Language

As the product may be further developed by another group of developers, it was important to use a language which makes the code easy to understand. For a website with a *JavaScript* framework, the decisions are fairly limited. However, *TypeScript* is a statically typed language built on *JavaScript*, which provides easier debugging due to fewer bugs associated with incorrect type errors. As a statically typed language, *TypeScript*'s data types are never ambiguous. Furthermore, as *TypeScript* and *JavaScript* are interchangeable, new project code may be written with *JavaScript* instead, if this is preferred by the future development team.

While there are benefits to using a strictly typed language, it also has some disadvantages, particularly when using third-party libraries. As explained in the previous section, *D3.js* was used to draw the graph. While this was a helpful tool, the strict types sometimes made the library more difficult to use. The type names were sometimes ambiguous, and there were several types with multiple generic parameters. As a result, the group sometimes had to resort to the *any* type, defeating the purpose of *TypeScript* over *JavaScript*.

Another problem with *TypeScript* and third-party libraries occurred with *Enapso GraphDB Client*. This library is not very commonly used and did not have defined data types. To avoid using *any* types whenever this library was used, the group wrote custom type definitions for the library. Writing type definitions for a third-party library meant the group had to get acquainted with the source code of the library, which reduces the benefits of using a third-party library.

The usage of *TypeScript* over *JavaScript* added more work to the development process. However, the group decided that the positive aspects of a statically typed language were worth the additional hours.

## 3.3 Architecture

Based on previous experience, the group initially created a physical view (see figure 3.5) to represent the architecture of the prototype. Although this is a sufficient way to represent a simple prototype, the group quickly realized that it does not give a sufficient overview of all the aspects within an application. To accommodate these limitations, the group decided to use the 4 + 1 architecture model, based on material provided for the course and the customer's recommendation.

### 3.3.1 4 + 1 Architecture Model

The 4 + 1 architecture model is a visual documentation for all aspects of the system during development, and is used to locate flaws or missing functionality.[35] The model breaks down the overall architecture into smaller views, providing the opportunity to inspect each view in greater detail. Four views comprise the model: logical, process, development and physical view. Additionally, the model contains a use case view, which represents an outsider's perspective on the system functionality. All of the views are concurrent and have their own perspective on the system's architecture.

The logical view, represented as a state diagram in figure 3.2, visualizes the components comprising the front-end, and how they interact with each other. The project uses a functional design rather an an object oriented design. To better represent the application's logic, a state diagram was created instead of a traditional object model diagram.[35] The diagram displays the global state and the components which utilize it. Furthermore, the diagram provides an overview of which components will re-render whenever any of the other components update the global state.

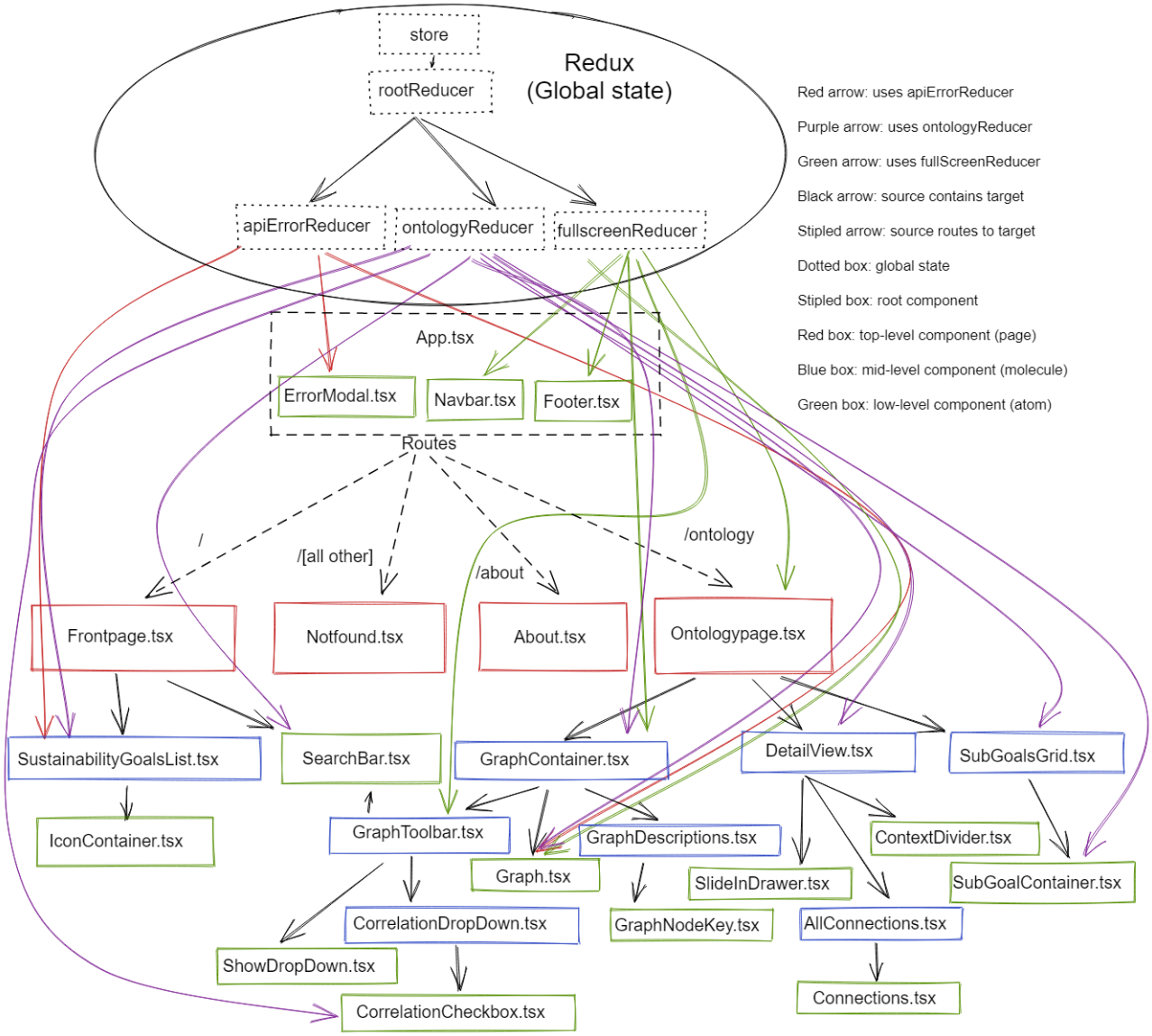


Figure 3.2: Global state diagram visualizing which components in the front-end make use of the global state.

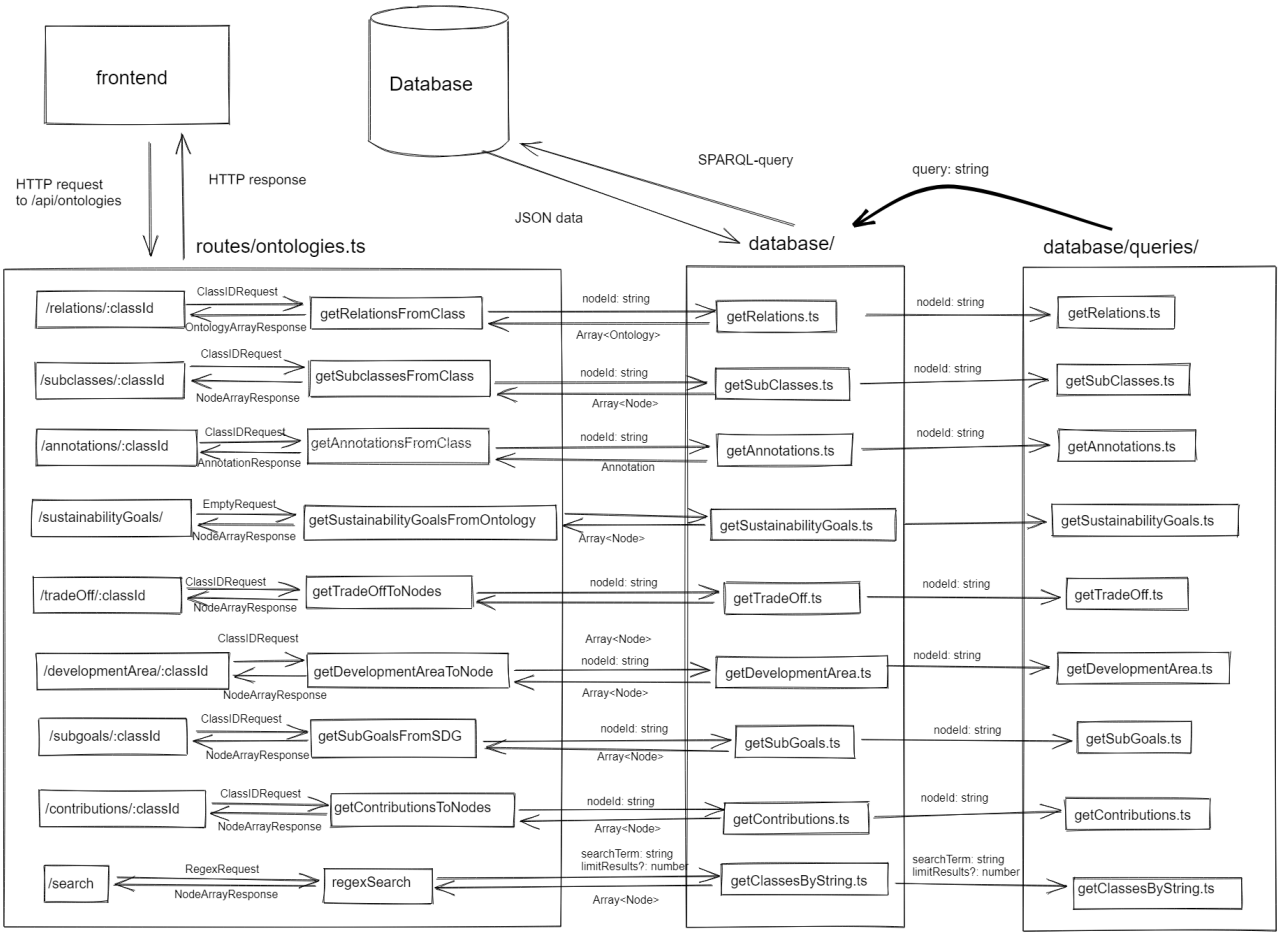


Figure 3.3: Communication diagram displaying how HTTP requests are propagated throughout the back-end.

The process view describes the system's processes and the communication between them.[35] In figure 3.3, this view is visualized as a communication diagram, which describes in detail how queries sent from the front-end are propagated through the back-end and return data from the database.

The development view depicts the system from a programmer's perspective. It describes the system's modulus and/or components, including packages, sub-systems and class libraries. Furthermore, the development view gives a building-block view of the system and provides an easy-to-read overview of the layers within.[35] The diagram describing this view is a component diagram, which shows the components and their corresponding props, illustrated in figure 3.4.

The physical view illustrates the system from a system engineer's perspective. It models the system's execution environment and maps the software artifacts onto the hardware that hosts them.[35] A Unified Modeling Language (UML) deployment diagram is used to model the physical view of a system, as displayed in figure 3.5.

The use case view shows the system's functionality and offers the view of the system from a user's perspective. This view captures user goals and scenarios, which is helpful when defining and explaining the structures and functionality in the other four views.[35] This is explained in figure 3.6, where a UML use case diagram is presented.

### 3.3.2 Deployment

As the project consists of three individual modules (database, server and front-end), the group realized that deploying individual modules would remove the need to start all of them during development. Initially, only the database was deployed to a server which one of the group members manages. As the changes to the database were the least common, this was a natural starting point. Later during development, changes to the back-end were less frequent. At this stage, the group decided to deploy the *Express* application to the same server as the database.

Having the entire back-end of the project deployed, allowed the group to only have the front-end running during development and connecting to it through the exposed endpoints. Changes to the back-end would have to be

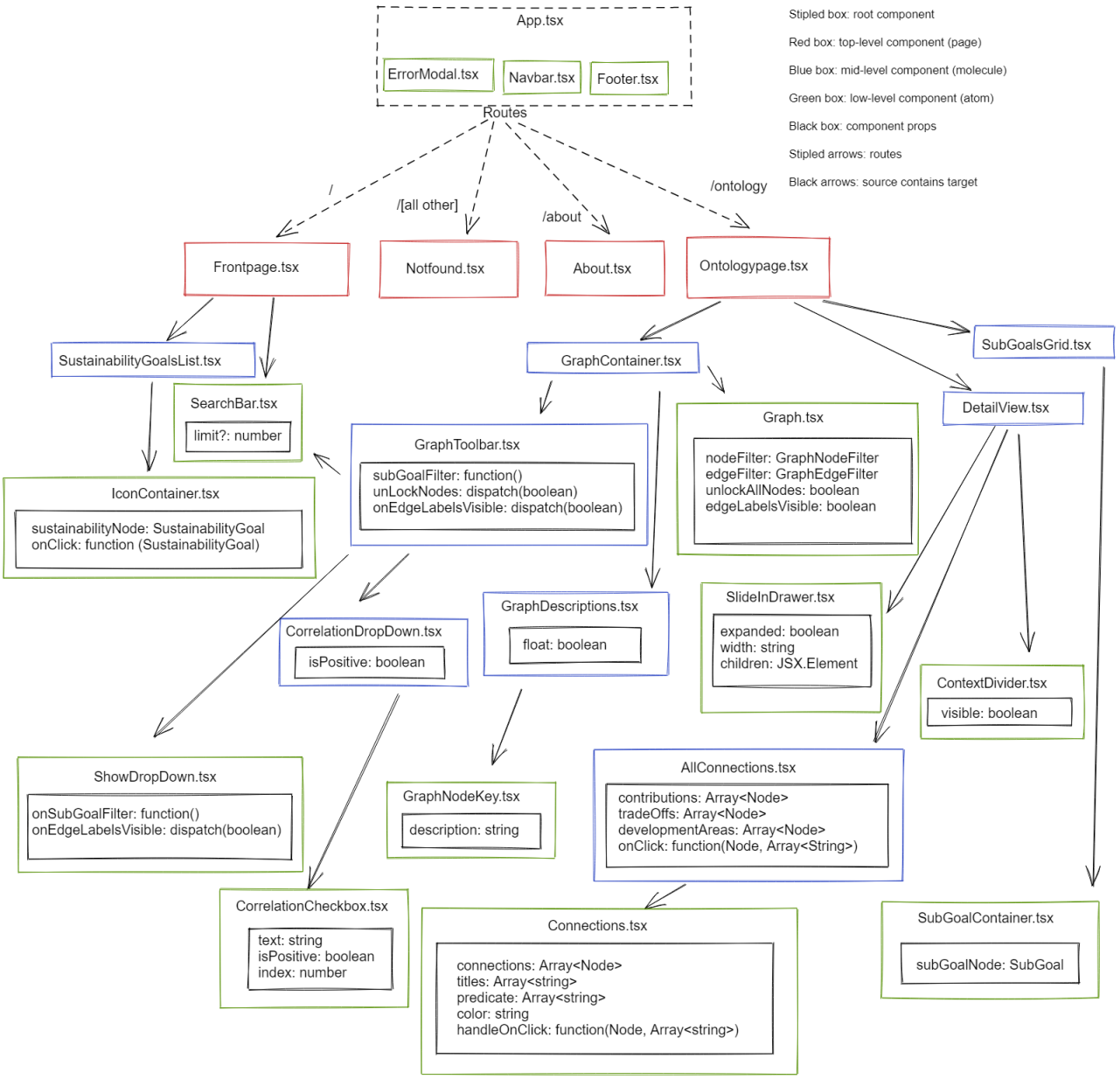


Figure 3.4: Component diagram showing the different input parameters for each component and how the components are related to each other.

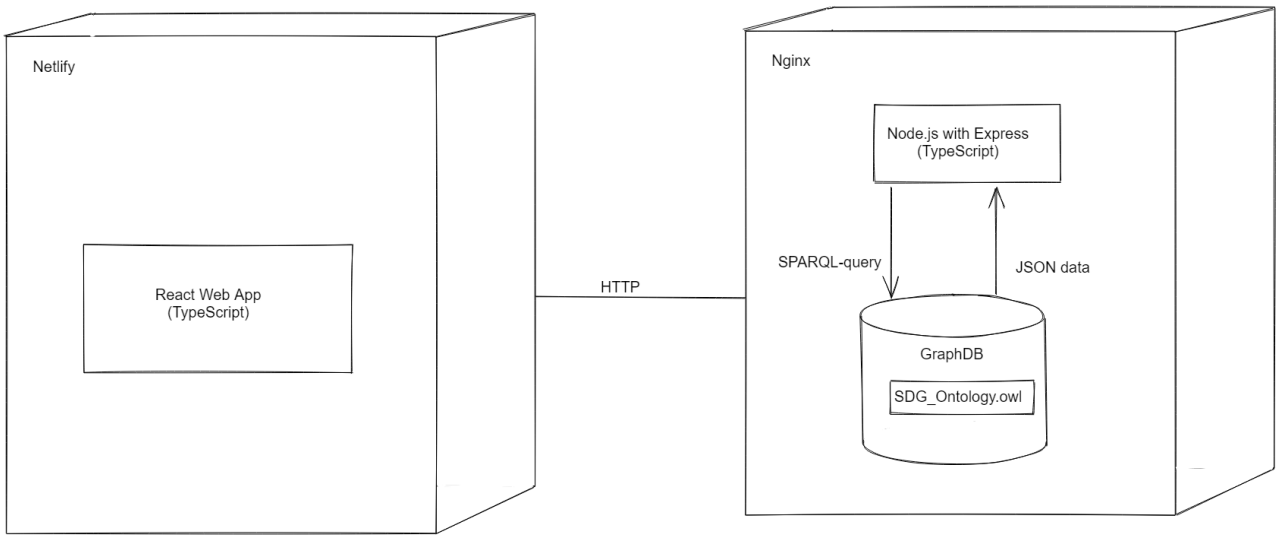


Figure 3.5: Deployment diagram showing the physical relationship between the front-end and back-end.

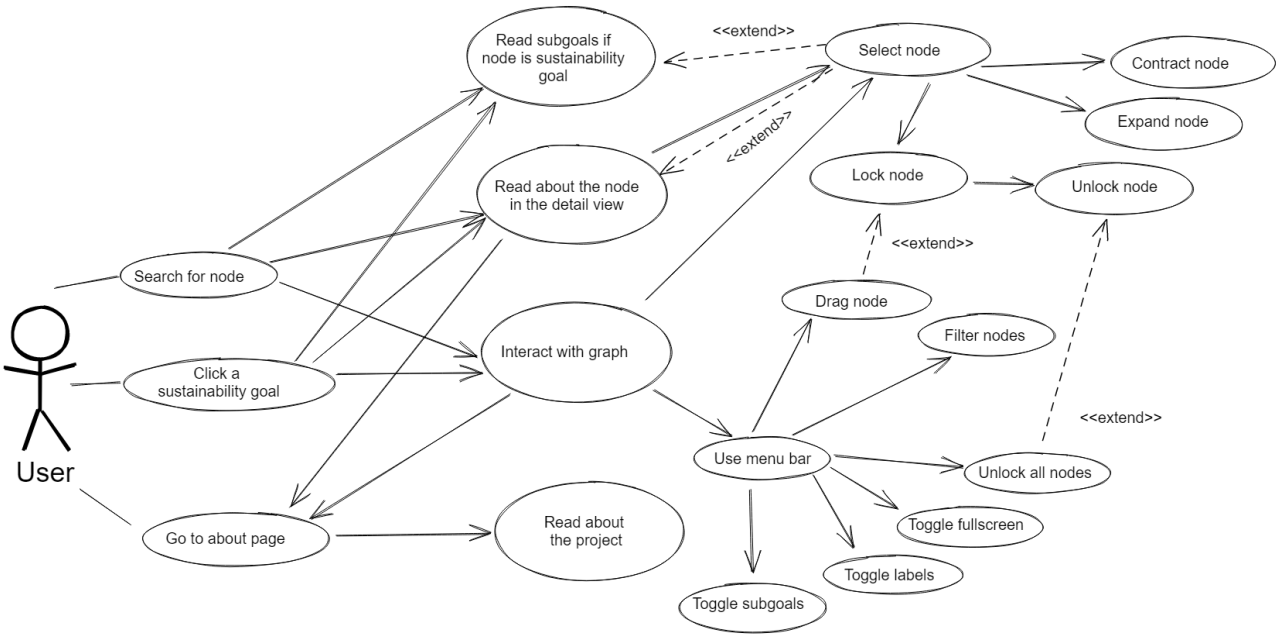


Figure 3.6: Use case diagram displaying the different use cases of the application, as well as what functionality is dependent on each other.

deployed manually on the server as there were no *Continuous Deployment* pipelines configured. In the future, having an automated pipeline for back-end deployment would be a logical step to reduce the amount of manual work needed for deploying new changes.

Before the second round of user testing (described in section 3.5.3), the group decided to deploy the front-end of the architecture as well. With a deployed version of the application, the test subjects did not have to download and run the project themselves, but rather just visit a website. *Netlify*[16] was used to host the website, as members of the group have previous experience with it. The service allows the most recent version of the front-end to automatically be deployed directly from *GitHub* for free.

At the end of the project, the code will be handed over and the product deployed to the municipality's servers. For this deployment, it would be logical to have the entire project deployed to a central source, which would lower the latency for the end-user. Additionally, it would reduce different services in use, which would simplify logging, monitoring, access and maintainability.

## 3.4 Testing

Testing is a key component to creating a high-quality product. When developing tests for a project there are two main approaches: test-driven development and agile development.[36] In test-driven development, tests are written first, and then the modules are coded until they pass the tests. Because a lot of the the decisions regarding the development of the application were left for the group to decide (see section 3.1), a more agile approach to test development was preferred.

As the prototype may be further developed by another group of developers, comprehensive testing will ensure future development will not break the existing code base. With that in mind, the group has implemented unit testing, snapshot testing and integration testing.

### 3.4.1 Continuous Integration

The project has been developed with *Continuous Integration* (*CI*), configured in the *GitHub* repository. Whenever a pull request is made, the *CI* verifies that the code runs and is formatted correctly before the branch is allowed to be merged. Throughout the entire project, this has ensured that the code in the main branch has maintained high quality.

The configuration file for the *CI* is located in *.github/workflows/main.yml* and consists of three stages: *lint*, *test* and *cypress-run*. The *lint* stage checks for inconsistencies and bad practices in the code with *ESLint*[17] and the coding standards defined in section 4.4.1. The *test* stage runs all the unit and snapshot tests, to ensure that the core functionality and components still work as intended. The *cypress-run* stage runs the *Cypress*[18] tests (see section 3.4.4) to ensure that the overall technology stack works as intended. When automatic deployment of the front-end was configured, a fourth *build* stage was added. This stage checked that the the front-end would build without problems to make sure deployment would not fail. However, this drastically increased the time required to run a *CI* job. Because the tests cover the most critical aspects of the application, the build phase would never fail if the tests and linting passed. Thus, the build stage was deemed redundant and later removed.

Even though the *CI* has ensured robust code, there have also been some downsides. One repeated inconvenience is the time required to verify even minor changes. This was especially noticeable after incorporating *Cypress* tests. One way to mitigate the time spent waiting for the *CI* would be to cache artifacts and test results. Node modules generated from the *package.json* file could be cached and only recreated if dependencies were added or removed. Unit and snapshot test results could also be cached to prevent them from running whenever the file that is being tested has no changes from the previous test run. Caching the node modules would save the most time as *Cypress* has to re-create these each time before it can start the tests itself.

### 3.4.2 Unit Testing

When selecting a unit testing framework, *Jest*[19] was a natural choice, as it is pre-installed with *React*.

During the development of the prototype, the group prioritized creating core functionality in the app over spending time writing tests. In the final month of the project, unit tests were added. A lot of the core functionality in the front-end was written in utility files, so the group focused on testing those parts with unit

[16] https://www.netlify.com/
[17] https://eslint.org/
[18] https://www.cypress.io/
[19] https://jestjs.io/

tests. These tests can be found in *frontend/src/tests/common.test.ts*[20]. Initially, the plan was to write unit tests for the back-end as well, but with the integration testing, the group felt that the back-end had sufficient test coverage. Thus, the group prioritized the final development of the application over increasing the test coverage further.

An overview of the test coverage can be seen in Appendix A, figure 5.1.

### 3.4.3 Snapshot Testing

Snapshot is a testing mechanism which is composed of two components. The first component instruments the program to be tested with Snapshot statements, which are used to produce a special trace during the program execution. The second component implements post-processing on the trace.[37] In simpler terms, a snapshot of each component is produced, and new code is tested to verify that the snapshot remains unchanged. The purpose of a snapshot test is to ensure components are not accidentally affected by a change somewhere else in the application.

In total, the project has 25 snapshot tests which cover the vast majority of the components in the project. Retrospectively, the group considers snapshot tests more of an inconvenience than a useful tool for this particular project. As the app is fairly simple with only two main pages and not many components being reused, snapshot tests were not very useful during development. The likelihood of unintentionally changing a component is very slim, and it would likely be noticed without snapshot tests as the group has had very thorough code reviews.

For this project, the snapshot tests proved to just be more time consuming. Whenever a component changes, a new snapshot must be generated by running the tests and updating snapshots. This process is easy to forget before uploading code, which would cause the *CI* to fail. Then the snapshot tests would be updated and pushed again. In case of a large commit, over 10 snapshots would be rewritten and pushed, which caused a lot of unnecessary changes in the commits.

Although snapshot tests turned out to not be very useful for this project, they may be worthwhile as the application expands. This may particularly be the case if the future development team uses a different development approach.

### 3.4.4 Integration Testing

For *end-to-end* testing, the group used the *Cypress* test framework. *Cypress* was chosen because all of the group members were familiar with it, and it works well with *TypeScript* and *React*. *Cypress* tests the application from a user's perspective by running the application, clicking on specific parts of the *DOM* and verifying the application's behavior.

As the customer did not have any specific user stories for the project, the group created two different scenarios. The first one tests if the page itself loads the correct data and that the routing works as intended. It also checks for search functionality, error handling and the rendering of the error modal whenever an error occurs. The second scenario tests the use of the graph and the detail view. It checks for nodes in the graph, labels and if the node menu works as intended. It also checks for the correct info in the detail view and whether the buttons in the detail view work as intended. This ensured that the core functionality does not break whenever new code was added to the project. Whenever the tests run *Cypress* will automatically save a video and screenshots of the tests running to a server. If the tests fail this allows the group to easier identify what went wrong without having to read through the logs.

Integration tests are easier to implement at the end of the project when the application is somewhat complete. Similar to snapshot tests, these tests also had to be updated alongside the application. To prevent unnecessary work, the *Cypress* tests were added at the very end of the project. Consequently, these tests did not help much to prevent errors during development. However, the framework and current tests may be useful for further development of the application, particularly if that development will be less agile, more based on user stories and have a more extensive initial planning phase.

## 3.5 Product Development

### 3.5.1 Prototyping

Before developing the application, user tests were conducted with design prototypes. As none of the group members had worked with ontology-based network graphs before, performing user tests to obtain feedback and

[20] https://github.com/ntnu-informatikk-2021/SDG-ontology-visualizer/blob/main/frontend/src/tests/common.test.ts

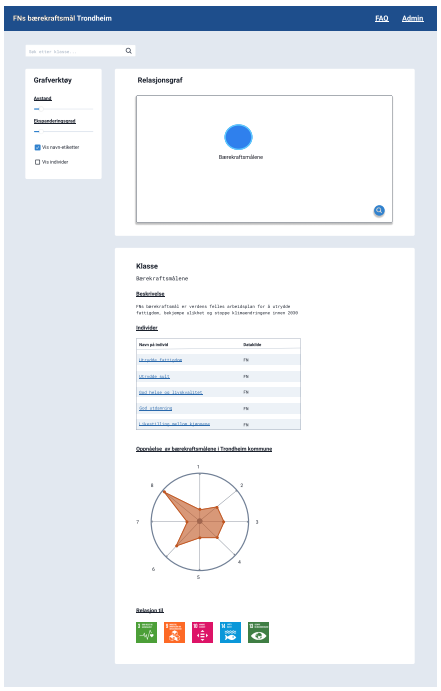


Figure 3.7: The initial frame of the prototype representing the layout and some of the features of the website.

visualize the ideas for the application was necessary. This ensured high usability and a smooth user experience for the website.

Developing a website can also be time consuming and resource-intensive, and it can be helpful to get an accurate impression of what content the website should have beforehand. Creating prototypes using design tools ignores technical details, and therefore freed up time to focus on usability and layout. As a result, the group was able to make rapid changes to the design, and could easily play around with different ideas during the design phase. Prototypes can also reveal what works early on, before having invested a lot of hours in implementing features that potentially will be dropped. [38]

The design prototypes were made using *Figma*[21], which allows for creating high fidelity prototypes with mouse interactions and a navigational structure similar to a real website. However, the prototypes could not be fully interactive, in regards to limitations within *Figma* and the available time.

Two prototypes were developed, one specifically for graph interaction, and one for general layout and functionality. The group wanted to see how the users reacted to the different types of navigation in the application, and whether the users found the website difficult to understand. Additionally, the layout prototype was filled with tools to control the graph to see if the users understood their purpose, and to check if any of the tools could be omitted.

During the development of the prototypes, some problems were encountered. First and foremost, the process was a lot more time consuming than initially expected. To create the graph prototype, there needed to be several frames made to cover all the ways the user could expand and contract it. In the attempt to make realistic prototypes, a lot of details were added that could have been omitted, as they were only contributing to the visual look. To save more time, the group could have instead made low-fidelity prototypes, using paper or story boarding. These do not convey the interactive elements of the website in the same way that high fidelity prototypes can, but the group could still have tested a lot of the key issues with mostly the same level of feedback.[38]

### 3.5.2 First Round of Usability Tests

Figure 3.8, illustrates the layout prototype for the usability test. Since the tests were made at an early stage of the development process, it made the sense to have a qualitative approach rather than a quantitative approach. Therefore five users were considered enough.[38] The user group for the prototype is people working with sustainable development within the municipality. However, only two participating users worked for the municipality including the customer. The rest of the users were all in different age groups and work sectors.

[21] https://www.figma.com/

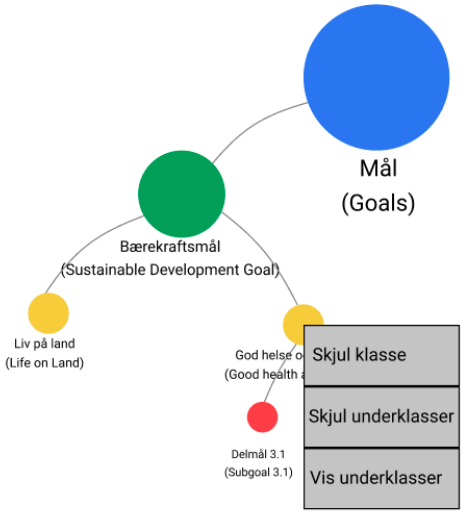


Figure 3.8: One of the frames from the prototype representing the ontology-graph and how to expand and contract it.

This was due to difficulties in finding available people from the municipality, in a short amount of time. Thus, some relevancy in feedback may have been lost.

The tests were conducted in a controlled setting with a predefined set of tasks that the users were told to solve. To reveal their thought process during the test, they were asked to talk out loud. The tests were performed remotely, which meant that the users had to share their screens while solving the tasks. Some of the users did not have a webcam, and were not able to share their reactions or facial expressions. Still, the tests yielded useful verbal feedback, as well as visual feedback through the screen sharing.

The information collected during the tests was mostly directly related to the given tasks, and only reflected the group's initial assumptions of what would be useful to test. As a result, the tests could not reveal how the application would have performed in a real world setting, and did not provide insight towards untested parts of the website. This issue was addressed in the next round of usability testing by reducing the amount of tasks and giving the users more freedom to explore on their own.

To acquire insight towards achieving the prototypes goal, the feedback was evaluated. Most users reported that the language was too technical. As a result, it became a priority to make the concepts more comprehensible. Additionally, the user interface was described as old fashioned and data-heavy. To gain the interest of users without a technical background, the group utilized *Chakra UI* to modernize the website accordingly.

Some usability issues occurred in regards to functionalities within the network graph. This particular issue, became an essential part of the development. An interactive and customizable graph was necessary, where *D3.js*, described in section 3.2.2, was ultimately chosen. The next section describes the results of the applied measures.

### 3.5.3 Second Round of Usability Tests

Unlike the first round of testing, which used sketches from *Figma*, the second round tested the developed website. To evaluate progression, all of the users from the first usability test participated in the second. A few more employees from Trondheim municipality also participated, as the group wanted feedback from users familiar with the sustainability goals.

When most of the planned functionality was implemented, the second round of usability tests, consisting of three tasks, were conducted. Figure 3.9 represents the starting point of all three tasks. The purpose of the two first tasks was to guide the users through the new website with defined objectives. The first task was related to the search functionality, and the second to interaction with icons. These tests provided broader results than the limited interactivity in the first user test. An advantage of giving a specific objective, rather than to guide the subjects, is that it provides insight into how intuitive the navigation feels to the user. The last task was open and the user was free to explore the entire application. This provided insight into what caught each user's attention when navigating the website.


Figure 3.9: Visualization of the front page of the developed prototype, consisting of interactive icons and a search bar.

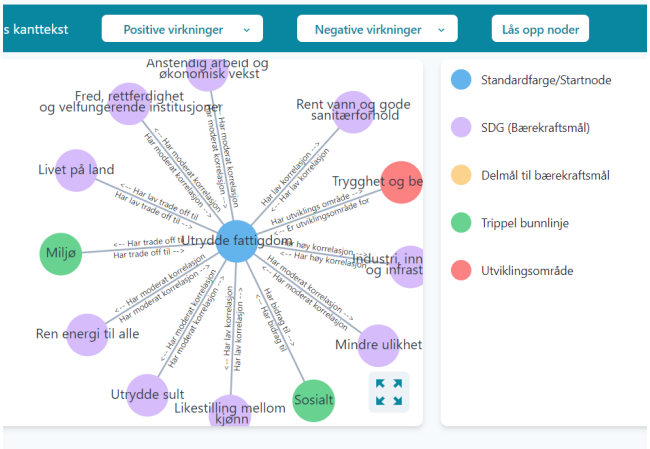


Figure 3.10: Visualization of the network graph of the developed prototype, showing connections between concepts, color scheme for type definitions and filtering options.

The first couple of tests were performed by test subjects not associated with the municipality. Originally, nodes within the graph did not have an associated description, only a color. Therefore, the users where unable to understand the color codes. As a result, the group wanted to incorporate the presented issue, before continuing with additional user tests. This is a clear example on how practical user testing can be during the development. Figure 3.10 visualizes the color description of node types.

Conducting additional tests, with users associated to Trondheim municipality, provided further feedback towards an unlock-all button for locked nodes and other user-friendly functionality. Overall, the response was positive, and the general impression was that the prototype could be favorable in many scenarios. This was great feedback, since the goal from the start was to visualize the strength of ontologies. The acquired feedback was collected and categorized and from there, the group structured the feedback into new issues, and labeled them accordingly.

Through the usability tests, the group learned that users with different backgrounds have different views regarding functionality. Also, implementing some features during the test period allowed for further feedback on new implementation. One could argue that this would create a different base for the testers. However, in this case, the group made the decision because it was no doubt these features needed to be implemented, and thus, it was deemed to be the solution resulting in the most dividend gained from the second round of user test.

### 3.5.4 Design Choices

The prototyping and first round of usability tests made it more clear what core functionality needed to be implemented. Two sections were identified as the most important in the beginning. Firstly, an interactive network graph visualizing the ontology was necessary, to enable users to see the connections and explore the concepts. Secondly, a section containing more information about a single node in the graph was crucial, to let the users gain more knowledge and context than what the graph could initially provide. Other than the graph view and detail view, the application needed to have the usual parts of any website, such as a front page, about page, navbar and footer. To supply more information about the sub goals of the SDGs, a sub goal section was included. Later in the development process, the group decided to implement a search field, which allowed users to search for more specific information.

A significant amount of time went into discussing design choices and finding good solutions, considering the importance of making the website coherent, and ensuring an overall good user experience. During these discussions, the feedback from the usability tests were considered, as well as the continuous feedback from the customer.

During development, the graph component went through major changes. This was mostly due to the group members gaining more experience with *D3.js*, but also because of an increased focus on enhancing the usability and coherency. Researching other attempts at visualizing ontologies revealed a common theme. The graphs were oftentimes difficult to use, and it was even more difficult to gain any meaningful information from them. A big obstacle in this project has been that there is an incredible amount of data in the ontology to be displayed. Followed by an increased difficulty in portraying the network graph, without it becoming incomprehensible. Using the research of other network graphs, as well as trying out different implementations, has been key to progressively enhance the network graph.

Another part of the prototype that has been discussed thoroughly, is the graph tools. In the first usability tests, several users reported that they had difficulties in seeing the purpose of them. It was therefore crucial to have this feedback in mind when implementing graph filters, to avoid further confusion.

During the development of the network graph, three filter options were identified as useful. All three involved giving the users the ability to reduce the amount of information displayed, to fit their individual needs. Two checkboxes were made for sub goals and graph edges, that could toggle them on and off. Additionally, a third filter option was made, allowing users to filter on different levels of positive and negative contributions ( see section 2.4). Initially, two range sliders were made. This constricted the ability to mix different levels, as one could only choose between defined ranges. It also introduced a new usability issue, as it was difficult to make the usage of the sliders intuitive. After some discussion, it was decided to implement dropdowns with checkboxes instead. This gave users more freedom than before, as they could now have different levels of contributions active simultaneously, in contrast to the single presets from the slider.

To summarize, the group has faced several challenges when it comes to designing a visually coherent and user friendly website, that also shows the strength of ontologies in a convincing manner. Through trial and error, the different parts have undergone numerous changes, to get to the current level of usability. There is a fine balance between ease of use and functionality, and it can be difficult to decide which measure to prioritize in each case. This has been a major takeaway, that will be taken into account when working on future projects.

### 3.5.5 Obstacles during Development

#### *Chakra UI*

One issue regarding *Chakra UI* was discussed during the development. Since the styling of a component is altered using style props, the code ends up cluttered and difficult to read. This is especially prevalent when the styling differs a lot from the default theme. A lot of style props also had to be repeated for the same types of components, increasing the amount of code. A way to deal with this problem is to customize the theme and components. This was attempted, but due to difficulties getting familiarized with the feature, development was prioritized. Nevertheless, it will be important to address this issue in future development to increase maintainability and code readability. This is explained in more detail in section 3.6.5.

#### *D3.js*

In the Stout sprint (see section 4.5.2), the graph's code structure was changed. Most notably, the graph was converted from a functional component into a class component. This resulted in greater structure when working with *D3.js* as it can be initialized as an object and have its fields set in the constructor. Another issue that became apparent during the sprint, was the lack of flexibility in *D3.js*'s force simulation. To be more specific - the forces acting between the nodes can only be defined generally, rather than for each individual node. This complicates further work, especially graph readability. The ability to move and lock nodes has been added, though more work is needed to counteract this problem. This is described in section 3.6.4.

#### Deployment

After deploying the *Express* server (see section 3.3.2) the database would sometimes stop responding. The group initially assumed it was related to a timeout issue. However, it was later uncovered that the server ran out of random access memory (RAM), which caused the database to crash. The problem was fixed by upgrading the server from having one gigabytes of RAM, to two.

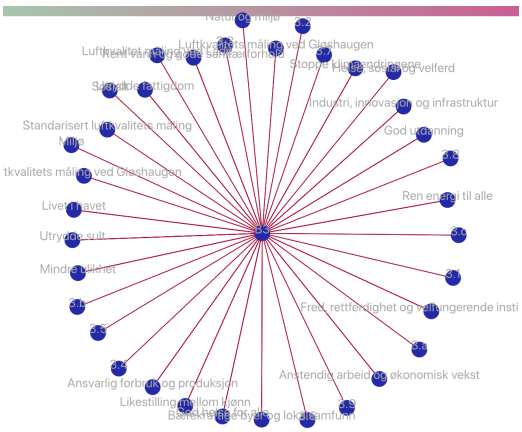

Figure 3.11: The graph component early in the development process

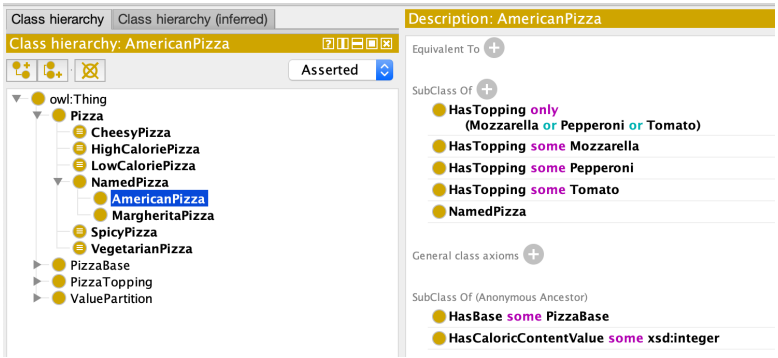


Figure 3.12: The ontology class view of the Pizza ontology tutorial[39] in *Protégé*. Notice the connections created with in the *SubClass Of* - section for the *AmericanPizza*.

### Compatibility Issues between *GraphDB* and *Protégé*

As a part of the initial research phase, all group members completed the *Pizza ontology tutorial*.[39] Figure 3.12 and figure 3.13 illustrate the class view of the concept *AmericanPizza* in *Protégé* and *GraphDB*, respectively. From figure 3.12, connections between other classes have been made, to define what an *AmericanPizza* actually is. However, when the local *Owl.file* from *Protégé* was exported to *GraphDB*, empty nodes were discovered (See figure 3.13, *:node397-405*). In the initial development, this limitation within *GraphDB* generated problems. The method of defining connections used in the *pizza tutorial*, were not supported by *GraphDB*.

A concrete example of the occurring exportation dilemma can be shown below:

1. *[Protégé] AmericanPizza SubClassOf HasTopping some Mozzarella*
2. *[Export] Protégé → GraphDB*
3. *[GraphDB] AmericanPizza SubClassOf :node398*

In other words, *Protégé* is able to convert expression 1 into a separated *subject → predicate → object* expression, while *GraphDB* generates one single *subject → predicate → object* expression. In this case: *HasTopping some Mozzarella* gets converted into a single object, defined as *:node398*.

In accordance with the developed prototype, the group developed the ontology using only single triplet statements. This means that one statement, only contains a *subject → predicate → object* statement, rather than the aggregated statements possible in *Protégé*. However, this limitation can generate implications in future development, see section 3.6.2 for further evaluation.

## 3.5.6 Final Product

Despite the complexity of the task, the final product exceeded the customers expectations. The functionalities in the design prototypes, discussed in section 3.5.2, were implemented in the finished prototype. These features

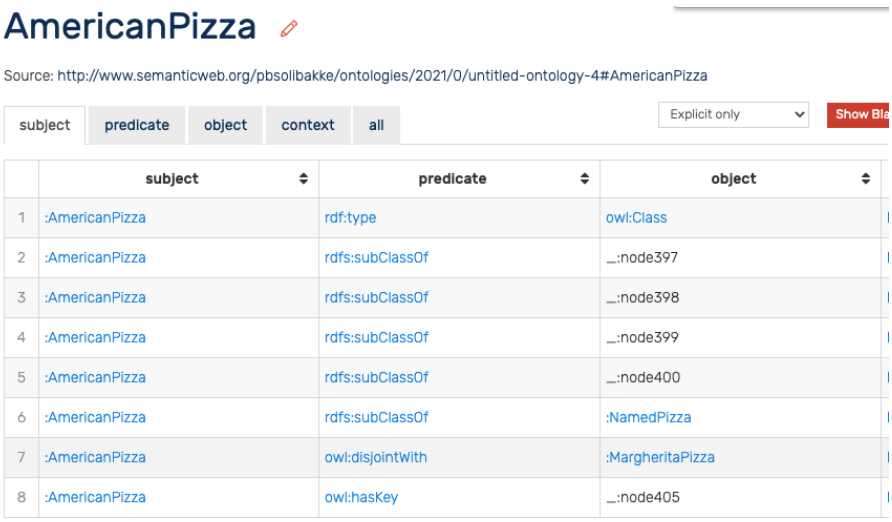


Figure 3.13: The connections for the subject *AmericanPizza* of the Pizza ontology tutorial in *Graphdb*.[39] Notice the object definitions *:node397 - 405*.

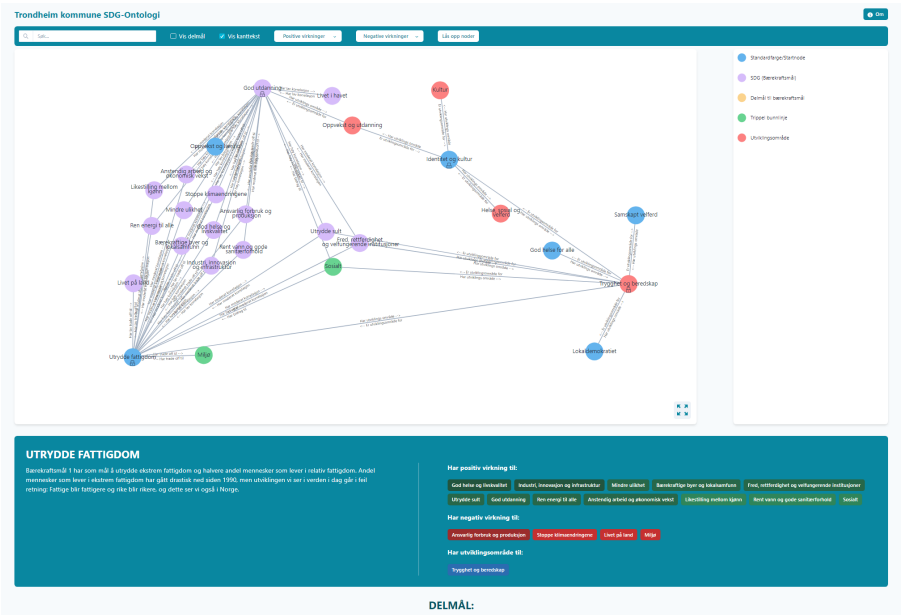


Figure 3.14: The ontology page

have gone through several changes, resulting in a user friendly web application. Nevertheless, the group has realized that the product is far from perfect, as reflected upon in section 3.6.

The frontpage of the website contains a search field and interactive icons for every sustainability goal, as shown in figure 3.9. From here, the user can choose to either search for a specific concept, or use one of the interactive icons as their entry point. Both of these actions, navigates the user to the ontology page, shown in figure 3.14. This is the heart of the application, where all the main functionality resides. The ontology page consists of a network graph and a detail view. Dependent on the users actions, the graph visualizes the concept in question and its connections, which is extracted from the ontology. As the connections point towards other concepts, the user can expand or retract these concepts, based on preferences. Additionally, the graph has a toolbar, which includes a search field and filtering options. Similarly, the detail view provides an in depth description about the same concept. Furthermore, the associated connections are represented as an interactive list. By interacting with these connections, the user can discover their definition. Lastly, the ontology page layout is dependent on conditions, in terms of both available information and type definition. For example, if the selected concept is a sustainability goal, the ontology page also renders a list of the related sub goals.

The web application is built for exploration, and providing a complete coverage of all conditions and functionalities is challenging. The reader is encouraged to experience the application, rather than only reading about it.

## 3.6 Further Work

As discussed previously, the prototype may be further developed. Certain decisions were made based on this assumption, and the focus was primarily on maximizing the development speed, rather than scalability. However, if the product will be developed further with an extended time period, the focus should be on improving performance, maintainability and usability.

### 3.6.1 Expansion of the Ontology

According to both the customer and the group, the structural expansion of an ontology is the most demanding task if the prototype is to be further developed. The ontology currently consists of 13680 lines, and is based on multiple sources to create well-defined concepts and associated connections. However, to create a trustworthy ontology systematizing sustainable development, the group's research is not enough. The first stage of further development would be to accommodate more sources to either verify or discard current definitions for many predicates. Once this has been achieved, the population expansion can commence where implementing indicators, similarly to the method in *SSB's* taxonomy (see section 2.3.1) would be the primary focus.

### 3.6.2 Triplestore

For this prototype, *AnzoGraph* and *GraphDB* were considered. Due to the late response from *Cambridge Semantics* (see section 3.2.1), the free version of *GraphDB* was chosen as it performed sufficiently for this purpose. There is, however, an apparent scalability issue regarding this version of *GraphDB*, is that it only allows two simultaneous queries to the database.[40]

For the future, this will likely be a performance bottleneck, and other *triplestores* should be considered. According to an article referenced by *Cambridge Semantics* on their website,[41] it appears that *AnzoGraph* is noticeably faster at querying and loading large datasets than other *triplestores*. Choosing *AnzoGraph* as the *triplestore* would potentially remove the need for the work-around regarding the compatibility issue with *Protégé* discussed in section 3.5.5. Due to time limitations and priorities, no further research has been done by the group regarding a performance-optimized *triplestore* for a future application.

Furthermore, as the group did not have previous experience with *SPARQL*, the queries made for this project may not be optimal in terms of performance. For example, some of the queries were overfetching, *i.e.* returned superfluous data, which had to be filtered afterwards in the server. One overfetched query is handled in the API call shown in figure 3.15, where the data set from the database query has to go through three filter functions. In the future, all of the data should be filtered in the query itself. No research was conducted regarding the performance of *SPARQL* queries, but for further work the book *Learning SPARQL* should be examined.[42]

### 3.6.3 Server Architecture

As discussed in section 3.2.1, *Express.js* was used for the server, as it was easy to use and conformed better with the *JavaScript*-based front-end. However, *JavaScript* (and subsequently *TypeScript*) is single-threaded,[43] which may cause performance issues with many concurrent users. A programming language with support for multi-threading may be beneficial for a future project.

Retrospectively, the usage of a REST server may not be ideal for ontologies in the first place. Instead of implementing many different endpoints, a query language, such as *GraphQL*,[22] is able to retrieve specific data from a single endpoint. This would prevent overfetching, as discussed in the previous section, and would be ideal for this application. However, the group has neither found a way to combine *triplestores* and *GraphQL*, nor an alternative query language with *triplestore* support.

### 3.6.4 Network Graph

Based on user tests it is clear that using color codes were appreciated to distinguish between different concepts. However, according to accessibility principles, information should not only be conveyed *via* colors, due to issues related to color blindness.[44] In order to address the issue, nodes may be categorized based on different geometrical shapes. While this has not been focused on in this prototype, it should be implemented in future development.

Currently, the strength of the correlation cannot be seen in the network graph itself. To more easily convey information to the user, correlations may be color-coded or otherwise categorized. However, even more colors may give the graph a cluttered appearance, and decrease comprehensibility. Before increasing the amount of

[22] https://grahql.org/

```
export default async (classId: string): Promise<Array<Ontology>> => {
  const node = mapIdToNode(classId);
  if (!node) {
    throw new ApiError(400, 'Could not parse node from the given class ID');
  }
  const query = getRelations(classId);
  const response = await DB.query(query, { transform: 'toJSON' });
  const records = response.records as Array<Record>;
  const ontologies = records
    .map(mapRecordToOntology)
    .map((ont) => addEntityToNullFields(ont, node))
    .filter(isRelevantOntology)
    .filter(isNotLoopOntology)
    .filter(filterDuplicatePredicates);

  return ontologies;
};
```

Figure 3.15: Example data recording code for the ontology, including three filtering functionalities.

information that is visualized through the graph, more research into data visualization and UX should therefore be conducted.

Additionally, there are some issues related to overlapping nodes and edges. To get a better overview of the current network graph, a user has to drag all the nodes further apart manually, as mentioned in section 3.5.5. As the used force simulation library does not support forces on specific nodes and edges, it was difficult to prevent such overlap. For further development, it may be better to create a custom force simulation library to support this functionality. A reassessment of the decisions regarding a visualization library (see section 3.2.2) may also be necessary.

### 3.6.5 Chakra UI

During development, it was found that using *Chakra UI* presents some issues related to code readability and maintainability, due to the use of style props, as mentioned in section 3.2.2. If *Chakra UI* is to be used in future development, it will be beneficial to make custom global themes and component styles, as the correct styling would be applied automatically to new components.

# Chapter 4

# Process

The objective of this chapter is to provide an overview of the group's development process. The first sections describe the project management, including the project plan, choice of process model, the release plan and the risk analysis. Then, the group organization and development environment will be explained. At the end of the chapter, a sprint diary depicting the history of the development process will be presented, along with an overview of work distribution within the group.

## 4.1 Project Management

### 4.1.1 Project Plan

To get an overview of the timeline of the project, a project plan with all of the important events and dates was created, illustrated in figure 4.1. Two group meetings were scheduled every week, Tuesdays *10am - 11am*, and Thursdays *10am - 12pm*. The start and end of each sprint were scheduled every other Thursday. At the end of every sprint, a retrospective meeting was held, which is why Thursday meetings had two assigned hours. A customer meeting occurred every Friday *11am - 1pm*. The supervisor meetings, along with defined deadlines, were entered tentatively in the comments section next to each week in figure 4.1.

### 4.1.2 Process Model

As described in section 3.1, the customer did not have specific requirements regarding the website's design or architecture. To have a lot of flexibility and fewer restrictions to explore and adjust goals during the development, an agile process model was chosen for the project. With the small time frame, it was also essential to be able to start the development as soon as possible. This is usually best achieved with the agile model, rather than the waterfall model.[45]

However, there were some disadvantages to using this model that needed to be addressed. When the project had such loose requirements, the group could easily lose track of the end goal and end up going in a different

| | |
|---|---|
| Customer meeting: | |
| Delivery: | |
| Group meeting: | |
| Vacation: | |
| Sprint start/end | |

| Week | Date | Monday | Tuesday | Wednesday | Thursday | Friday | Saturday | Sunday | Comments |
|---|---|---|---|---|---|---|---|---|---|
| 4 | 25 - 31. jan | | | | | | | | |
| 5 | 1 - 7. feb | | | | | | | Preliminary | Preliminary delivery |
| 6 | 8 - 14. feb | | | | Lager | | | | |
| 7 | 15 - 21. feb | | | | | | | | Supervisor meeting |
| 8 | 22 - 28. feb | | | | Amber Ale | | | | |
| 9 | 1 - 7. mar | | | | | | | | Supervisor meeting |
| 10 | 8 - 14. mar | | | | Stout | Midterm | | | Midterm delivery |
| 11 | 15 - 21. mar | | | | | R&SA | | | Reflection & Self Assessment |
| 12 | 22 - 28. mar | | | | Stout end | | | | Supervisor meeting |
| 13 | 29. mar - 4. apr | | | | | | | | Easter |
| 14 | 5 - 11. apr | | | | Porter | | | | |
| 15 | 12 - 18. apr | | | | | | | | Supervisor meeting |
| 16 | 19 - 25. apr | | | | IPA | | | | |
| 17 | 26. apr - 2. may | | | | | | | | |
| 18 | 3 - 9. may | | | | | Final delivery | | | Final delivery 7/5 |

Figure 4.1: Planned project timeline. Color schemes illustrate different occurrences within each respective week.

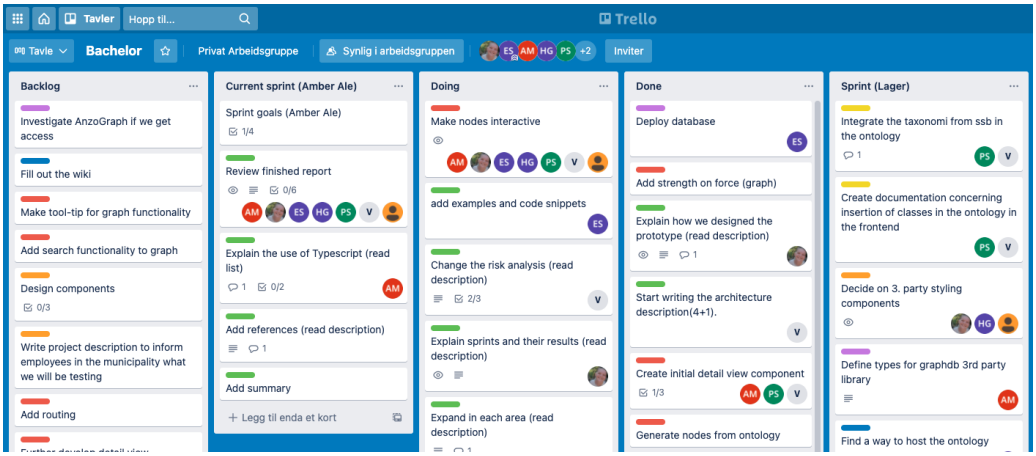


Figure 4.2: A screenshot of the Trello-board during the Amber Ale sprint.

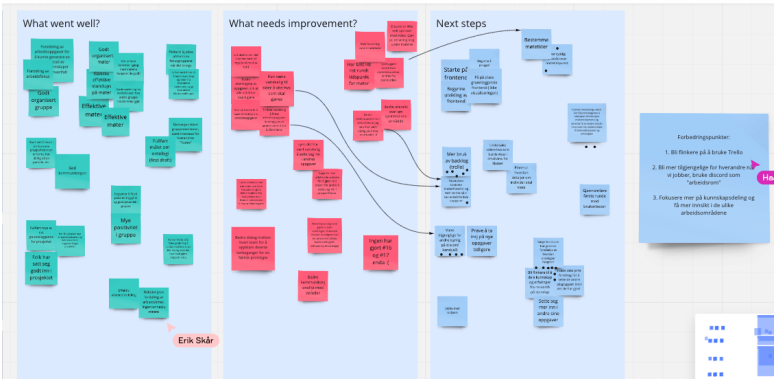


Figure 4.3: A screenshot from the Lager retrospective using Miro.

direction than the customer originally wanted. To prevent this issue, good communication with the customer was established, as discussed in section 4.3.3.

To structure and manage the work throughout the project, the group decided to base the development process model on the *Scrum* framework.[1] Since *Scrum* consists of many elements, a subset that fits the needs of the project was chosen.

Similar to *Scrum*, the project was divided into sprints. An overview of the sprints and their goals can be seen in table 4.1, and each sprint is discussed in detail in section 4.5.2. Each sprint began with a sprint planning meeting, and ended with a retrospective meeting. The group leader facilitated these meetings and prepared a board in *Miro*[2] before the retrospective meetings. The role is further explained in section 4.3.1. In *Miro*, the group members created post-it notes for three categories: "What went well", "What needs improvement" and "Future steps". A screenshot of one of these retrospective meetings is shown in figure 4.3. After discussing the different insights gained from the retrospective, the three most important measures were voted for and addressed in the next sprint. The tasks and issues established in these meetings and during development were organized in a board in *Trello*[3], as seen in figure 4.2.

Another concept of *Scrum* is burn-down charts and time scoping, where the time needed for every issue is estimated by the group.[46] Based on experience from previous projects, the group feels that time scoping usually ends up being inaccurate and more time consuming than beneficial. Therefore, the group decided to not use it.

### 4.1.3 Release Plan

To keep track of the sprints and sprint goals, a release plan was created. Both the goals for each respective sprint and the sprint lengths were altered to some extent during the project development, and diverges from the initial project plan. This is explained in further detail in section 4.5.2. The release plan can be seen in table

[1] https://scrumguides.org/scrum-guide.html
[2] https://miro.com/
[3] https://trello.com/

Table 4.1: The five sprints of the project.

| Sprint | Weeks | Period | Goal |
|---|---|---|---|
| 1 - Lager | 2 | 11.02 - 25.02 | Lay down the groundwork needed to have a smooth start to beginning development of the application in the next sprint. |
| 2 - Amber Ale | 2 | 25.02 - 11.03 | Start developing the application and implement the key functionalities identified in the previous sprint, in order to have a base to further improve upon. |
| 3 - Stout | 4 | 11.03 - 08.04 | Add additional features and enhance design and functionality to make the application more coherent and user friendly. |
| 4 - Porter | 2 | 08.04 - 22.04 | Finalize the development and add finishing touches to product, based on feedback from usability tests. |
| 5 - IPA | 2 | 22.04 - 06.05 | Prepare prototype for final delivery and ensure that the delivered product is ready for further development by the customer. |

4.1.

## 4.2 Risk Evaluation

To prepare the group for all possible risks, a thorough risk analysis was made. All of the risks were given an ID, a name, its consequence and the likelihood of the scenario occurring. The consequence and likelihood have a corresponding rank, which is visualized in the left risk matrix seen in figure 4.4. To minimize both the corresponding consequences and likelihood of each risk happening, the group identified and applied countermeasures for each risk. The countermeasures are defined in table 4.3 and described further in section 4.2.2. The right matrix in figure 4.4 shows the severity of each risk after applying the countermeasures. As the analysis was created at the start of the project, this section is written in future tense.

### 4.2.1 Risk Categories

#### Group Risks (R.1)

A group member not being available may cause huge delays towards the development, since the member might have detailed knowledge about a specific part of the project. This might occur for several reasons, such as: sickness, lack of motivation, etc. Disagreement between group members regarding the product can result in delayed delivery, and lower the group morale. However this will most likely have no severe impact. It is likely that there will be some disagreements, because of different perspectives. Uneven distribution of workload, in which a group member decides to contribute to a large extent without inclusion of other members, or its counterpart where a group member contributes to a low extent, may have a large impact. Furthermore, differences in ambition between each group member, might promote an uneven workload. If a member quits, this will lead to more work for the rest of the group. The consequences may vary depending on the individuals work load and project related knowledge. This subject is mandatory for the degree (last semester), thus every student should be fairly motivated to complete the subject. However, unpredictable events could potentially occur.

#### Customer Risks (R.2)

Lack of communication with the customer will most likely affect the group's efficiency and may cause efficiency to decline if work cannot be performed or is executed incorrectly because of misunderstandings with the customer. Additionally, decision-making on important choices will be difficult without input from the customer. This might happen due to unforeseen events, though the customer has so far proved to be quite accessible. Unrealistic demands or low expectations from the customer can result in a limited prototype. Furthermore, the group's learning outcomes may be limited, followed by a dissatisfied customer. The customer has had several projects with students in the past and has experience in terms of what students can manage during a bachelor's degree, which will contribute to minimizing the likelihood of this happening. Misunderstandings between the customer and the team might result in the group developing a product that does not match the customer's expectations.

The project requires an understanding of new concepts for the group. While learning these concepts there is a chance for misunderstandings to occur. If the ontology is constructed incorrectly compared to the intended goal, this risk can also set the group back several weeks. Additionally, the problem can be discovered when there is not enough time to make the necessary changes. Limited understanding of how ontologies work may lead to errors in the design. Since none of the group members have previous experience with ontologies, there is a fair chance this may happen.

#### Task Risks (R.3)

Limitations in the chosen technology stack can set the group back several weeks or even prevent the project from being completed. It is challenging to know in advance exactly what kind of technologies and solutions will work best for the project, therefore problems may occur. If the project does not contain the required tools for displaying the useful properties of the ontology, the value of the product will be greatly reduced due to the front-end not being capable of exploiting the power of the ontology. Considering that the front-end solutions is based on technologies that are familiar to the entire group, it is unlikely that this will happen.

#### Time Risks (R.4)

If the product would not be delivered on time, it would result in an unsatisfied customer and most likely a lower grade achieved by the group. At the current stage the product is likely to finish in time. The risk of the deadlines throughout the semester are not met has the consequence with the project falling behind, thus the final delivery might not be finished within the deadline. Some of the technology is new to the group members, and it is therefore hard to estimate how much time is needed for different tasks.

#### Scope Risks (R.5)

The customer has warned the group that ontologies can rapidly become large and complicated. Therefore, misunderstanding the size of the project can lead to an incomplete product within the deadline. With no prior experience with ontologies, the group does not know how much time is needed to gain the necessary knowledge and to create the ontology, which makes it hard to estimate.

#### Covid-19 (R.6)

Quarantine or high infection rates causing group members to work remotely might lead to lack of motivation and more distractions. Also by working remotely, cooperation may be more difficult. Quarantine is unpredictable and can happen to everyone, therefore the risk is considered likely. Sickness may cause group members to be unavailable, and if a member possesses knowledge no one else has, this may cause a huge delay to the project. Still, it is unlikely that any group member will be sick for a longer amount of time. The Corona virus is extremely contagious, but there are low infection rates in Norway and Trondheim. When having online meetings, communication between group members and customer can have a lower quality. This may be due to technical problems or less response to what is being expressed. It can also be more difficult to perceive what people think and feel. One of the group members is in the high-risk group, and therefore the group will have to adapt to this by working remotely (at least when the individual is included). As with sickness, loss of internet can make a group member partly or completely unavailable because of the necessity of remote work. If the problem remains unsolved for several days, this can cause significant delays as described in the R1a in the risk table. A stable internet connection is almost always available in some way (4G, fiber, coax, ADSL). However it can happen by power failure etc.

### 4.2.2 Countermeasures

There are four main types of countermeasures that can be used to minimize both the outcome of consequences and likelihood. They are *avoid*, *reduce*, *transfer* and *accept*. Avoid is used by stopping or changing the activity. Reduce is utilized to minimize the outcome. Transfer is applied when there is a need to transfer the responsibility to someone else completely or partially. Examples of transfer is insurance or sharing information. Accept should only be used when none of the three other options are available, and the consequence or likelihood must be accepted.

#### Group Risk (R.1)

Group member not available (R1a) - If a group member becomes unavailable the group can *transfer* the consequence by making sure at least two members has enough knowledge on each subject in order to operate it

Table 4.2: Risk table showing the different risks associated with the project. Consequence and likelihood entries are color coded to more easily differentiate them at a glance.

| ID | Risk name | Consequence | Likelihood |
|---|---|---|---|
| R1a | Group member not available | High May cause huge delay to the project | Unlikely A result of: sickness, lack of motivation etc. |
| R1b | Disagreement between group members. | Medium May result in delayed delivery, and lower group morale. | Likely Some disagreements, due to different perspectives. |
| R1c | Uneven group distribution of workload. | Medium Might have severe impact if a member does not complete their given tasks. | Unlikely Different ambitions is supporting that uneven workload may occur. |
| R1d | A group member quits. | Critical Lead to more work for the remaining group. | Very unlikely Unpredictable events may occur. |
| R2a | Lack of communication | Critical It can be hard to make certain decisions without getting input from the customer. | Unlikely Unforeseen events can occur and make the customer less accessible. |
| R2b | Demands and expectations too great/low. | High May result in poor grade and reduce the group's learning outcome. | Very unlikely The customer's previous projects helps setting realistic expectations. |
| R2c | Misunderstanding between customer and the team. | High Product does not meet the customer's expectations. | Unlikely While learning new concepts there is a chance for misunderstandings to occur. |
| R3a | Technology risks | Critical Set the group back several weeks or prevent the project from being completed. | Unlikely Uncertainty with the technology used might cause problems. |
| R3b | The ontology is constructed incorrectly in relation to the goal. | High Can be discovered when there is not enough time to make the necessary changes. | Unlikely Limited understanding of how ontologies work, may lead to errors in the design. |
| R3c | The project failing to make use of the ontology. | Critical The product value will greatly be reduced. | Very unlikely Technologies familiar to the entire group, makes this unlikely. |

Table 4.2 continued: Risk table showing the different risks associated with the project. Consequence and likelihood entries are color coded to more easily differentiate them at a glance.

| ID | Risk name | Consequence | Likelihood |
|---|---|---|---|
| R4a | Product not delivered on time. | Critical Unsatisfied customer. | Unlikely Unlikely at current project stage. |
| R4b | Deadlines throughout the semester are not met. | High This will result in the project falling behind. | Unlikely New technology makes it hard to estimate workload for different tasks. |
| R5a | Misunderstanding the size of the project. | High This may result in an incomplete product within the deadline. | Unlikely No experience with ontology's makes estimation difficult. |
| R6a | Quarantine causes group members to work remote. | Low Lack of motivations and working remote makes cooperation more difficult. | Likely Quarantine is unpredictable and is therefore likely. |
| R6b | Sickness makes group members unavailable. | High Can cause a huge delay to the project. | Unlikely Low infection rates in Norway and Trondheim makes this unlikely. |
| R6c | Online meetings. | Medium Can result in lower meeting quality and more difficult to perceive what people think and feel. | Very likely Adept to online meetings since a member is in the critical group, |
| R6d | Group member loses internet. | High Result in partly or completely unavailable. | Very unlikely A stable internet connection is almost always available in some way. |

alone. This will allow the group to continue working on every topic even with fewer members present. Also the group can *avoid* the likelihood by establish a motivating work environment.

Disagreement between group members (R1b) - If disagreements occur, the group will *reduce* the consequence through the processes as stated in the Group contract (see Appendix B: figure 5.2 and figure 5.3). Disagreements will be discussed in a meeting with every member present. If they are not solved, it will be discussed with the supervisor. Further, by being open minded and seeing the problem from a different perspective, and attacking the problem, not the person, the likelihood will be *reduced*.

Uneven group distribution of workload (R1c) - By using the same countermeasure as in "R1a", *transferring* the risk will minimize the consequences because more than one member sits with the knowledge. This likelihood can be *reduced* by logging work hours and have an average of 20 hours each week. Logging will produce statistics which can be compared within the group. The group has also discussed expectations regarding the project to make sure everyone is on the same page.

A group member quits (R1d) - In the event of a group member quitting, by *transferring* as in "R1a" more than one group member sits with the same knowledge in the case of such an event. Given the reason a group member quits due to frustration and bad team-chemistry, the likelihood can be *reduced* by establishing a healthy work environment within the team. Unpredictable reasons are hard to countermeasure, and will therefore have to be *accepted*.

**Customer Risks (R.2)**

Lack of communication (R2a) - By creating an agenda before, and a summary after each meeting, communication through meetings will be more clearly structured. This gives a source of documentation of the customer's previously expressed wishes and requirements, and thus *reducing* the consequences if the customer becomes temporarily unavailable. Getting a clear impression of what the customer wants has also made it easier for the group to make independent decisions. The group has created easily accessible ways of communication by using Slack and weekly meetings, *reducing* the likelihood of lack of communication.

Demands and expectations are to great or low (R2b) - Weekly progress updates given the customer's expectations will give both the group and the customer time to adjust expectations depending on the progress, *reducing* both consequences and likelihood of this happening.

Misunderstanding between customer and the team (R2c) - Countermeasures for "R2a" and "R2b" will also be *reducing* the consequences of misunderstandings. Good communication is key to avoid critical misunderstandings. Writing meeting agendas and summaries will also be *reducing* likelihood of misunderstandings.

**Tasks Risks (R.3)**

Technology Risks (R3a) - By researching technologies and designing layout prototypes, the group can more easily discover the necessary changes in the chosen technology stack early on and therefore only minor changes may be necessary later on. Research and layout designs will also contribute to minimizing the need of technology changes, thus *reducing* both the consequences and likelihood.

The ontology is constructed incorrectly in relation to the goal (R3b) - By beginning at a logical starting point, and continuously consulting with the customer (who has experience with ontologies), the group will *reduce* the risk by making sure any errors in the ontology are quickly discovered.

The project user interface does not display or contain the tools for using the ontology in a way that makes it useful (R3c) - This is an important part of the product, it would be critical if the group could not complete the task and have to *accept* the consequences. However, extensive documentation about what went wrong will be useful for the customer. Still, preventing this is the preferred option, and developing and testing the front-end throughout the project period will help *reduce* the likelihood.

**Time Risks (R.4)**

Product not delivered on time (R4a) - The consequence of not completing the project in time is not ideal, however we can *reduce* the outcome by at least delivering an MVP that the customer can use. By doing proper project planning along the way, we can more easily predict the outcome, thus *reducing* the likelihood. This will be done by regular meetings and sticking to the schedule.

Deadlines throughout the semester are not met (R4b) - By designating time specifically for finishing deliveries, the deficiencies within each delivery will be *reduced*. Also, by applying the same countermeasure as in R4a with proper project planning, the group can *reduce* the likelihood of wrongfully estimating necessary workload to reach the deadlines.

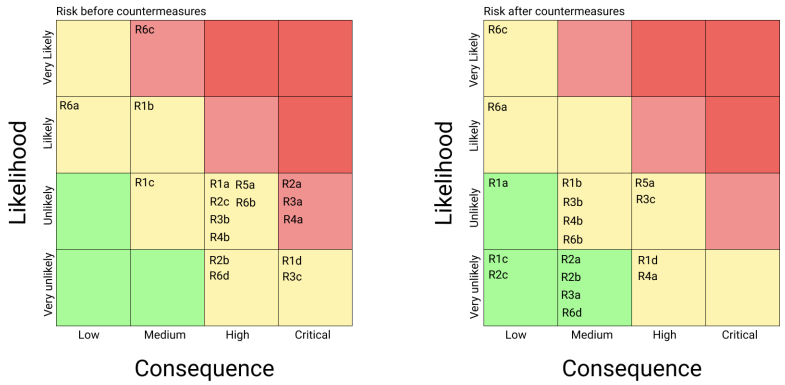


Figure 4.4: Risk matrices showing the likelihood - consequence correlation for the risks from table 4.2 before and after applying the countermeasures from table 4.3.

#### Scope Risks (R.5)

Misunderstanding the size of the project (R5a) - This risk can be *reduced* by thoroughly discussing the task with the customer. Also, the group can make sure misunderstandings are quickly discovered and corrected. With proper project planning and a continuous dialog with the customer, we can have a better understanding of the task and prevent misunderstandings, thus the likelihood will be *reduced*.

#### Covid-19 (R.6)

Quarantine causes group members to work remote (R6a) - The group has to *accept* quarantine, however this should not have any severe impact as the group works remotely as described earlier. Quarantine can happen to anyone whenever and it is therefore hard to countermeasure.

Sickness causes group members to be unavailable (R6b) - Applying the same countermeasures as in "R1a", will minimize the consequences because more than one member sits with knowledge in case a group member gets sick. Thus *transferring* the risk. The group members will *avoid* being exposed to potential sources of infection by following the national and municipal infection control measures and recommendations. The national infection control measures. [47]

Online meetings (R6c) - The group members will *reduce* the consequences by using the camera during meetings to be able to show facial expressions. If possible, the microphones will also be kept unmuted during meetings. Attendees will then not just hear the substantive responses, but also the 'B-roll' of subtle feedback that helps the group feel the people in the room and evaluate the pulse of the conversation. Since the group consists of a member in the critical group, it has been decided to *accept* carry out all meetings remotely.

Group member loses internet (R6d) - Countermeasures for "R1a" would also apply here, transferring the knowledge to more than one person. It would still be very unpractical if the connection remains lost for several days, limiting possibility to partake in meetings and contribute on tasks. The group members should *avoid* traveling to places without stable internet connection.

## 4.3 Group Organization

### 4.3.1 Roles and Responsibilities

During the early stages of the project, the group decided to avoid defining roles for the members of the group. This was done to facilitate an agile work dynamic, and to enable group members to switch between different areas of focus, in accordance with the chosen process model (see section 4.1.2). The role of group leader was mandatory for the project, and was therefore the only defined role.

The role of group leader included certain defined responsibilities, such as participating in lectures and Q&A-sessions associated with the course. Additionally, the group leader would take responsibility of leading and facilitating customer, supervisor and internal meetings.

Even though the group never defined any other roles, the members naturally developed greater expertise within certain areas during the development process. For example, some members focused more on the design, some focused on the graph simulation, and some focused on the ontology and the back-end. This sort of expertise turned out to be useful for the team, as the different groups could get familiar with a specific topic, increasing

Table 4.3: Countermeasures to prevent or minimize the consequences of the associated risks from table 4.2.

| ID | Risk name | Consequence and countermeasures | Likelihood and countermeasures |
|---|---|---|---|
| R1a | Group member not available | Low (Transfer) | Unlikely (Avoid) |
| R1b | Disagreement between group members. | Medium (Reduce) | unlikely (Reduce) |
| R1c | Uneven group distribution of workload. | Low (Transfer) | Very unlikely (Reduce) |
| R1d | A group member quits. | High (Transfer) | Very unlikely (Reduce) |
| R2a | Lack of communication | Medium (Reduce) | Very unlikely (Reduce) |
| R2b | Demands and expectations are too great/low. | Medium (Reduce) | Very unlikely (Reduce) |
| R2c | Misunderstanding between customer and the team. | Low (Reduce) | Very unlikely (Reduce) |
| R3a | Technology Risks | Medium (Reduce) | Very unlikely (Reduce) |
| R3b | The ontology is constructed incorrectly in relation to the goal. | Medium (Reduce) | Unlikely (Reduce) |
| R3c | The projects user interface does not display or contain the tools for using the ontology in a way that makes it useful. | High (Accept) | unlikely (Reduce) |
| R4a | Product not delivered on time. | High (Reduce) | Very unlikely (Reduce) |
| R4b | Deadlines throughout the semester are not met. | Medium (Reduce) | Unlikely (Reduce) |
| R5a | Misunderstanding the size of the project. | High (Reduce) | Unlikely (Reduce) |
| R6a | Quarantine causes group members to work remote. | Low (Accept) | Likely (Accept) |
| R6b | Sickness causes group members to be unavailable. | Medium (Transfer) | Unlikely (Avoid) |
| R6c | Online meetings. | Low (Reduce) | Very likely (Accept) |
| R6d | Group member loses internet. | Medium (Transfer) | Very unlikely (Avoid) |

| Date | Week | Number of Hours | Category | Description | GitHub issue ID (if programming) | | |
|---|---|---|---|---|---|---|---|
| 22.01.2021 | 3 | 3 | Møter | Oppstartsmøte med kunde | | | |
| 26.01.2021 | 4 | 2 | Research | OWL pizza tutorial | | | |
| 27.01.2021 | 4 | 3 | Forelesning | | | | |
| 27.01.2021 | 4 | 4 | Research | OWL øl-ontologi | | | |
| 28.01.2021 | 4 | 2 | Annet | Lage timeliste + grafer i Excel | | | |
| 28.01.2021 | 4 | 2 | Research | Gjøre ferdig øl-ontologi | | | |
| 29.01.2021 | 4 | 2 | Møter | Kundemøte og planlegging i gruppa fram til neste tirsdag | | | |
| 29.01.2021 | 4 | 4 | Research | Utforske GraphDB og sette opp Kotlin server for å sende requests til database | | | Number of hours this week |
| 01.02.2021 | 5 | 1 | Møter | Kjapt møte med Per og Erik for å planlegge backend | | | 15 |
| 02.02.2021 | 5 | 2 | Koding | Sette opp enkel express server og lage database.js modul | #2 | | |
| 03.02.2021 | 5 | 2 | Koding | SPARQL, sette opp enkel frontend for demo | #3 | | |
| 04.02.2021 | 5 | 3 | Koding | Mer SPARQL og server endpoints | #3 | | |
| 04.02.2021 | 5 | 2 | Møter | | | | |
| 05.02.2021 | 5 | 1,5 | Møter | Kundemøte | | | |

Figure 4.5: One group member's timesheet table.

the efficiency of the development process. A possible risk with this approach, however, is that group members working on one part of the application may not understand how the rest of the application works. To avoid this issue, the group had two stand-up meetings every week, during which everyone described what they had worked on since the last meeting.

### 4.3.2 Group Collaboration

The collaboration within the group has mostly been frictionless throughout the project. There has been a focus on stimulating good communication and cooperation through regular meetings, including the meetings with the customer. *Slack*[4] was also used frequently.

However, there have been some minor issues throughout the project. As a result of indistinct specifications regarding meeting times, there were a few instances where one or more group members were unaware of when a meeting would start. This reduced the productivity to some extent and caused some frustration. To solve the issue, the meeting schedule was reviewed and regular meeting times were clarified. After this countermeasure, there were no further issues regarding attendance.

As explained in section 4.2.1, the meetings have been digital due to the Covid-19 pandemic. A possible issue with such meetings is more difficult communication due to a lack of visual feedback. Additionally, the group members may not develop the same interpersonal connection to each other and the project as a whole. However, the group did not feel that this was a problem, as meetings were arranged with webcams and there was a generally friendly tone within the group.

To facilitate transparency and provide an overview of the time spent on different categories of work, a common spreadsheet for time tracking was created with *Google Sheets*.[5] Each group member had its own sheet where work hours were added and categorized, which made sure everyone was aware of their own contribution compared to the rest of the group. Additionally, each member's sheet had a cell showing the number of hours worked this week, with the color of the cell changing gradually from red to green as the work hours got closer to the target of 20 hours. While this may seem silly, it made time tracking more enjoyable and helped motivate the group to reach the target work hours. A screenshot of one team member's sheet is shown in figure 4.5.

### 4.3.3 Customer Collaboration

Collaboration with the customer has been excellent during the entire project period. In the initial stages, the customer was contacted by email, but communication through *Slack* was quickly established. Weekly meetings were held every Friday, keeping both parties updated on the development. Here, any uncertainties were clarified quickly. Occasionally, more frequent customer meetings were necessary, such as during usability testing (see section 3.5.1) and before the presentation for the *The Norwegian Digitalisation Agency* (see section 4.5.3). Such flexibility was possible because the customer was available to answer questions and requests on short notice.

### 4.3.4 Supervisor Collaboration

Manos Papagiannidis was assigned as the group's supervisor for the project. Similar to the customer, the supervisor mostly communicated with the group through *Slack*. However, communication with the supervisor was not as frequent as with the customer, and meetings were not held regularly. As there were no noteworthy issues regarding the group collaboration, most of the inquiries made to the supervisor were related to the report. These inquiries were usually responded to quickly.

[4] https://slack.com/intl/en-no/
[5] https://www.google.com/sheets/about/

While the supervisor provided some valuable feedback regarding the grammar and language of the report, the group felt that the feedback regarding the overall structure was limited. However, two meetings were arranged with the supervisor to discuss the feedback in detail. During these meetings, the group felt that the feedback was more thorough, as it was easier to ask follow-up questions. Retrospectively, arranging more meetings, as opposed to simple textual communication, could have improved the collaboration.

## 4.4 Development Tools and Standards

### 4.4.1 Coding Standards

With a team of seven developers it is important to follow the same coding standards. The group chose *Airbnb's* standard[6], as it is a well-defined and commonly used standard within the *JavaScript* community which all of the group members were familiar with.

To ensure that the coding standard is followed, linting and code-formatting tools had to be configured. The group decided to use *ESLint*[7] for linting and *Prettier*[8] for code-formatting, as these are widely used tools which the group has experience with from previous projects. The group did not require a specific development environment, as long as the environment supported *ESLint* and *Prettier*. Due to personal preference, all of the group members chose to use *Visual Studio Code*[9].

### 4.4.2 Git Standards

To ensure that the code was peer reviewed and maintained high quality, committing directly to the main branch was prohibited. Furthermore, pull requests required at least one approval to be merged.

Initially, the group decided that branches should be named in the format "*[issue number]-name-of-issue*". However, as *GitHub* counts pull requests as issues, this created a mismatch. For example, a pull request solving issue #9, named "Add caching to CI", would be given the ID #10 and named "9-add-caching-to-CI". This creates unnecessary confusion, and the standardized branch name was later changed to "*name-of-issue*".

Another problem regarding *Git* standards is that some pull requests have closed several issues. This is a bad practice, as it makes it harder to figure out where a change has been implemented based on the commit history. However, this problem has been rare and was not further addressed.

### 4.4.3 Documentation

The difference between enough and too much documentation is a fine line. To ensure that the project is not cluttered with superfluous documentation, the group established certain rules regarding documentation. The rules were discussed with and agreed upon by the customer.

As described in section 3.2.3, *TypeScript* is strictly typed. This makes the code more self-explanatory, and with good coding practices, documentation is often not necessary. However, if the code is ambiguous, the code should be documented.

Initially, the plan was to document all of the *API*-endpoints in a separate document. When working in the front-end, group members could easily find the correct endpoints without having to switch context to the back-end code. However, at a later stage in the project, the group realized that the server would have few endpoints, and that they were self-explanatory. Thus, externally documented *API*-endpoints were deemed unnecessary and removed.

### 4.4.4 Ontology Editor

In general, ontologies are stored as one large file containing all the triplets as individual lines. Manually editing this file is both overwhelming and inefficient. Consequently, several ontology editors have been discussed to simplify the ontology development.

Five ontology editors were discussed, which were chosen as the most promising from *W3C*'s list of ontology editors,[48] an article by B. Kapoor *et al.*[49] and recommendations from online forums and the customer. However, two of them, namely *OntoStudio*[10] and *TopBraid*[11], were discarded quickly because they are not

[6] https://airbnb.io/javascript/
[7] https://eslint.org/
[8] https://prettier.io/
[9] https://code.visualstudio.com/
[10] https://www.w3.org/2001/sw/wiki/OntoStudio
[11] https://www.topquadrant.com/products/topbraid-enterprise-data-governance/

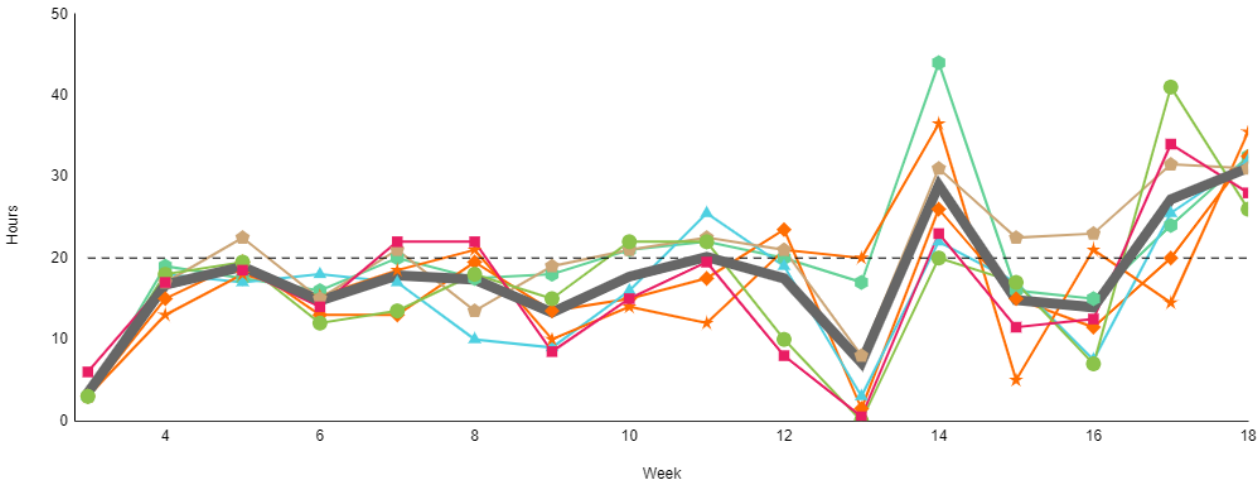


Figure 4.6: Number of work hours per member per week. The dashed horizontal line is the target of 20 hours. The thick gray line is the group's average hours.

open-source and required a licence. The three remaining editors were *Apollo*[12], *Swoop*[13] and *Protégé*[14]. The first two are open source, but provide less functionalities, such as interoperability between ontology editors, than *Protégé*.[50] Additionally, *Protégé* supports more import and export of languages, such as *OWL*, *OWLOC*, *Java* and *HTML*, which are not supported by *Apollo* and *Swoop*.[49]

A final ontology editor the group considered was an online variant of *Protégé* called *WebProtégé*[15]. This would let all group members view and edit the ontology at the same time. However, *WebProtégé* was unreliable, as changes done by one group member were not automatically updated for other members, and required a manual refresh. Furthermore, *WebProtégé* had limited functionality compared to the local editor, such as help functionalities, views and inferencing properties. *Protégé* was the first editor the group used to understand and implement ontologies, and it provides a better functionality compared to the previously discussed editors and its web counterpart. Thus, the *Protégé* editor was chosen for the remainder of the development.

With a local editor, it was difficult to include the whole team in the work related to the ontology. As a result, the ontology was mainly developed by a few members of the group. Because of the pandemic, the group members had to develop the ontology while screen sharing.

## 4.5 Development Process

### 4.5.1 Work Distribution

According to section 3.2 in the group contract (see Appendix B: figure 5.2 and figure 5.3), the expected workload for the project was 20 hours on average per week per group member. Figure 4.6 shows that the group as a whole has worked relatively evenly each week. The figure also shows that everyone in the group has contributed relatively equally.

Figure 4.7 shows the group's work hours categorized per week, and describes how the project has been divided into three phases. The research phase can be observed by the high number of research hours in week 4 and some hours the following few weeks. During the design phase, some of the work, namely the prototype design in *Figma*, was categorized as "other", while the initial configuration of the application was categorized as "programming". The figure shows that the design phase lasted from week 6 to 8. Following the design phase was the development phase, which lasted until week 16 and can be clearly seen in the figure. Lastly, the figure shows a clear peak in the "report" category at the end of the project.

As shown in figure 4.6, the group had the fewest work hours (except for the Easter week) in the first week of the development phase (week 9). As discussed in the following section, this issue was addressed at the retrospective meeting in week 10. The figure shows that the average work hours were increased for the rest of the project.

[12] https://genomearchitect.readthedocs.io/en/latest/index.html
[13] https://www.w3.org/2001/sw/wiki/SWOOP
[14] https://protege.stanford.edu/
[15] https://webprotege.stanford.edu/

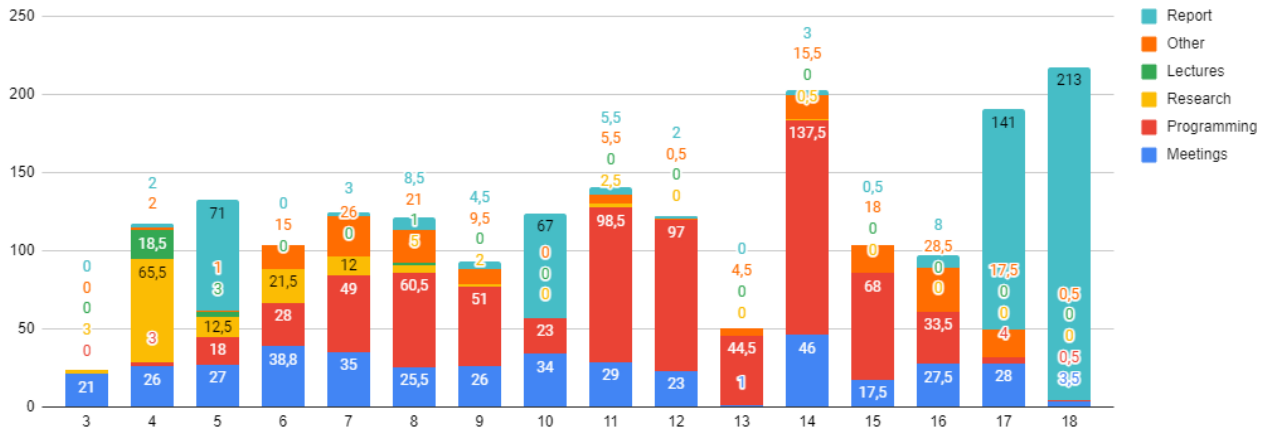


Figure 4.7: Number of hours per category per week.

A complete overview of the group's hours can be found in Appendix C: figure 5.4, and an overview of the work hours for each group member in figure 5.5. This data is the basis for the graphs in figure 4.6 and figure 4.7.

### 4.5.2 Sprint Diary

#### Lager

The first sprint started in week six, and the main goal was to prepare to begin development of the application in the next sprint. This involved configuring the back-end and front-end, researching third party libraries, developing the ontology, and creating prototypes for usability testing. The group was divided into smaller units, and were able to finish most of the planned work. During the sprint, *D3.js* was chosen to create the graph component. Also, the component library *Chakra UI* was decided to be used for the rest of the front-end components. More information regarding *D3.js* and *Chakra UI* is located in section 3.2.2. Moreover, a lot of progress with the ontology was made, adding documentation concerning insertion of classes. The prototypes were finalized as well, although the tests had to be pushed forward to the next sprint, as the development took more time than initially expected.

Both positive and negative experiences from the sprint were expressed in the retrospective meeting. The main issues were related to communication and project management. The first measure was that the *Trello*-board should be used and updated more frequently in the future. Secondly, the group members should be more available during work hours, in order to make it easier to contact other members for assistance. To facilitate this, a virtual meeting room should be used while working in the future, to make up for the lack of a physical work space. Lastly, the group members should focus more on sharing knowledge and gaining insight on the different parts of the project during stand-up meetings, which would increase the collective understanding of the product.

#### Amber Ale

The second sprint started in week eight, and focused on starting development of the front-end application. The goals for the sprint were to visualize the ontology, create a base layout, conduct the usability tests and evaluate the feedback. It began with programming the graph component with the whole team, commonly known as mob programming.[46] This was useful to let everyone experience and learn how *D3.js* worked early on. Additionally, as the usability tests were finished, a lot of valuable feedback was available during the development. The group was also able to connect the ontology to the front-end application, enabling the information to be visualized in the graph and detail view components.

At the retrospective meeting, the group acknowledged that the level of motivation and teamwork within the group was high. Most of the measures from the last sprint were improved by using *Discord* as a digital workroom, being more available, and having an increased focus on sharing knowledge between work areas. Nevertheless, there was a potential for improvement in certain areas. Firstly, the group recognized that better notes could be taken during development to not forget why decisions were made. This was noticed when reviewing decisions while writing the report draft. Secondly, the mob programming in this sprint allowed for more knowledge sharing, but also contributed to an inefficient development process. To mitigate this, there should be more focus on programming in pairs rather than in bigger groups. This will keep the positive effects of team programming while also dividing the work in an efficient manner. Finally, despite making improvements from the last sprint,

the group did not have an established routine for updating the *Trello*-board. As this measure had not improved from the last retrospective, it needed more work.

**Stout**

The third sprint began week ten. In this sprint, the main goals were to make the application more coherent and user friendly by adding additional features and enhancing design and functionality. In the first part of the sprint, the ontology was expanded with *SSB's* taxonomy for sustainable development. This taxonomy will work as an important foundation in further expansion of the ontology. The general layout and design of the front-end and its components were improved, and more functionality was added to the graph and detail components. As the second round of user testing was scheduled in the beginning of the next sprint, several features of the application had to be finalized to get useful feedback.

The group decided to extend the sprint by two weeks to include Easter. As several of the group members planned to work during the holidays, it made sense to have a longer sprint that ended after Easter, rather than beginning a new one right before.

At the retrospective meeting, the group concluded that the pair programming was an efficient technique, with tasks being completed faster than before. The last sprint's measure to update the *Trello*-board was also improved during this sprint, making it a lot easier to keep an overview of the development. However, the group found it cumbersome to keep *Github* and *Trello* synchronized. Furthermore, similar to the last sprint, better notes could be taken during the development.

There was a lack of time during the Thursday meetings to do proper sprint planning, which resulted in some uncertainty regarding which tasks needed to be done in the first part of the sprints. As a consequence, the previous sprints had a slow start with most of the work being completed in the second half. Improving the sprint planning will therefore be prioritized in future sprints. Certain technical obstacles were also encountered during this sprint, which are explained in section 3.5.5.

**Porter**

The fourth sprint spanned from week 14 to 15, and its main goal was to finalize the development and add final touches to the product based on feedback from the usability tests. As mentioned in section 4.1.2, the sprint started with usability tests. Overall, the group was happy with the feedback, and were able to convert it into several useful *Github* issues. The usability tests are described in detail in section 3.5.3.

During the sprint, most of the improvements were related to the network graph. Most importantly, a correlation filter was implemented in the graph, which allows the user to decide which connections to display. Additionally, unit, snapshot and cypress tests were added. More information about testing is described in section 3.4.

As a result of the measure taken in the last sprint to arrange sprint planning meetings earlier, the sprint planning improved significantly for this sprint. Hence, the work done throughout the sprint was more spread out than before.

After the last retrospective, a shared document was created to store the notes in one place. As a result, better notes were taken during this sprint. However, there was still room for improvement, as not all of the group members remembered to add notes. Some expressed that long code sessions was a contributor, as it is easy forget to take notes while coding during long sessions. A possible solution is to divide coding sessions into smaller sessions, and to help remind each other to write notes whenever discussions occur. Nevertheless, as the prototype was mostly finished during this sprint, this solution could not be applied and tried for a future sprint.

After the sprint, the group moved every unfinished task to the backlog for future work. Furthermore, as the prototype was mostly finished, the main focus forward will be to finalize the product and hand it over to the customer.

**IPA**

For the final sprint, which began in week 16, the main objectives were to finish the report and hand over the product to the customer in a state where Trondheim Municipality can easily continue the work. By March, the customer wanted the group to hold a presentation with *The Norwegian Digitalisation Agency*. This required a lot of planning. The group decided who were going to work with the presentation, while the rest of the group worked on the report and finalized the product. More information about the presentation can be found in the following section.

Since there would not be another sprint, a regular retrospective meeting was not held at the end of this sprint. However, some final reflections on the overall development process were noted. The use of the agile process

method has been beneficial and helpful, as described in section 4.1.2. Despite having few initial requirements, the group has managed to develop a prototype which the customer found more than satisfactory. There have still been some difficulties throughout the process which partially inhibited the progress. Keeping a consistent scrum board on *Trello* while simultaneously updating *Github issues* has been a continuous problem that was never solved adequately. There was, as previously mentioned, also insufficient sprint planning in the first part of the project period, which was eventually resolved. In hindsight, the flexible guidelines for the agile approach may have contributed to these issues. This experience may be valuable in future projects.

### 4.5.3 Presentation with the Norwegian Digitalisation Agency

As the project was meant as a prototype to be used as an example and argument to fund further development, the customer wanted to present the prototype nationally. Consequently, the group was asked to hold a presentation for *The Norwegian Digitalisation Agency* at the end of the project,[16] which implies that the customer was satisfied with the product. The group considered this a fun and interesting challenge, and accepted the offer. Additionally, as this presentation was held a week before the final delivery of the project, the group thought the responses could be useful to discover possible limitations of the prototype before the final project delivery.

A full hour was assigned to the presentation. The directorate recommended that the actual presentation should be approximately 40 minutes to leave the remaining minutes for questions from the audience. The presentation was divided by the group into two sections. As the purpose of this presentation was to show the power of ontologies, the first section included an introduction, a description of ontologies and a comparison between ontologies and taxonomies. The second section was a live demonstration of the application. The demonstration was similar to the demonstration video supplied with the delivery of this report, but was described more in depth and presented practical examples of how the application can be used from the perspective of a fictional employee from Trondheim Municipality. After the demonstration, a URL to the deployed website was given to the audience to try out the application for themselves.

Around 100 people attended the presentation. In general the responses were very positive. The group was even asked to present the product for other agencies, including another municipality and *The Norwegian Directorate of eHealth*. Ironically, as the responses were exclusively positive, the presentation did not provide any limitations which could be presented in this report. Nevertheless, the goal to visualize the advantages of using ontologies to systematize sustainable development was reached.

[16] https://www.digdir.no/informasjonsforvaltning/faglig-arena-informasjonsforvaltning/2492

# Chapter 5

# Summary

This chapter provides a summary of the overall report and its essential findings. In accordance to the project's vision and goals (see section 1.2), the group has developed a prototype, in collaboration with Trondheim Municipality, to systematize sustainable development using the Web Ontology Language (OWL).

The objectives in this project introduced the group members to complex and unfamiliar topics and technologies. The customer required a prototype based on OWL. As a result, a preliminary research chapter was included, and was deemed necessary for the success of the project. Three main topics associated to sustainable development was researched and the acquired knowledge visualized the complexity of sustainable development. Furthermore, the baseline of the project, the representation method: ontologies based on OWL, initially unknown to the group members, were evaluated for the purpose of systematizing sustainable development. Other representation methods, such as taxonomies and graph databases were evaluated. Nevertheless, it became clear why ontologies were the customer's preferred representation method.

Further functional requirements were open for the group to decide, but the customer wanted a website containing a network graph. The visualization of, and interaction with, the network graph should be simple enough for non-technical users to understand. In addition to the functional requirements, there were also some non-functional requirements. It was important for the customer that the project was open source. Thus, the group decided to use a public Git repository for version control.

The architecture of the prototype was described by the 4 + 1 architecture model (see section 3.3).

Trondheim municipality has experience with *JavaScript*, therefore the group chose to use this language to facilitate further development. However, for easier debugging, *TypeScript* was selected. A library was needed to create a customizable force-directed graph, and the group concluded with *D3.js* as the best alternative. Even though the group was satisfied with the use *D3.js*, some difficulties were encountered when combining the library with *TypeScript*.

For the back-end, *Node.js* with *Express.js* was used. The group decided to use *Node.js* as it allows for having the same programming language in the front-end and back-end. Additionally, the ontology was stored in *GraphDB*, which is a *triplestore*. The choice made by the group was based on the project being a prototype, and the *triplestore* vendor should be reconsidered for future work (see section 3.6.2).

To visualize sustainable development, the group created an ontology page, consisting of two main components: a graph and a detail view. The detail view displays the definition of the currently selected node, and its connections in a interactive list, extracted from the ontology. The second component is a network graph, which visualizes the extracted information. Furthermore, one can interact with the graph by moving, dragging, locking, expanding and collapsing nodes. Combining these components allows for an intuitive and user-friendly experience.

Usability testing was essential to the success of the project, especially regarding the developed user interface. Changes were continuously made to enhance the user experience. Because of the limited amount of time, the group could not implement all of the wanted functionality, such as support for color blindness. The group has learned that some users requires extra support and/or custom functionality, and will bring this experience into future projects.

Different types of testing have been implemented, such as unit, integration and snapshot testing. These tests are a part of continuous integration in the *GitHub* repository, where every test needs to be approved before

branches can be merged. Some tests were more useful than others (see section 3.4), the group will take this experience with them.

The group decided to use *Airbnb's*, enforced by *ESLint*, as the groups preferred code standard. Manual code reviews and *CI* contributed to high code quality. Furthermore, the group decided that a branch should only fix one issue, though in some cases a branch fixed several. It was not a problem, however, it made it more difficult to track issues in the commit history. In the future the group will be even more strict regarding the use of branches, and follow the set standards.

*TypeScript* is strictly typed, which means the code is generally self-explanatory. As a result, documenting every line of code was deemed unnecessary by the group and the customer. Nevertheless, when using third party languages, especially *D3.js*, some parts of the code can be quite advanced. To ease the transition for potential future development, corresponding functions and methods have therefore been documented. Furthermore, if additional endpoints are included in the back-end, they should be documented in more detail.

To ensure a product of high quality, it was important for the group to plan the development process thoroughly. A project plan containing all the important dates was created to get an overview of the project period. The group also developed a release plan describing goals and duration for each sprint. However, the group was prepared for minor deviations from the release plan, to enable the agile process model that was chosen for the development.

In the early stages of the project, the group did an extensive risk analysis covering potential dangers associated with the project. To minimize risk, countermeasures were applied to reduce both the consequences and likelihood of each risk. This work paid off, as the group was able to avoid any major complications throughout the development process. Furthermore, the group believes that the countermeasures applied, in addition to the general awareness gained from completing such a thorough analysis, played an essential part in the process.

Group organization and collaboration were a key factor for the success of the project. The group only defined one role, the group leader, while remaining responsibilities were naturally distributed among the group members. This facilitated for a very fluid work flow, which was necessary to follow the agile development model. This structure worked well, as only a few minor conflicts arose, which were quickly solved in a collected fashion. The group organization also made work distribution easy. As a result, the work was well distributed among the group, and the total work load remained fairly even each week.

Collaboration with the customer turned out to be essential in the development process. To facilitate for this, structure and mutual expectations for the collaboration were developed right away. The customer's dedication to the project played an important role in the successful outcome. The collaboration with the supervisor provided general useful feedback in the further development of the report. Most of this communication was via text, and in hindsight, the group could have focused more on verbal communication.

In conclusion, the developed prototype exceeded both the group's and customer's expectations. The final product includes a significant amount of functionality, which would not have been possible without the extraordinary group collaboration and extensive work effort. Even if the prototype is not further developed, the group believes that it visualizes a new way of storing data for the municipality, especially in terms of sustainable development. This became apparent during the presentation held for *The Norwegian Digitalisation Agency* as the prototype sparked interest on a national scale. At last, the reader is encouraged to experience the prototype rather than only reading the report.[1]

[1] https://epic-ardinghelli-d1ee4d.netlify.app/

# Bibliography


1. United Nations Conference on Sustainable Development, Rio+20. https://sustainabledevelopment.un.org/rio20 (Fetched 2021-02-06)
2. Transforming our world: the 2030 Agenda for Sustainable Development — Department of Economic and Social Affairs. https://sdgs.un.org/2030agenda (Fetched 2021-05-02)
3. Scharlemann J, Brock R, Balfour N, and al. et. Towards understanding interactions between Sustainable Development Goals: the role of environment–human linkages. (English). Sustainability Science 2020; 15:1573–84. DOI: https://doi.org/10.1007/s11625-020-00799-6
4. Department of Economic and Social Affairs, Sustainable Development United Nations. https://sdgs.un.org/goals (Fetched 2021-02-02)
5. L.M. Fonseca JD and Dima A. Mapping the Sustainable Development Goals Relationships. (English). MDPI, 2020
6. Thomson SA, Pyle RL, Ahyong ST, Alonso-Zarazaga M, Ammirati J, Araya JF, Ascher JS, Audisio TL, Azevedo-Santos VM, Bailly N, and al. et. Taxonomy based on science is necessary for global conservation. PLOS Biology 2018 Mar; 16. DOI: 10.1371/journal.pbio.2005075
7. Taxonomy: Definition, Objectives and Characteristics. https://www.biologydiscussion.com/bacteria/taxonomy-definition-objectives-and-characteristics/49718 (Fetched 2021-05-04)
8. Li-Chun Zhang Johan Fosen BAH and Pekarskaya T. Taksonomi for klassifisering av indikatorer til bærekraftsmålene . (Norwegian). Statistisk sentralbyrå, 2021
9. Bingheng Chen Chuanjie Hong HK. Exposures and health outcomes from outdoor air pollutants in China. (English). Toxicology 2004; 198(1-3):291–300. DOI: https://doi.org/10.1016/j.tox.2004.02.005
10. Berners-Lee T, Hendler J, and Lassila O. The Semantic Web. Scientific American 2001; 284:34–43
11. W3. Resource Description Framework (RDF). https://www.w3.org/2001/sw/wiki/RDFS (Fetched 2021-05-02)
12. Allemang D and Hendler J. Semantic Web for the Working Ontologist: Effective Modeling in RDFS and OWL. 1st. Morgan Kaufmann, 2008
13. W3. RDF Vocabulary Description Language 1.0: RDF Schema (RDFS). https://www.w3.org/RDF/(Fetched 2021-05-02)
14. W3. OWL Web Ontology Language. https://www.w3.org/TR/2004/REC-owl-features-20040210/ (Fetched 2021-02-06)
15. Carmen Martinez-Cruz Ignacio J. Blanco MAV. Ontologies versus relational databases: are they sodifferent? A comparison. Artificial Intelligence Review 2011 Dec; 4:38. DOI: 10.1007/s10462-011-9251-9
16. David R, Aubert BA, Bernard JG, and Luczak-Roesch M. Critical Mass in Inter-Organizational Platforms. (English). AMCIS 2020. Available from: https://aisel.aisnet.org/amcis2020/adv_info_systems_research/adv_info_systems_research/21
17. Protégé. Object properties. https://go-protege-tutorial.readthedocs.io/en/latest/ObjectProperties.html (Fetched 2021-05-02)
18. Prud'hommeaux E and Seaborne A. SPARQL Query Language for RDF. W3C 2008
19. Neo4j. Experience the Power of the World's Leading Graph Database. https://neo4j.com/download-neo4j-now/ (Fetched 2021-05-02)
20. Property Graphs: Training Wheels on the way to Knowledge Graphs. https://www.semanticarts.com/property-graphs-training-wheels-on-the-way-to-knowledge-graphs/ (Fetched 2021-04-28)
21. Konys A. An Ontology-Based Knowledge Modelling for a Sustainability Assessment Domain. (English). Sustainability 2018; 10. DOI: 10.3390/su10020300

22. Rusher J. Triple Store. https://www.w3.org/2001/sw/Europe/events/20031113-storage/positions/rusher.html (Fetched 2021-05-02)

23. Enapso-GraphDB-Client - npm. https://www.npmjs.com/package/enapso-graphdb-client?activeTab=readme (Fetched 2021-05-06)

24. How to publish your first npm package. https://medium.com/@bretcameron/how-to-publish-your-first-npm-package-b224296fc57b (Fetched 2021-05-02)

25. Architecture & Components — GraphDB Free 9.7.0 documentation. https://graphdb.ontotext.com/documentation/free/architecture-components.html (Fetched 2021-05-05)

26. Programming with GraphDB — GraphDB Free 9.7.0 documentation. https://graphdb.ontotext.com/free/devhub/programming.html (Fetched 2021-05-05)

27. FAQ - Kotlin. https://kotlinlang.org/docs/faq.html#what-build-tools-support-kotlin (Fetched 2021-05-05)

28. SPARQL 1.1 Overview. https://www.w3.org/TR/sparql11-overview/ (Fetched 2021-05-04)

29. What is SPARQL? — Ontotext Fundamentals Series. https://www.ontotext.com/knowledgehub/fundamentals/what-is-sparql/ (Fetched 2021-05-04)

30. Facebook/React: A declarative, efficient, and flexible JavaScript library for building user interfaces. https://github.com/facebook/react (Fetched 2021-03-11)

31. Chakra UI - A simple, modular and accessible component library that gives you the building blocks you need to build your React applications. https://chakra-ui.com/ (Fetched 2021-03-25

32. Visx. https://airbnb.io/visx/ (Fetched 2021-05-01)

33. D3.js - Data-Driven Documents. https://d3js.org/ (Fetched 2021-03-11)

34. Npm Daily Downloads. https://observablehq.com/@mbostock/npm-daily-downloads?name=d3 (Fetched 2021-05-04)

35. Kruchten P. The 4+1 View Model of architecture. IEEE Software 1995; 12:42–50. DOI: 10.1109/52.469759

36. Madeyski L. Test-Driven Development: An Empirical Evaluation of Agile Practice. Springer, 2010

37. Factor M, Farchi E, and Ur S. Rigorous testing using SnapShot. IEEE Xplore 1997

38. Preece J, Rogers Y, and Sharp H. Interaction design, beyond human-computer interaction. Wiley, 2015 :386–97, 474–5

39. Ochs C, Geller J, and Perl Y. Summarizing the Structure of the Pizza Ontology: Ontology Development with Partial-area Taxonomies and the Ontology Abstraction Framework. https://saboc.njit.edu/files/pizzaOntologyTutorial.pdf (Fetched 2021-02-06). 2016

40. GraphDB Feature Comparison — GraphDB EE 9.7.0 documentation. https://graphdb.ontotext.com/documentation/enterprise/graphdb-feature-comparison.html (Fetched 2021-05-06)

41. Addlesee A. Comparing Linked Data Triplestores. https://medium.com/wallscope/comparing-linked-data-triplestores-ebfac8c3ad4f (Fetched 05/05/2021). 2018

42. DuCharme B. Learning SPARQL. 2nd ed. O'Reilly Media, 2013

43. JavaScript - MDN. https://developer.mozilla.org/en-US/docs/Web/JavaScript (Fetched 2021-05-06)

44. Bruk av farger — Tilsynet for universell utforming av ikt. https://www.uutilsynet.no/regelverk/bruk-av-farger/206 (Fetched 2021-05-06)

45. Sommerville I. Software Engineering. Pearson Education Limited, 2011 :30–33

46. Kniberg H. Scrum and XP from the trenches. C4Media, 2015 :63–67

47. Oversikt over nasjonale tiltak. https://www.regjeringen.no/no/tema/Koronasituasjonen/oversikt-over-nasjonale-tiltak/id2826828/ (Fetched 2021-02-06)

48. Wiki W. Ontology Editors. https://www.w3.org/wiki/Ontology_editors (Fetched 2021-05-04)

49. Kapoor B and Savita S. A Comparative Study Ontology Building Tools for Semantic Web Applications. International Journal of Web Semantic Technology 2010 Jul; 1. DOI: 10.5121/ijwest.2010.1301

50. Alatrish E. Comparison Some of Ontology Editors. (English). Management Information Systems 2013; 8:18–24

# Appendix

## A Test Coverage

| File | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s |
|---|---|---|---|---|---|
| All files | 39.85 | 38.36 | 33.48 | 40.33 | |
| src | 16.67 | 0 | 33.33 | 16.67 | |
| App.tsx | 100 | 100 | 100 | 100 | |
| index.tsx | 0 | 100 | 100 | 0 | 6-16 |
| reportWebVitals.ts | 0 | 0 | 0 | 0 | 3-10 |
| src/api | 23.29 | 0 | 21.43 | 23.29 | |
| api.ts | 25 | 0 | 20 | 25 | 8-21,35-51 |
| ontologies.ts | 22.81 | 0 | 22.22 | 22.81 | 5-12,21-24,29-34,41-44,49-56,61-70,75-80,85-92,97-102 |
| src/common | 62.15 | 66.99 | 47.37 | 63.4 | |
| colorSwitcher.ts | 100 | 100 | 100 | 100 | |
| d3.ts | 37.11 | 29.79 | 31.82 | 36.25 | 28-49,70-76,81,88-111,134-137 |
| node.ts | 98 | 97.37 | 100 | 100 | 33 |
| other.ts | 33.33 | 100 | 0 | 33.33 | 2-4,9 |
| regex.ts | 100 | 100 | 100 | 100 | |
| setBrowserPosition.ts | 0 | 100 | 0 | 0 | 2 |
| src/components/atoms | 70.41 | 46.51 | 67.65 | 71.91 | |
| Connections.tsx | 80 | 100 | 75 | 80 | 34 |
| ContextDivider.tsx | 100 | 100 | 100 | 100 | |
| ErrorModal.tsx | 78.57 | 42.86 | 60 | 83.33 | 45-46 |
| Footer.tsx | 100 | 100 | 100 | 100 | |
| Graph.tsx | 53.13 | 25 | 71.43 | 55.56 | 25-28,32-35,45-61 |
| GraphNodeKey.tsx | 100 | 100 | 100 | 100 | |
| GraphToolbar.tsx | 100 | 100 | 100 | 100 | |
| IconContainer.tsx | 66.67 | 100 | 50 | 66.67 | 21 |
| Navbar.tsx | 75 | 100 | 50 | 75 | 24 |
| SearchBar.tsx | 66.67 | 62.5 | 42.86 | 66.67 | 37-38,42-43,47,76-85 |
| SlideInDrawer.tsx | 100 | 100 | 100 | 100 | |
| SubGoalContainer.tsx | 100 | 50 | 100 | 100 | 31 |
| src/components/molecules | 68.24 | 47.5 | 65.38 | 68.35 | |
| AllConnections.tsx | 100 | 100 | 100 | 100 | |
| DetailView.tsx | 72.5 | 57.14 | 70 | 72.97 | 43-46,55-57,63,128-134 |
| GraphContainer.tsx | 55.56 | 0 | 33.33 | 62.5 | 13,17-18 |
| GraphDescriptions.tsx | 100 | 100 | 100 | 100 | |
| SubGoalsGrid.tsx | 76.92 | 50 | 80 | 72.73 | 15-16,30 |
| SustainabilityGoalsList.tsx | 52.63 | 25 | 50 | 52.63 | 19,27-32,40-44 |
| src/components/pages | 90 | 100 | 80 | 90 | |
| About.tsx | 100 | 100 | 100 | 100 | |
| Frontpage.tsx | 100 | 100 | 100 | 100 | |
| NotFound.tsx | 75 | 100 | 50 | 75 | 35 |
| OntologyPage.tsx | 100 | 100 | 100 | 100 | |
| src/d3 | 5.88 | 0 | 0 | 6.32 | |
| GraphSimulation.ts | 5.88 | 0 | 0 | 6.32 | 61-673 |
| src/hooks | 83.33 | 100 | 66.67 | 88.24 | |
| useDebounce.ts | 85.71 | 100 | 75 | 85.71 | 16 |
| useWindowsDimensions.ts | 81.82 | 100 | 60 | 90 | 16 |
| src/state | 100 | 100 | 100 | 100 | |
| store.ts | 100 | 100 | 100 | 100 | |
| src/state/reducers | 60 | 50 | 33.33 | 75 | |
| apiErrorReducer.ts | 60 | 50 | 33.33 | 75 | 13-18 |
| ontologyReducer.ts | 60 | 50 | 33.33 | 75 | 19-24 |
| src/tests | 0 | 0 | 0 | 0 | |
| setupTests.ts | 0 | 0 | 0 | 0 | |
| src/types | 0 | 0 | 0 | 0 | |
| genericTypes.ts | 0 | 0 | 0 | 0 | |
| ontologyTypes.ts | 0 | 0 | 0 | 0 | |
| src/types/d3 | 0 | 0 | 0 | 0 | |
| simulation.ts | 0 | 0 | 0 | 0 | |
| svg.ts | 0 | 0 | 0 | 0 | |
| src/types/react | 100 | 100 | 100 | 100 | |
| componentTypes.ts | 100 | 100 | 100 | 100 | |
| src/types/redux | 50 | 0 | 0 | 50 | |
| errorTypes.ts | 33.33 | 0 | 0 | 33.33 | 6-9 |
| ontologyTypes.ts | 100 | 100 | 100 | 100 | |

```
Test Suites: 23 passed, 23 total
Tests:       42 passed, 42 total
Snapshots:   25 passed, 25 total
Time:        16.195 s
Ran all test suites.
Done in 17.61s.
```

Figure 5.1: Test coverage for the front-end. The coverage provided by *Cypress* tests are not included.

# B Group Contract

## Group Contract for project TRDK03

This contract concerns all the members at the project TRDK03 in the subject IT2901. The undersigned commits to comply with the guidelines listed one through five. The agreement period is valid from February 1st until the project is completed. Every infringement will be discussed within the group and delivered to supervisor/faculty if needed.

**Group members:**

- Øyvind Jalland Schjerven
- Haakon Gunnarsli
- Per Solibakke
- Erik Skår
- Vemund Eggemoen
- Benedicte Myrvoll
- Andreas Winther Moen

***1. Project goals:***

*1.1.* Solve issues efficiently and get the best outcome.

*1.2.* Deliver a product of high quality.

*1.3.* Every group member is pleased and feel they have contributed to the finished product.

*1.4.* Achieve a good grade.

*1.5.* Satisfy the client.

**2. Presence and meetings.**

*2.1.* Attendance at every meeting and complete the mandatory assignments in the subject.

*2.2.* Two meetings every week, Tuesday 10.15 - 11.00 and Thursday 10.15 - 12.00. Meetings with the client is every Friday 11.00 - 12.00.

*2.3.* Every group member has the responsibility to contact other team members to plan group work, pair programming etc.

*2.4.* For every sprint/delivery, there will be a smaller meeting to reflect.

*2.5.* If problems can't be solved within the scheduled meetings, additional meetings can be scheduled.

Figure 5.2: Group Contract (Part 1) made for the project.

2.6. Every meeting has two secretaries. The secretaries will rotate.

2.7. Everyone has the right to have a break every other hour for fifteen minutes.

**3. Expectations to every group member**

3.1. Every group member will contribute to all parts of the project and meetings.

3.2. Expected workload is average 20 hours every week.

3.3. Every task that has been taken on should be completed within the given timeframe, if possible.

3.4. Orientate other group members if problems occur, and if needed ask for assistance.

**4. Disagreements and deviation.**

4.1. If disagreements tied to the project schedule occur, the client will decide.

4.2. Disagreements within the team will be discussed with every member present.

4.3. Notes will be taken if serious infringements occur, and if necessary discussed with the supervisor.

**5. Communication**

5.1. Communication will mainly take place during the meetings. The group also has a Slack for questions and discussions.

5.2. Joint work days will be scheduled if needed.

**Trondheim, 06.02.2021**

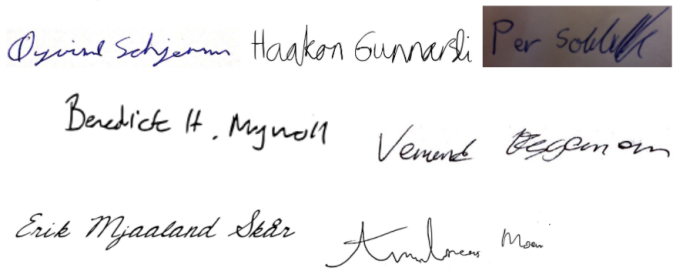

Figure 5.3: Group Contract (Part 2) made for the project.

## C Work-hours

| | | | | Total hours for different work categories | | | | |
|---|---|---|---|---|---|---|---|---|
| **Week** | **Total Hours** | **Average Hours** | **Meetings** | **Programming** | **Research** | **Lectures** | **Report** | ***Other*** |
| 3 | 24,0 | 3,4 | 21,0 | 0,0 | 3,0 | 0,0 | 0,0 | 0,0 |
| 4 | 117,0 | 16,7 | 26,0 | 3,0 | 65,5 | 18,5 | 2,0 | 2,0 |
| 5 | 132,5 | 18,9 | 27,0 | 18,0 | 12,5 | 3,0 | 71,0 | 1,0 |
| 6 | 103,3 | 14,8 | 38,8 | 28,0 | 21,5 | 0,0 | 0,0 | 15,0 |
| 7 | 125,0 | 17,9 | 35,0 | 49,0 | 12,0 | 0,0 | 3,0 | 26,0 |
| 8 | 121,5 | 17,4 | 25,5 | 60,5 | 5,0 | 1,0 | 8,5 | 21,0 |
| 9 | 93,0 | 13,3 | 26,0 | 51,0 | 2,0 | 0,0 | 4,5 | 9,5 |
| 10 | 124,0 | 17,7 | 34,0 | 23,0 | 0,0 | 0,0 | 67,0 | 0,0 |
| 11 | 141,0 | 20,1 | 29,0 | 98,5 | 2,5 | 0,0 | 5,5 | 5,5 |
| 12 | 122,5 | 17,5 | 23,0 | 97,0 | 0,0 | 0,0 | 2,0 | 0,5 |
| 13 | 50,0 | 7,1 | 1,0 | 44,5 | 0,0 | 0,0 | 0,0 | 4,5 |
| 14 | 202,5 | 28,9 | 46,0 | 137,5 | 0,5 | 0,0 | 3,0 | 15,5 |
| 15 | 104,0 | 14,9 | 17,5 | 68,0 | 0,0 | 0,0 | 0,5 | 18,0 |
| 16 | 97,5 | 13,9 | 27,5 | 33,5 | 0,0 | 0,0 | 8,0 | 28,5 |
| 17 | 190,5 | 27,2 | 28,0 | 4,0 | 0,0 | 0,0 | 141,0 | 17,5 |
| 18 | 217,5 | 31,1 | 3,5 | 0,5 | 0,0 | 0,0 | 213,0 | 0,5 |
| **Total** | 1965,8 | 18,4* | 408,8 | 716,0 | 124,5 | 22,5 | 529,0 | 165,0 |

*The first week is excluded in the calculation of average hours, as the project commenced in the middle of the week. The Easter week is also excluded.

Figure 5.4: The group's total and average work hours, and total hours for different work categories.

| Name | Total hours |
|---|---|
| Andreas | 322,5 |
| Benedicte | 259 |
| Haakon | 278,3 |
| Erik | 259,5 |
| Vemund | 275,5 |
| Per | 322,5 |
| Øyvind | 265 |

Figure 5.5: The total number of work hours for each group member.