NTNU

Norwegian University of Science and Technology

DEPARTMENT OF COMPUTER SCIENCE (IDI)

IT2901 - INFORMATICS PROJECT II

TRDK1_graph-NESTA

Date:
10.05.2023

Number of pages:
50 Pages

Members who have contributed to the delivery:

Student Number	Forename	Surname	Student mail
524299	Jesper	Barfod	jesperba@stud.ntnu.no
541880	Andreas Ringereide	Berg	andrerb@stud.ntnu.no
541896	Edvard	Bjørnevik	edvardbj@stud.ntnu.no
541891	Silviu Catalin	Mihai	silviucm@stud.ntnu.no
541871	Mats Thoresen	Nylander	matstn@stud.ntnu.no
476457	Magne	Slåtsveen	magne.slatsveen@ntnu.no
524299	Erland Lie	Amundgård	erlandla@stud.ntnu.no

Abstract

This project was a collaboration with Trondheim Municipality to develop a web-based prototype aimed at facilitating the cooperation between municipalities and systematizing recorded municipal data problems. By adopting an agile project development approach, close collaboration with the customer ensured the project met all critical requirements. The prototype utilizes *ontologies* to detect similarities between problems and adopts *Nestas' ODA method* to record them. The report elaborates on the significance of ontologies as a representation method for municipal problem systematization, as well as the limitations caused by the lack of documentation on this topic. The report concludes that the project was successful, and future work has been identified to implement a production version.

Table of contents

1. Introduction

1.1 Context & Motivation

Trondheim municipality and other neighboring municipalities struggle to connect with each other and collaborate on data-related problems they face. As municipalities have the same governing responsibilities in their respective areas, they often face the same tasks and problems. Consequently, it is possible that the problems that Trondheim municipality is currently facing have already been addressed by another municipality, or vice versa. By pooling resources and collaborating, these municipalities can work together to find solutions to shared problems. As part of this effort, 34 municipalities in Trøndelag County, including Trondheim, have launched a joint project called "Data-Driven Municipalities" (DigiTrøndelag, 2021).

Currently, the municipalities involved in the project are utilizing Google Sheets to monitor their data-related problems. However, the document has not been utilized much, leading to limited collaboration between municipalities. When inserting a new problem into the spreadsheet, they follow the *Offices of Data Analytics* (ODA) method, which is explained in section *2.1 Nesta & ODA*. While the client confirms that the ODA method has been effective in defining problems and their scope, it has been noted that storing them in a Google Sheet becomes inefficient when dealing with larger datasets.

1.2 Vision and Goals

1.2.1 Goal and Objectives

The goal of this project is to create a proof of concept for a web-based system to enhance collaboration between municipalities. The system's user interface should be user-friendly and capable of visualizing how data connects in a graph database. By taking advantage of the ODA method for inserting new problems, data will maintain a consistent format. This will enable users with similar cases, system types, and data types to find peers to connect with and discuss potential solutions. Despite its underlying complexity, the system should not require high technical competence to be used.

Furthermore, the system must support *inferencing*, allowing new facts to be inferred from the knowledge graph, even if they are not explicitly stated. This feature will be essential for connecting related problems based on their similarities.

The customer also emphasized the need for role-based access control for editing. While the customer has not specified explicitly how he envisions this to work, he has suggested that the system should support *staging*. This means that regular users should be able to stage a post for publication, and an administrator's approval is required for the post to be published and made visible to other users.

Trondheim municipality has a license for a graph-based database technology called *Ontotext GraphDB*[1] that they require us to use. The customer also requested us to use *ontologies* to display the structure of the subject

[1] https://www.ontotext.com/products/graphdb/

area and how different problems are related. When it came to other software requirements, we were mostly free to choose (see *3.1 Project Requirements* for more information about the requirements).

1.2.2 Project Scope

The scope of this project was to develop a functional prototype that proved the benefits of using an *ontology* for connecting similar problems that distinct users may have. An additional part of the project scope was to determine if the potential end product was feasible. Ontologies are described in detail in *Chapter 2* to help the reader understand the concepts and problems that occur when using ontologies.

1.2.3 Constraints & Limitations

The project faced a significant challenge in terms of a steep learning curve, particularly related to ontologies, which was unfamiliar to all group members. As a result, the group had to allocate time to grasp the concepts and create an ontology that met the project's requirements. This added to the overall time limitation, as the group struggled to determine the most effective route to a finalized ontology. Furthermore, creating ontologies is challenging, as one mistake can be detrimental, resulting in the need to start over.

1.3 Stakeholders

A stakeholder refers to an individual, group, or company/organization that is directly or indirectly involved in the project. Stakeholders may affect or get affected by the outcome of the project. One can divide stakeholders into two categories: internal and external stakeholders (Dingsøyr et al., 2010, p. 118).

Table 1.1 - Project Stakeholders

Who	Role	Internal/External
The group	Developers	Internal
Claudia Maria Cutrupi	Supervisor	Internal
NTNU	Course facilitator	Internal
Trondheim Municipality	Customer	External
DigiTrøndelag	End-user/secondary customer	External
Municipality employees	End-user	External

1.3.1 Internal stakeholders

The internal stakeholders for this project are the development group of seven students, the Norwegian University of Science and Technology (NTNU), and Claudia Maria Cutrupi as the project supervisor.

1.3.2 External stakeholder

External stakeholders in this project are Trondheim municipality as the customer, DigiTrøndelag as the end-user/secondary customer, and employees of the municipalities in Trøndelag County as the end users of the developed product.

2. Preliminary Research

A comprehensive research phase was necessary to gain a better understanding of the project, the customer's requirements, and the necessary technologies needed for developing a prototype that would systematize data-related problems, hereinafter referred to as "data problems", faced by municipalities. The preliminary research phase lasted almost a full month, starting on January 23rd with our initial customer meeting, and concluding on February 21st, just prior to the onset of the first sprint.

In Chapter 2, we offer a comprehensive overview of the preliminary research phase, which starts with an introduction to Nesta and the ODA method. We further delve into classification methods, specifically taxonomies, and ontologies, and examine the technologies utilized to create ontologies. Then we discuss the advantages and disadvantages of ontologies, emphasizing their ability to merge and systematize complex concepts in a way that conventional IT classification methods cannot. Lastly, we present our conclusion that ontologies could be a fitting solution for the project task.

2.1 Nesta & ODA

Nesta (National Endowment for Science, Technology and the Arts) is a governmental foundation in the UK that works with creating technical solutions and modernizing governmental bodies. One such thing Nesta has had great success with is the ODA (Offices of Data Analytics) method.

The ODA method is a procedure for recording and formatting data problems governmental bodies or organizations face, that can be solved using data analytics. By 'data analytics' we refer to the discovery, interpretation, and communication of meaningful patterns in data (Copeland et al, n.d, p. 4). An example of such a problem may be: "*Given the recurring fire incidents in our municipality, we are interested in creating a visualization that highlights the districts where such incidents commonly occu*r" The ODA method ensures that such problems follow a concise and standardized format, and emphasizes the importance of achieving actionable insights from the data.

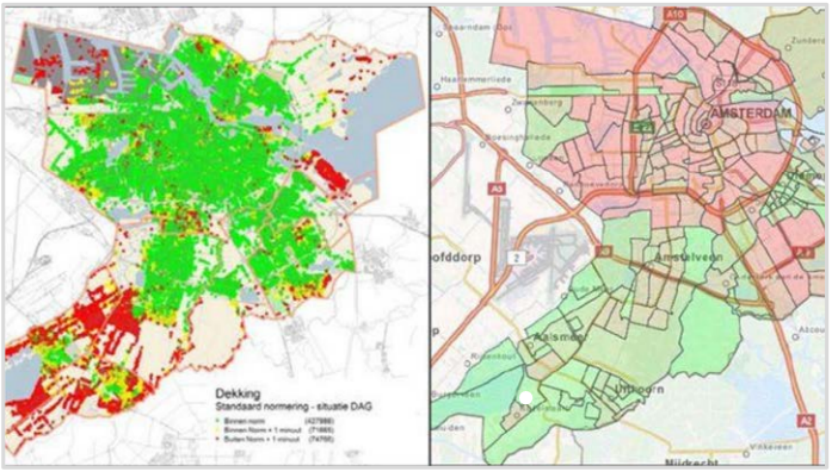


Figure 2.1 - A real-world example of data analytics being used to tackle municipal data problems. The Amsterdam fire brigade collected and combined data from multiple sources with past incident records to create maps visualizing fire occurrences in terms of location, timing, and frequency. The ODA method would have been a suitable fit for this type of problem. (Copeland et al., n.d, p. 36)

Enforcing a standardized format for recording data problems makes it easier for employees to understand and identify similar problems being addressed by other parts of the organization. Moreover, it also reduces the chances of misunderstandings or duplicate entries. As the ODA method requires employees to describe the data problem at hand concisely, the method also makes data problems easier to be understood and implemented by IT professionals. Most importantly for this project, the standardized format is suitable for collaboration across organizations, i.e. across municipalities in our case. Nesta defines the ODA method as: *"A model for multiple organizations to join up, analyze and act upon data sourced from multiple public sector bodies to improve services and make better decisions."* (Eaton & Bertoncin, n.d.)).

The ODA method breaks down a problem into four fundamental components: *Specific problem*, *Clear data product*, *Defined action*, and *Accessible data* (Copeland et al., n.d., p.25).

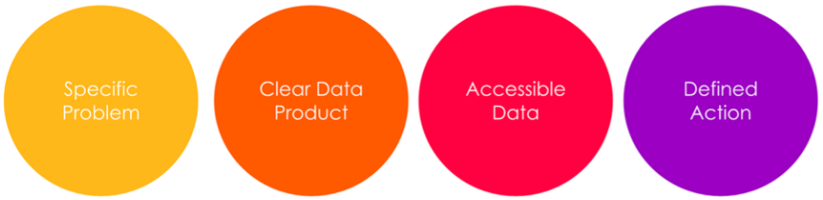


Figure 2.2 - The four components of an ODA problem
(Copeland et al., n.d., p35)

The *"Specific problem"* is the most important component of an ODA problem, as it outlines what problem the municipality is facing. When writing down the specific problem one should avoid large macro-level problems and instead focus on narrow and actionable challenges. For instance: "*Many teenagers in our municipality are struggling with mental illnesses*" is too broad, and one should instead decouple this overarching problem into several smaller and actionable specific problems. One such problem could be: "*We don't know which risk factors are associated with mental illness in our city.*" In short, a specific problem should be a well-defined and clearly articulated problem that can be solved using data analytics. *Specific*

problems can be further divided into five subcategories, which provide a more detailed specification of the type of problem we are dealing with (see Figure 2.3).

5 Specific Problem Types*
Targets are difficult to identify within a broader population
Services do not categorise high-priority cases early
Resources are overly focused on reactive services
Repeated decisions made without access to all relevant information
Assets are scheduled or deployed without input of latest service data

Figure 2.3 - The five subcategories of a 'specific problem'
(Copeland et al., n.d., p. 27)

Once the problem has been identified, the next step is to create a "*Clear data product*" that will be used to address the problem. This data product should be designed with the end user in mind and should provide relevant, accurate, and actionable insights. In layman's terms, the data product is what an employee would need to see displayed on a computer screen in order to find solutions to the data problem. The format for displaying the data should be specified if necessary, such as whether it should be displayed as a heatmap, a prioritized list, a dashboard, etc. Continuing with the previous example, a clear data product could be "*A visualization of the demographics of teenagers affected by mental illnesses in our city*"

"*Accessible data*" refers to the data needed to generate the desired data product. This data can be sourced from various channels, including but not limited to open data, public sector, business & third sector, and citizen data (Copeland et al., n.d., p. 41).

The last component, "*Defined action*" describes what the organization would do differently if it had all information needed about the specific problem and the desired data product. In our example, it could be "*With the desired information being systemized and visualized we could more easily implement policies to specifically target teenagers that are prone to developing mental illnesses*".

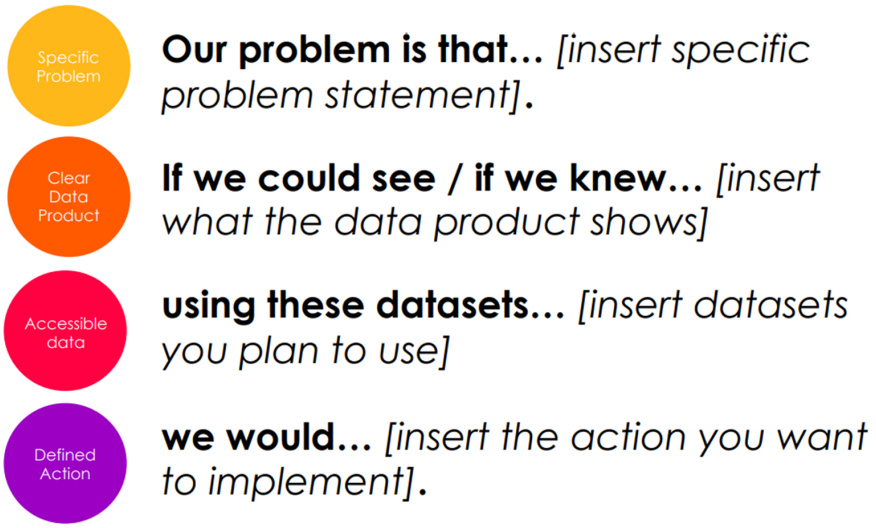


Figure 2.4 - A description of how each of the four components of ODA problems can be formulated, as presented in Nesta's guide to Public Sector Data Analytics (Copeland et al., n.d., p. 45)

As the ODA method is an integral part of this project, doing thorough research was important in order for us to fully understand what the method entailed. By fully understanding the ODA method we would gain a clearer understanding of how the ontology should be constructed, and what inferences would be important. Moreover, the customer emphasized the importance of the prototype being easy to use, and this would be easier to achieve if we fully understood the ODA method ourselves and could simplify the understanding for newcomers.

Trondheim municipality conducted a trial experiment by utilizing the ODA method to track some of their municipal data problems, with Google Sheets serving as the storage platform. Upon examining the spreadsheet, we observed that numerous employees possessed an erroneous understanding of the ODA method. A common issue we identified was the tendency of employees to mistake the ODA method for a universal solution for all types of problems when in reality, it is designed specifically for those that can be addressed through data analytics. Consequently, some employees entered problems that were not applicable to the ODA method. Furthermore, many employees lacked a basic understanding of *Clear data product* due to their limited knowledge of statistics or data visualization, leading to difficulty distinguishing between various data representations, such as heatmaps and scatter plots. These insights were taken into consideration in the development of a more user-friendly interface.

2.2 Methods of classification

The ODA problems tracked by municipalities have to be stored and systematized. In the earlier described trial experiment, digital spreadsheets were used for storage, but this method was found to be inconvenient for several reasons. The user interface was cluttered and difficult to use, updating the spreadsheet required a significant amount of time, and Google Sheets offers very limited programming logic, making it incapable of

supporting inferencing. Moreover, the spreadsheet lacks proper classification of problem entries, which can make it time-consuming to locate specific entries as the dataset grows in size.

In this subchapter, we will discuss methods of classifying information, which will be crucial for this project in order to identify potential similar data problems, for avoiding duplicate entries, and to establish an organized storage system. The customer has specified that the web-based solution we are developing should incorporate an ontology to classify the information and that Ontotext GraphDB should be used as the database. The customer is a strong advocate for ontologies, and we will briefly explain what they are, and why they are valuable for addressing tasks such as the one our customer is facing.

2.2.1 Taxonomies

A classical method of classification is organizing the domain into hierarchies, forming a *taxonomy*. Taxonomies were originally created for classifying species in biology. For instance, the 'linnean taxonomy' is used to classify living organisms (see Figure 2.5). In computer science, taxonomies essentially mean organizing a subject matter into subclasses and superclasses, where the root superclass incorporates all other subclasses. The relations between the superclasses and the subclasses can be seen as an "is-a" relationship, where the subclasses are also members of the superclass (Noy & McGuinness, 2001). For example, the subclass "sapiens" (modern human) is a type of "homo" (human). A taxonomy thereby creates a tree-like structure, with the outermost "leaves" representing the most specific level, whereas the upper 'branches' get more general. Taxonomies consequently provide a way of looking at a domain of knowledge at different levels of granularity. However, a major limitation of taxonomies is that they have to be hierarchical in structure, and a singular class can only have relations with its immediate superclass and subclass. In taxonomies, it is therefore impossible to define relationships between classes that belong to the same level in the hierarchy or to define relations between classes that are situated in different parts of the hierarchy (Figure 2.6).

To display potential similar entities among ODA problems, our project would likely necessitate relationships between entities at the same level in the hierarchy. Consequently, a regular taxonomy would be an unsuitable choice for this project. Additionally, taxonomies have limited reasoning capabilities compared to other methods of classification, which the customer has specified is important for this project.

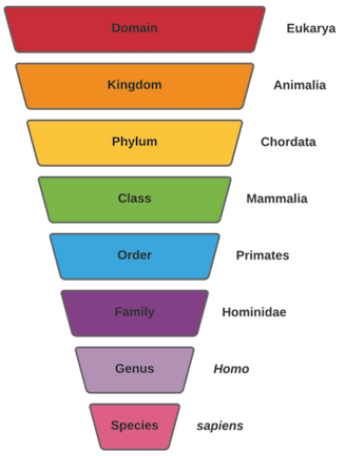


Figure 2.5 - The linnean taxonomy, showing the superclasses of human beings (sapiens).

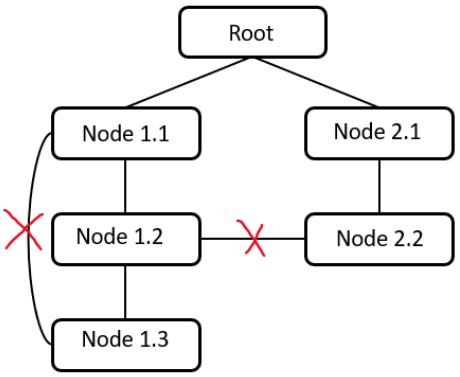


Figure 2.6 - A diagram showing some illegal relations in a taxonomy.

2.2.2 Ontologies

An alternative approach to classification is the use of ontologies, which are similar to taxonomies in that they are also formal representations of a domain, containing classes of varying hierarchical levels and specifying the relations within the domain. However, there are several significant differences between taxonomies and ontologies. Firstly, ontologies may contain "*individuals*" within classes. Individuals are specific instances of a certain class. For example, an individual of the class "Human" could be "Linus Torvalds", or "Grace Hopper" as they are specific instances of humans. The classes in ontology are otherwise similar to classes in taxonomies: they may have sub- or superclasses, and there always exists an uppermost root class.

Another major difference is that ontologies have a more expressive way of delineating relations. Relations (called "*object properties*" in ontologies) are **not** between classes but between individuals. Relations in ontologies are too complex to be explained in detail here, but some key characteristics are that they enable relations between instances of different classes no matter their hierarchical level, they allow for "*restrictions*", and their features can be further specified using concepts from discrete mathematics. By "restrictions" we refer to the fact that a relation can specify which classes an individual must belong to in order to have a certain relation, and which classes the receiving end of the relation must belong to. A restriction can also specify what relations an individual of a class must have in order to be considered a part of that class. For example, we could specify that an individual must have the relation "hasChild" to another individual, and be part of the "male" class, in order to be considered a part of a "father" class. A More detailed overview can be found in the OWL 2 primer (*OWL 2 Web Ontology Language Primer (Second Edition)*, 2012)

Moving on, relations can further be modified as being symmetric, transitive, reflexive, inverse, and so on. For example, a "hasChild" relation would be inverse to a "hasParent" relation, meaning that if an individual A has a relation "hasChild" to another individual B, then B has the "hasParent" relation to A (see Figure 2.7). It

is important to note that once the “hasChild” relation from individual A to individual B has been created, then the inverse “hasParent” relation will automatically be created (or rather, it will be “inferred”).

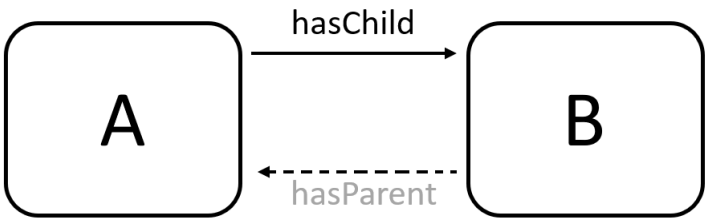


Figure 2.7 - An example of two relations that are inverse. The “hasParent” can be inferred from its inverse relation “hasChild” and will therefore be automatically created.

Learning ontologies; the "Pizza ontology exercise”

In the early stages of the research phase the group was instructed by the customer to create and present a “pizza ontology”. The pizza ontology exercise is a well-known introductory task in the realm of ontologies and helps in understanding how ontologies and ontology software work. The general idea is to create a simple ontology consisting of different pizzas, bases, and toppings. The task necessitates the use of restrictions, and constructing a well-thought-out class hierarchy. As an example, it is essential to specify that vegetarian pizzas do not contain meat toppings. The task emphasizes the hidden complexities of ontologies, and as the number of relations and classes grows, so does the complexity. Learning how to use ontologies is a complex task with a steep learning curve - at NTNU there is a master’s course[2] worth 10 study points in Health informatics, where learning ontologies is one of the major goals.

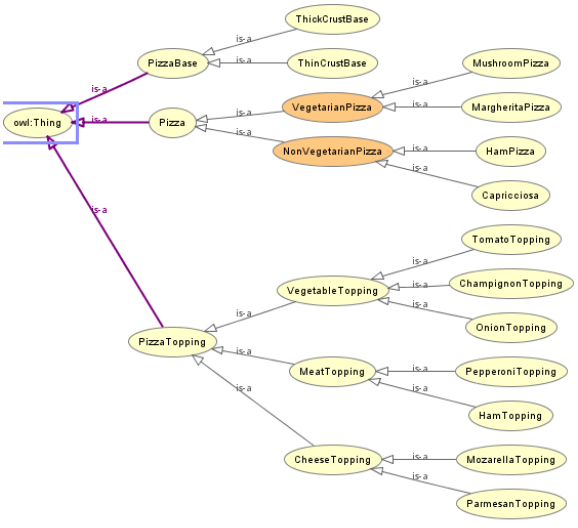


Figure 2.8 - A graphical overview of the pizza ontology

[2] https://www.ntnu.no/videre/gen/-/courses/nv22551

Although the pizza ontology did not contribute to the project directly, its creation enhanced the group's understanding of how ontology software functions. This led to an increase in the group's confidence and awareness of the various possibilities offered by ontologies. Additionally, it highlighted the significance of a methodical and well-planned development, as even minor mistakes can be detrimental to the ontology. As such, if one finds that the ontology one has created is not functioning as intended, it may be advantageous to begin anew since attempting to modify a highly intricate and nested ontology often leads to the emergence of additional errors. These valuable insights were taken into consideration when we started developing the actual ontology for the project.

2.3 Technologies used for creating ontologies

2.3.1 Protégé

The group used Protégé, an open-source ontology editor, as the primary tool to construct the product ontology. Protégé provides a range of features, including tools for ontology construction and visualization, as well as semantic reasoners capable of performing inference on the ontology. There exist various languages for creating ontologies, including well-known options like RDF, RDFS, and OWL (Ehimwenma et al., 2020).

However, manually creating these files can be challenging. Fortunately, Protégé offers a graphical interface for creating ontologies, which can easily convert to these formats. Additionally, Protégé allows users to take advantage of various plugins and extensions for adding extra functionality and to integrate with other tools and systems.

In order to gain familiarity with protégé, the group chose a trial-and-error approach. This approach allowed the group to learn about the tool's inner workings as they progressed with the ontology development, ultimately improving their productivity.

2.3.2 SPARQL

The query language utilized for our *GraphDB Ontotext* database is *SPARQL*, which differs from SQL and other query languages since it uses URIs to identify data and follows a *subject-predicate-object structure* (semantic triples). Its technical details aside, the most significant advantage of using SPARQL over SQL is that it is not restricted to querying individual or combined tables. Instead, SPARQL can perform queries over the entire web of linked data in the database, making it unnecessary to include FROM statements.

2.4 The Pros and Cons of Ontologies

The main advantage of using ontologies is their compatibility with semantic reasoners. These are software tools that can infer logical statements from a set of asserted facts. We briefly touched upon inference in the

last paragraph of *2.2.2 Ontologies*, but we want to expand upon it here to give the reader a thorough understanding of its possibilities.

A problem in the IT industry is that data can often be inconsistent, out of sync, and disconnected. Oftentimes when data changes, someone must manually update associated references to correctly represent the new data. For instance, Norway has governmental websites for all municipalities, and suppose that the government decides that all these websites should display their neighboring municipalities. In such a case, government-employed IT professionals would likely have to scour through all these websites and manually add the neighboring municipalities. While there are probably easier ways of doing this, the fact remains that it is a hassle and a time-consuming task. Even if for instance "Trondheim" has added "Melbu" as its neighboring municipality, one still would have to add the same relation the other way around. While the fact that this neighboring-municipality relation is symmetric is self-evident for us humans, computers struggle to understand unless that fact is listed explicitly. **"Semantic data"** however, solves this problem, as it would allow the relation to be explicitly noted as being symmetric. With semantic data, the interconnectivity automatically updates the references (Allemang & Hendler, 2009). When semantic data is written, its meaning or context associated with it is written as well. Semantic data is tightly connected to inference as inferencing is what makes the data automatically update. When the semantic data "A --(symmetric)--> B" is created, then "B → A" will automatically be inferred. This project will rely on inference to accomplish various tasks, such as identifying comparable ODA problems and identifying redundant entries. Moreover, while clever inferences might be difficult to construct for the software developer, the inference works "under the hood", hiding complexity and making seamless connections for the user.

One drawback with ontologies, however, is that they rely on human knowledge, which can be inconsistent, incomplete, or contradictory. Ontologies are by nature, subjective and context-dependent. Even two ontologies that deal with the same domain, are likely to differ significantly in terms of which concepts they consider relevant and how they conceptualize them. The fact that ontologies are subjective, is in and of itself not noteworthy, as all classifications of this nature suffer from this issue. However the real issue comes from the fact that ontologies are more complex to develop than other methods of classification, and they are therefore more prone to errors caused by a wrong conceptualization. Developing ontologies requires a high level of expertise, not only in the field of ontology but also in the domain which the ontology covers. This often necessitates collaboration between multiple partners which can be time-consuming, challenging, and costly. If an ontology is wrongly constructed, for example, if a relation is missing an attribute, it may lead to "domino effect errors" creating an ontology that does not accurately represent the reality it is trying to depict.

Furthermore, Ontologies have not been utilized much and as such they are susceptible to standardization and inter-compatibility problems. While there exist several ontology languages, tools, and frameworks, many of these are deprecated or have very limited compatibility with other tools. Consequently, developing large and complex systems utilizing ontologies can prove challenging. Additionally, learning about ontologies beyond the basics can be a difficult task due to scarce documentation and limited resources. The group experienced this problem firsthand during the development process.

Although constructing ontologies is a challenging and time-consuming task, once up and running, the payoff is that the data being stored will be more valuable (because it is semantic and therefore context-sensitive) and

that the data may be easier to maintain and update. An ontology takes longer to develop before it can provide useful insights, but once the initial hard work is done, the benefits are substantial. Other alternative database types, such as SQL databases or regular graph databases, do not support inference to the same extent as ontologies. In conclusion, while ontologies are complex, inference is an important concept for hiding complexity and doing implicit reasoning, which makes ontologies a good fit for this project.

3. Product

3.1 Project requirements

The approach used to establish the requirements for the project was unconventional in nature, as the group was granted significant leeway to shape the requirements, rather than solely relying on the customer's predefined specifications. The only exceptions from this were the requirements to use *GraphDB Ontotext*, to utilize ontologies for the project, and to have the code GPL3 licensed[3]. These three requirements were non-negotiable and had to be adhered to.

3.1.1 Functional requirements

The functional requirements were extracted from the project description given by the customer, internal discussions within the group, and meetings with the customer to further discuss the potential requirements. The functional requirements are listed below, and to make consistent wording, all requirements start with the following statement: "The system should…".

Table 3.1 - Functional Requirements

ID	Functional Requirements	Priority	User Story ID
01	**Be a web-based solution.** The application must run as an HTML web page, designed for desktop use. This is a requirement provided by the customer and thus is of high priority.	High	**1**
02	**Be usable across municipalities.** Though Trondheim municipality is the customer, the application must be usable by employees in other municipalities. This allows for collaboration in problem-solving between several municipalities. This is a requirement provided by the customer and thus is of high priority.	High	**2**
03	**Use a GraphDB Ontotext database for storing data.**	High	**n/a**

[3] https://www.gnu.org/licenses/gpl-3.0.en.html

	GraphDB is a graph-based database that supports inferencing, and will thus be used to link related problems between municipalities. This is a requirement provided by the customer and thus is of high priority.		
04	**Be Ontology-based.** In order for the system to deduce that a problem is similar to another problem, a custom ontology should be created with clearly defined rules based on NESTAs ODA-guide. This is a requirement provided by the customer and thus is of high priority.	High	**n/a**
05	**Support inferencing.** By letting the system support inferencing, we would automate the process of linking related problems, saving time and effort for the ones administrating the website. The requirement is of high priority, as it is directly linked to requirements 03 and 04.	High	**n/a**
06	**Require problems to be formulated using the ODA method** In order for the inferencing to work properly, the database requires that users form problems that satisfy the requirements derived from the structure of the ODA method. This requirement is of high priority, as it is directly linked to requirement 05.	High	**3**
07	**Permit users to add data to a database** Users must have the ability to send a created problem to the database, so it can be examined by an admin. This requirement is of high priority, as it describes the core functionality of the application.	High	**4**
08	**Permit users to edit a published post.** Users must have the ability to edit the content of a problem posted by themselves. This requirement is of high priority, as it describes the core functionality of the application.	High	**5**
09	**Permit users to delete a published post.** Users must have the ability to delete a problem posted by themselves This requirement is of high priority, as it describes the core functionality of the application.	High	**6**

10	**Support searching and filtering for other users' posts.** A user must have the ability to discover problems published by other users, with the help of search and filtering through the UI. This requirement is of high priority, as it describes the core functionality of the application.	High	**7**
11	**Support role-based access control for editing.** It must be possible for an admin to edit all problems posted by users belonging to the same organization. This requirement is a medium priority, as it would provide significant improvements to the system's usability.	Medium	**8**
12	**Display status information for a specific ODA-problem.** Users need to know how their published problems are progressing, so it is important that they can view the status of each issue. This feature helps users quickly gauge how far the municipality has come in finding/creating a solution. There are four possible statuses: • Under evaluation (a problem that has yet to be approved by an administrator) • New problem (a problem that has been identified, but not yet addressed) • In progress (a problem that is being worked on) • Solved (a problem that has been solved by at least one municipality) This requirement is of medium priority, as it would provide significant improvements to the system's usability.	Medium	**9**
13	**Display contact information for a specific problem.** The user must have the ability to contact the person or organization who published a specific problem, as well as others that have the same problem. By accessing a problem in the UI, the user will see the publisher's and subscribers' email addresses and phone numbers. This requirement is of medium priority, as it would provide significant improvements to the system's usability.	Medium	**10**
14	**Support "subscribing" to a problem.** A user must be able to subscribe to a problem so that they can more intuitively monitor the status of the problem-solving process. This requirement is of medium priority, as it would provide significant improvements to the system's usability.	Medium	**11**

3.1.2 Non-functional requirements

Some of the non-functional requirements were also extracted from the project description, as with the functional requirements. The non-functional requirements can be seen below, and also start with the statement “The system should…”.

Table 3.2 - Non-Functional Requirements

ID	Non-functional Requirements	Priority	User story ID
01	**Have a consistent UI for easy, non-complicated navigation** Related quality attribute: ***Usability*** The different pages of the UI must correspond with each other in regards to color schemes and component styling. In addition, it must utilize icons that are familiar to the average user. This is a high-priority requirement, as many employees in the municipalities do not have much technical background. The application and its features should be easily accessible and comprehensible, allowing users to make full use of them within minutes of their initial launch.	High	**12**
02	**Handle multiple users using the system simultaneously** Related quality attribute: ***Scalability, Performance*** Multiple users must be able to use the application simultaneously, without causing errors or failures. This is a medium-priority requirement, as simultaneous use will be important for this application since it will be utilized by numerous users across various locations. However, this is not the focus of this project.	Medium	**16**
03	**Fully load the system within a maximum of 2 seconds with the assumption that the user is situated in Norway** Related quality attribute: ***Performance*** The system should be able to fully load and display all externally stored data within 2 seconds of accessing the web page, as long as the user is situated in Norway.	Medium	**17**

	This requirement is of medium priority, as it would ensure an effective initialization of the user experience, but it is not an absolute requirement.		
04	**Be compatible with modern web browsers (Edge, Safari, Firefox, Chrome)** Related quality attribute: ***Portability*** The application should work equally on these widely used web browsers; Microsoft Edge, Safari, Mozilla Firefox, and Google Chrome This requirement is of medium importance as users tend to utilize various web browsers. Requiring users to switch to a particular browser to access the website is not user-friendly.	Medium	**15**
05	**Be secure against unauthorized access** Related quality attribute: ***Security*** The application should implement some form of security measures with regard to authentication. However, the customer does not require us to finish this for the proof of concept, which is why it is of low priority.	Low	**14**
06	**Meet the Web Content Accessibility Guidelines, WCAG 2.1.** Related quality attribute: ***Accessibility*** The application should satisfy the accessibility requirements defined in the WCAG 2.1 standard[4]. This is to ensure that most users are able to use the application, despite disabilities such as visual impairment. The public sector must comply with the WCAG standards (Uutilsynet, n.d.). This requirement is of a low priority since it is not the focus of this project as this is a prototype. In a production version, however, this would be of high importance.	Low	**13**

3.2 Technology stack

3.2.1 Frontend

The group opted to use *React*[5] for the frontend part of the project. React is an immensely popular JavaScript library in the industry and has proven to be an efficient tool for creating user interfaces. With strong

[4] https://www.w3.org/TR/WCAG21/
[5] https://react.dev/

community support and resources, we assessed the library as one where finding help or resources to solve problems or learn new features was a guarantee.

Due to its popularity, React offers interoperability with many other libraries and technologies, making it highly customizable (i.e. we can fine-tune the frontend experience to our project needs). Furthermore, all group members had experience with React from previous projects. As this project required the group to learn many new IT concepts and technologies, we felt it would be sensible to also work with something we were all familiar with.

Going with React had the additional benefit of being a technology that was already in use by Trondheim municipality, meaning that they could easily take project lead and develop the system further once we finished the prototype.

Lastly, we decided to use Typescript[6] alongside React as our frontend programming language. Our experience has shown that static typing can significantly enhance the debugging and testing process.

3.2.2 Backend

Initially, the plan for the project's backend involved utilizing the RDF4J Java framework (Eclipse, n.d.) to extract information from Ontotext GraphDB. However, as the development process progressed, it became clear that one could use TypeScript together with SPARQL to connect the backend with the database through the RDF4J REST API (*RD4J REST API - OpenAPI Spec*, n.d.). This approach was advantageous as it would require little boilerplate code and allow us to create highly customizable queries to suit our needs. The node library graphdb.js was also considered, but its functionality was too general to suit the needs of the project (*JSDoc: Home*, n.d.). A consideration was also the fact that having fewer dependencies in the project would make the codebase easier to update in the future.

Connecting the backend to the frontend was done through the Express framework. (*Express - Node.Js Web Application Framework*, n.d.). The framework is widely used and is a streamlined way of creating an API for our frontend, making the choice to use a simple one.

The approach of using typescript in the backend gave the group the advantage of utilizing the same programming language in the frontend and backend, making it easier for the members to work on different parts of the project.

3.2.3 Ontotext GraphDB

Ontotext GraphDB, which we will refer to as "GraphDB", is a graph database that is compliant with RDF and SPARQL. It is one of the most widely used RDF stores (i.e. purpose-built database for the storage and retrieval of triples through semantic queries.) and graph database management systems (DBMS) (solid IT, n.d.). It is considered a robust and efficient DBMS. The main reason for choosing GraphDB was the customer's specific requirement to use this DBMS for the project.

For local development, the database was set up as a *Docker container.* A container is software that packages another software into standard units (Docker, 2021). This means the database and all its dependencies are

[6] https://www.typescriptlang.org/

packaged up and put in the container, which allows the database to run standalone, quickly, and reliably, in any computing environment. To allow an easy setup of the container for GraphDB on any computer, we used Docker-compose[7]. This also allowed us to automatically create a repository and upload our ontology to it, by creating a configuration file that worked across the board.

3.3. Software architecture

The field of software architecture lacks a clear consensus on what constitutes “architectural design” versus “application design” (Solms, 2012). There are therefore many different ways to show the architecture of the software. The group chose the *4+1 View Model* to explain and show the software architecture of the project.

3.3.1 The 4+1 View Model

The 4+1 View Model is an architectural framework that describes the software architecture using five concurrent views (Kruchten, 1995). These views are visual representations that document all features of the system during development and serve as instruments to identify any deficient or flawed functionalities. The reason for using multiple views is to cater to the different perspectives of stakeholders involved in a software project. The four primary views are the *Logical view*, *Process view*, *Development view*, and *Physical view*, with the fifth view being *Scenarios*: the most crucial use cases of the system.

Logical View

The logical view focuses on the system's functional requirements, primarily for end-users, and is usually represented using class diagrams that show the relationships between classes (Kruchten, 1995). Figure 3.1 shows the various pages from the frontend, the routes in order to reach them (black arrow), and procedures using the global state (green arrow). The diagram omits the use of arrows that link pages together as that would quickly make the diagram unreadable. Two examples of this are that (1) most of the pages can be reached from *Home.tsx*, and (2) every page that requires a logged-in user automatically redirects the user to *Login.tsx* if they are not logged in.

[7] https://docs.docker.com/compose/

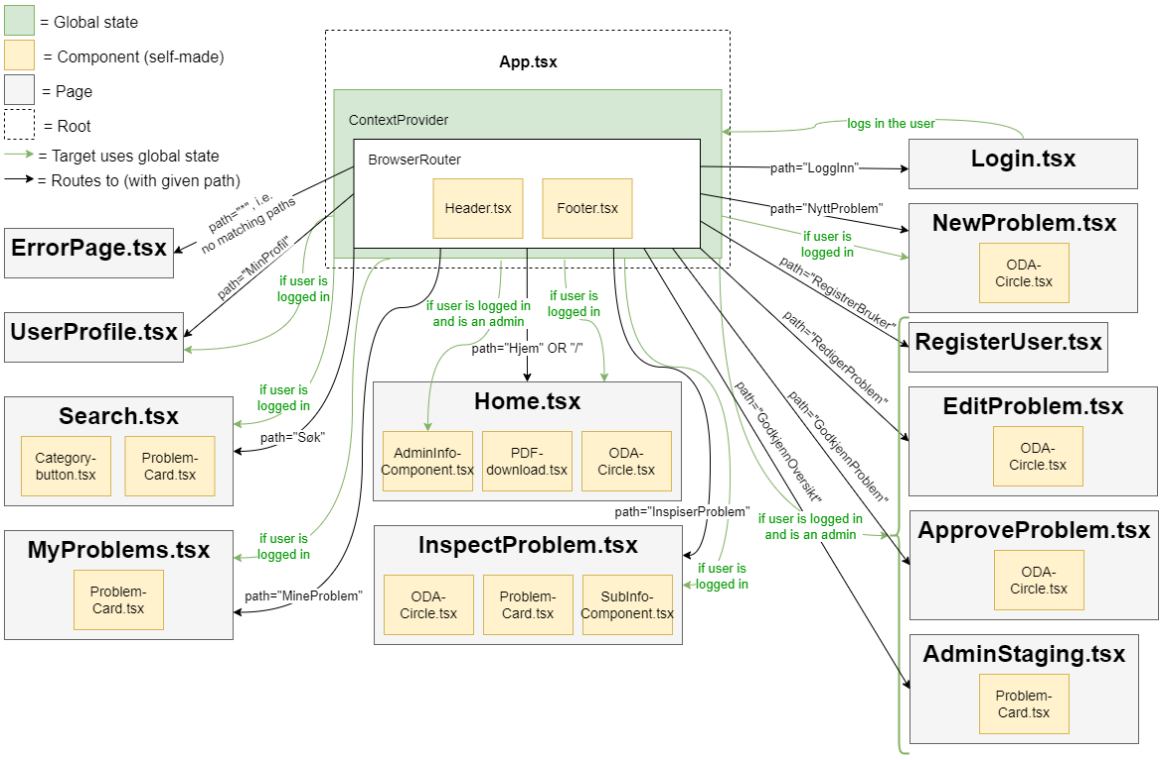


Figure 3.1 - Logical view of the pages in the web application

Process View

The process view concerns the system's runtime behavior, which includes the system's processes and how those systems communicate (Kruchten, 1995). Figure 3.2 is an activity diagram that visualizes the workflow of creating a problem on the website and the website showing related problems retrieved via inferencing both during and after problem creation.

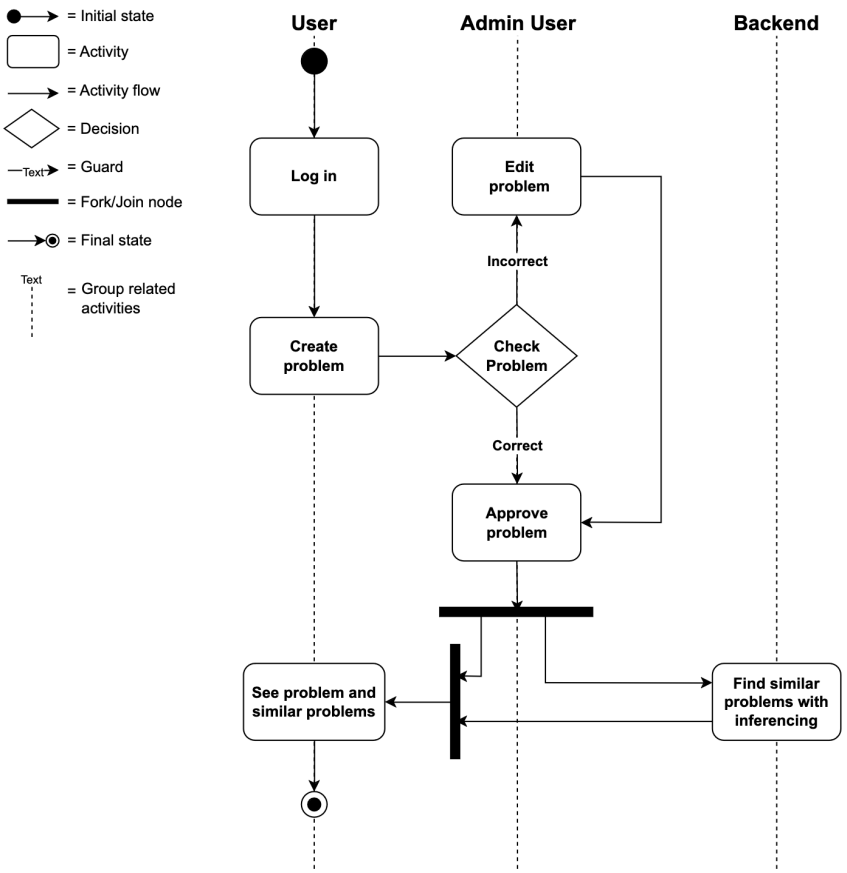


Figure 3.2 - A process view of the project.

Development View

The development view primarily focuses on the organization of software modules in a development environment and describes subsystems that can be developed by one or more developers. This view is from the perspective of developers and managers of the system (Kruchten, 1995). Figure 3.3 visualizes all implemented files in both the frontend and backend in their respective folders and their connections to every other implemented file. Most of the connections are color-coded with a defined meaning (e.g. red arrow indicates that a file in the folder *utils* is used in another file). In an attempt to make the view not too cluttered, forking a lot of the arrows was necessary. The yellow arrows indicate "*component used in…*", and the blue arrows, "*hook used in…*", fork because they can be used in multiple files. However, the green arrows, indicating *communication with API*, are grouped together based on having similar API calls, mostly to prevent having 10 different arrows go to the same file. Additionally, for overlapping lines with the same color, a dashed line where they overlap is to avoid confusion about which line goes where.

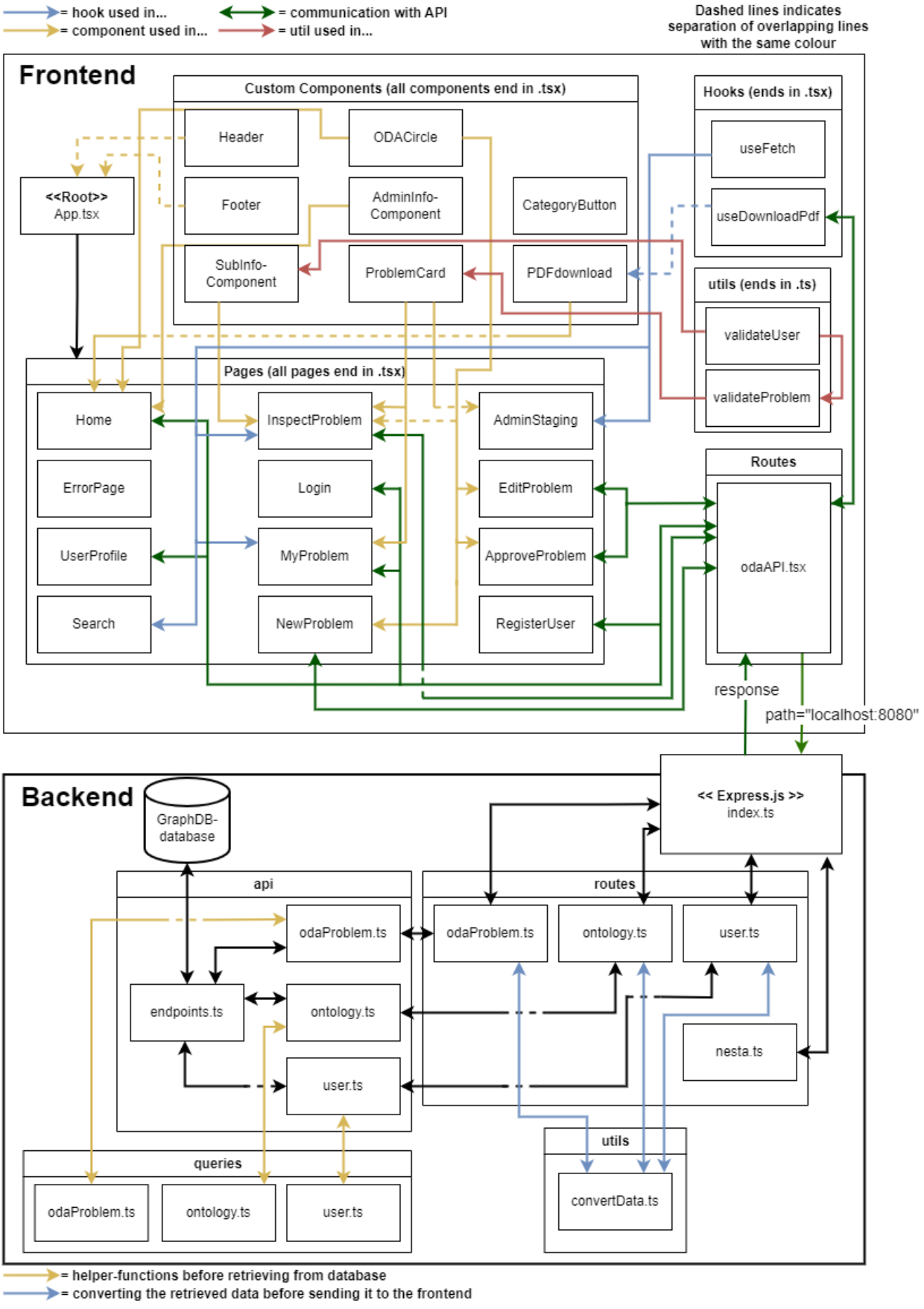


Figure 3.3 - Development view of the project.

Physical View

The physical view is concerned with the software component's topology on the physical layer, alongside the components' physical connections (Kruchten, 1995). The following figure portrays the system's physical view.

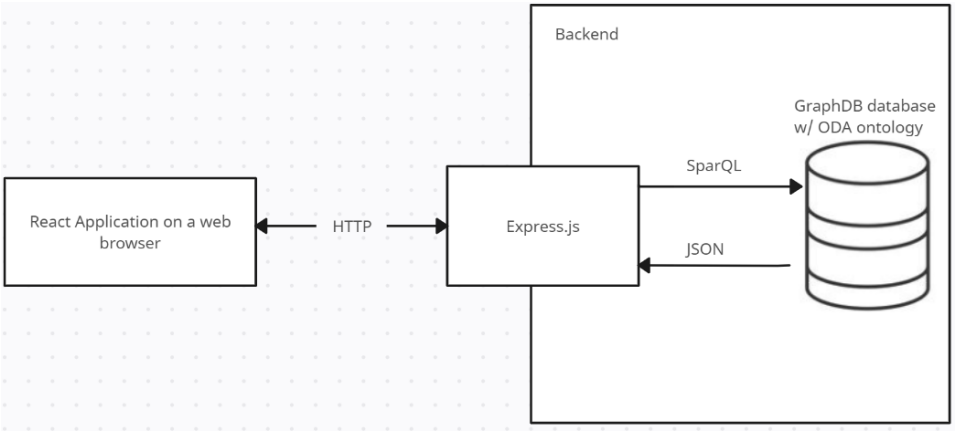


Figure 3.4 - Physical view of the project

Scenarios

Scenarios are used to capture the most critical functionality of the system (Kruchten, 1995). Figure 3.5 is a use-case diagram that displays these critical functionalities.

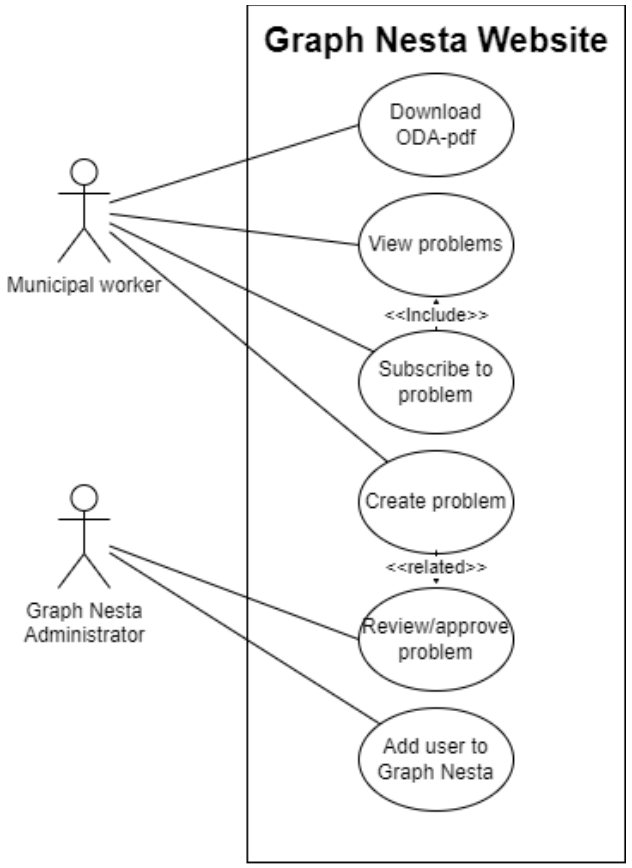


Figure 3.5 - User scenario of the most critical functionality of the product

3.4. Testing

A combination of manual and automated software testing was used to test the product. Initially, manual testing was performed, and later on, automated testing tools were employed to create and execute tests. Moreover, a validation test was conducted with test subjects being facilitated by Trondheim municipality to obtain feedback on the overall application's usability. All automated testing results can be found in *Appendix E*.

3.4.1 Unit Testing

For unit testing the group resorted to *Jest*, which comes alongside React when initiating a project. The frontend unit tests mainly involved testing the React components. Since most of the tested components are simple, the tests only verify that they render correctly and handle user events properly. The *ProblemCard.tsx* test required slightly more testing, as that component deals with numerous react props that require validation.

3.4.2 Snapshot Testing

Jest was also used for snapshot testing. Snapshot testing involves capturing the rendered output of a component and comparing it to a previously saved snapshot to detect differences (Tsadok, 2020), allowing the group to catch and fix any unintended changes during development.

3.4.3 End-To-End Testing

Cypress was utilized fo*r end-to-end testing. Cypress* is a JavaScript-based testing tool that can test the frontend of the project (Cypress, n.d.). With end-to-end testing, the group could test the behavior of the project and check that the flow of data was maintained for all types of user tasks and processes. The group chose to use *Cypress* because members already had experience in utilizing it, and therefore it would take considerably less time to set it up as well as write adequate tests compared to using a new and unfamiliar testing tool. When writing tests, the group envisioned potential sequences of actions end-users may perform while navigating the website.

3.4.4 User Testing

Our original plan was to conduct an early user testing session, using the Figma prototype we created in the first sprint. However, during our weekly meetings with the customer, we got to demonstrate the Figma prototype multiple times and received extensive feedback on the layout each time. Although no formal user testing was conducted, the feedback helped us implement a frontend that was more suitable for the customer's needs, and as a result, we felt that user testing was unnecessary as the frontend part of the project approached its intended state. The Figma prototype was therefore neglected, and we decided to wait with usability testing until we had a running prototype, and not just a visual mockup using Figma.

In retrospect, we realize that the approach we took could have had negative consequences if it was not for the extensive feedback we received from the customer. Usually, when performing agile development one would arrange user tests early in the design phase, before the actual coding of the frontend starts (UserTesting, n.d.). A visual prototype design tool like Figma can be used to gather feedback and alter the prototype until it reaches a desired state. Then one can program the actual frontend in accordance with the prototype, ensuring

that it remains user-friendly and suitable to the user's needs. Conducting a user test on a close-to-finished user interface is not preferable, as altering code is much more time-consuming than altering a Figma prototype.

3.4.5 Validation Testing

On the 26th of April, we arranged a validation testing session with the customer representative, Leendert Wienhofen, and a colleague of his, Ashley Muller - program leader of DigiTrøndelag. The purpose of a validation test is to ensure that the product is satisfactory to the customer's wishes. The customer had suggested letting Ashley test the product, as she was already familiar with NESTA and the ODA method, and is a prime example of someone in the application's target group.

The validation testing worked much like a user test, where we had the test subjects perform tasks that showcased the core functionality of the application while thinking out loud. The customer was the first test subject and was asked to navigate the application on his own, using one of the group members' laptops. Ashley was the second test subject and had to participate digitally over Google Meet, and thus navigated the application "by proxy", telling a group member where to click and scroll by looking at a shared screen displaying the user interface.

We found that testing the application "by proxy" was a surprisingly effective way of going through the tasks in the tests. This is likely because it is more natural to vocalize your thoughts when explaining to another person how the application should be navigated, as opposed to when you are exploring the application on your own. Additionally, we had only planned for the testing to be performed on a MacBook, which proved to be difficult for users who were not accustomed to using MacBooks. This was not an issue when performed by proxy, as in that case the "pilot" was already familiar with the MacBook. However, the team should have researched what the test subjects were familiar with in order to remove any surprise variables during testing.

In general, the product fulfilled customer expectations. However, we did receive some suggestions about changing the wording and adding more responsive user feedback upon interactions within the application. For example, no confirmation message was given to the user when they published an ODA-problem, leaving them unsure if their action had been registered.

The validation testing proved to be useful, as it gave us points to improve. But most of all, it validated for both us and the customer that the proof-of-concept prototype was successful. When asked whether or not our product was something Ashley would like to use in the work with DigiTrøndelag, the answer was a resounding yes. Furthermore, she asked us for permission to present screenshots of our product at a meeting with IT- and department managers from almost all municipalities in Trøndelag County. To add to the already great feedback, we received a message that DigiTrøndelag would like to create a production version of our product.

3.5 Product Development

3.5.1 User Stories

User stories are made to form requirements from the user's point of view. These requirements can be both functional and non-functional. Not all of the requirements were made from user stories, but the ones that the user sees and interacts with were.

The definition of "done" is a set of formal criteria for a user story, where all criteria have to be satisfied for the user story to be considered fully implemented. This set of criteria helps guide developers in the pre-implementation activities such as discussion and design. In addition, having an explicit checklist helps reduce the odds of misunderstanding the task between developers. However, not explicitly writing out the definition of done may result in less effectiveness of the method. ("Definition of Done," 2015). Generally, our definition of "done" is that for a new feature:

- All required functionality is fully implemented
- All unit tests are satisfied
- Implementation is reviewed by another developer during merge
- Implementation is merged into the main branch of the project

It should be noted that the definition of done for certain user stories may not follow the aforementioned bullet points. An example of this is the requirement to satisfy WCAG 2.1, which can not be tested using unit tests, and should instead be measured by conducting a usability test. The user stories listed below all start with the following statement: "As a municipality worker, I want …".

Table 3.3 - User Stories

ID	User story	Related requirements
1	to access the solution through a web-based interface so that I can access it from any location.	**FR01**
2	to post my data-related problems so other municipalities with similar tasks can contact me and start collaborating	**FR02**
3	the application to require problems to be formulated using the ODA method so that problems are more efficiently and accurately described.	**FR06**

4	to add data to the database so that I can share information with others.	**FR07**
5	to edit a published post so that I can correct any mistakes.	**FR08**
6	to delete a published post so that I can remove incorrect or outdated information.	**FR09**
7	to search and filter for other users' posts so that I can find relevant information more easily.	**FR10**
8	to have role-based access control for editing so that I can help other users with the correct formatting of problems.	**FR11**
9	to see the status information for a problem that is similar to mine so that I can see if they have come further in the process than me.	**FR12**
10	to see some form of contact information for other municipality workers so that I can get in touch with those who share my problems.	**FR13**
11	to subscribe to a problem so that I can receive updates about its progress.	**FR14**
12	the solution to have a consistent user interface for easy navigation so that I can use it without any difficulty.	**NF01**
13	the application to have a high contrast between elements so that people with poor eyesight can use it.	**NF02**
14	the solution to be secure from unauthorized access so that my data is protected.	**NF06**
15	the solution to be compatible with several modern web browsers so that I can use it on any device.	**NF05**

16	the solution to handle multiple users simultaneously so that I can collaborate with others efficiently.	**NF03**
17	As a municipality worker currently situated in Norway, I want the solution to fully load within 2 seconds so that I can use it without any delay.	**NF04**

3.5.2 Figma Prototype

Former experience with Figma prototyping led the group to create a *Figma prototype*, as shown in *Appendix B,* since it would help to create a common understanding of how the user interface is supposed to look. Moreover, creating the visual representation of the application without writing code would save time, and would help the group present the envisioned user interface to the customer. This enabled the customer to give feedback before any implementation had been worked on and enabled the group to adjust the prototype accordingly. Which again saved time by reducing time spent on the implementation of visual components the customer might have wanted to change later in the development process.

The Figma prototype proved to be a useful tool for the actual coding process as well. It acted as a set of goals to reach when programming the frontend part of the application, where we tried to replicate the design of the prototype. The final design of the product ended up looking very similar to the prototype, with some minor design changes to for example buttons, icons, and drop-down menus.

Design & Color Scheme

The design and color scheme was heavily inspired by Trondheim municipality´s graphical charter[8]. This was not a requirement by the customer, but the group chose the design and color scheme because the product was a proof of concept, and could be further developed by Trondheim municipality.

While the group would have preferred a darker color scheme to reduce light emissions and be more environmentally friendly, we ultimately decided to prioritize following Trondheim municipality's established design and color scheme.

3.5.3 Implementation

Backend

As mentioned in *section 3.2.2*, the backend has been connected to Ontotext GraphDB. The main flow of the backend consists of querying the database with HTTP requests, which send the retrieved data to the frontend through the Express Framework. Considering the project was a proof-of-concept, the implementation of the backend was still lacking some quality assurance by the end (see section 3.6 for future work).

[8] https://sites.google.com/trondheim.kommune.no/grafiskprofil

Connecting to the database is done through a combination of HTTP requests through the Node package Axios[9], and self-written SPARQL queries stored in the backend. Writing the SPARQL queries took up the majority of the time in creating the backend. Therefore, a lot of time was spent in the SPARQL part of the GraphDB browser as seen in Figure 3.6. With the mentioned Express framework, the backend has an API with clear URLs and query parameters that are easy for the frontend to use and easy to document.

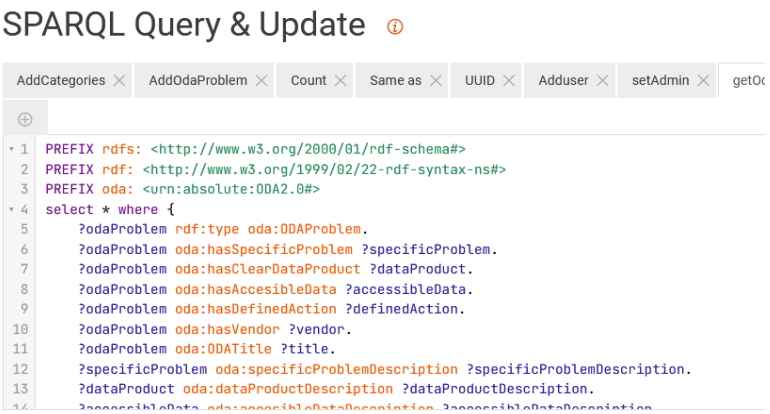


Figure 3.6 - Screenshot of some SPARQL in the SPARQL page of GraphDB's user interface

A main goal of the connection between the frontend and the backend is to hide as much of the complexity of the solution as possible. The backend does a lot in terms of mapping similarities between problems, creating a lot of new nodes and arcs. While this functionality can be visible and understood by admin users, the end-user should be isolated from the system working in the background. This has also been a larger focus in the implementation of the frontend.

The inner workings of the project ontology

The project's ontology is the primary focus of the project, and therefore, a more detailed explanation of its inner workings is necessary. Starting with the class hierarchy (see Figure 3.7), the most important class "*ODAProblem"* contains the ODA-problem individuals, which the group aims to pair together based on relevance using the "*relatedODAProblem*" relation. Moving on, the classes *accessibleData, clearDataProduct, and specificProblem* are constructed in accordance with NESTA's ODA method and have their own subclasses. It is through these subclasses' relations with the ODAProblem-class that problems are paired together; the exact details of how this pairing works are going to be expanded upon in the next paragraph regarding inference. The rest of the classes such as *user, DefinedAction, and sector* are not relevant for inference but aid in information storage and retrieval.

[9] https://github.com/axios/axios

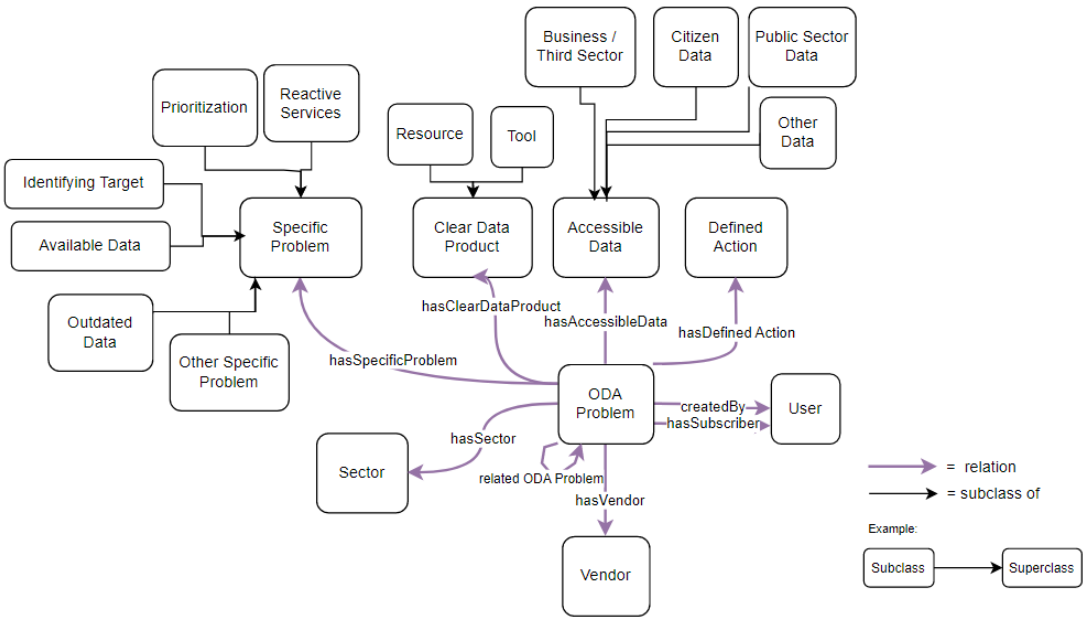


Figure 3.7 - A class and relation diagram for the ontology. "Subclasses of…" are represented by black lines, while relations are depicted by purple lines. Most inverse relations, such as *creatorOf* (which is the inverse of *createdBy*) have been omitted to minimize clutter. It should be noted that all classes in the diagram are subclasses of the root superclass.

Inference in the project's ontology works by way of property chaining (forward chaining), which is a process that allows us to specify that two entities are related to each other, indirectly, through a chain of intermediate entities. In other words, it starts by taking some known facts and combining them with rules to generate new facts, this process is repeated until no more new facts can be produced.

In the project's ontology, each new individual problem is assigned a relation to *Clear Data Product, Accessible Data, Specific Problem, and Vendor* by a user. The group ontologists have decided to call those relations *HasSpecificProblem, HasAccesibleData"* and so on. These relations go both ways, meaning the inverse relations (from these 4 classes to *ODAProblem*) also exist. These are called *isDataProductOf, isVendorOf,* and so on. Each individual in these 4 classes is related to all other individuals in the same category (subclass), i.e. graph and pie chart, both belong to the subclass *Resource* and are thus related by a relation which the ontologists have called *SameCategoryAs.* When two problems have a relation to two individuals belonging to the same category, a connection between them is automatically inferred. This connection also describes which property ties the two problems together, i.e. *RelatedAD_ODAProblem* is a relation that states that two problems are related because they share a common (or similar) *Accessible Data* source. This, combined with a property chain that states (paraphrased in plain English) "if a problem has an *Accessible Data* source, and that source is the *Same Category As* another source which is the *Accessible Data Source Of* another problem, these two problems are related".

The webpages of the application

Home page

Once a user is logged in, they will be presented with the application's home page. This page contains a short description of the purpose of the application, along with buttons to access the "Search page" and the "New Problem Page". Furthermore, the homepage contains a brief description of the ODA method, provided by the Norwegian translation of the NESTA guide.

No matter what page the user is currently looking at, there will always be a navigation bar at the top and a footer at the bottom. Through the navigation bar, the user can interact with an extendable “hamburger menu”, from which the user may access any of the other pages. Meanwhile, the footer contains a button that, when clicked, scrolls the current page to the top. The home page, the footer, and the navigation bar are mainly designed according to the Figma prototype.

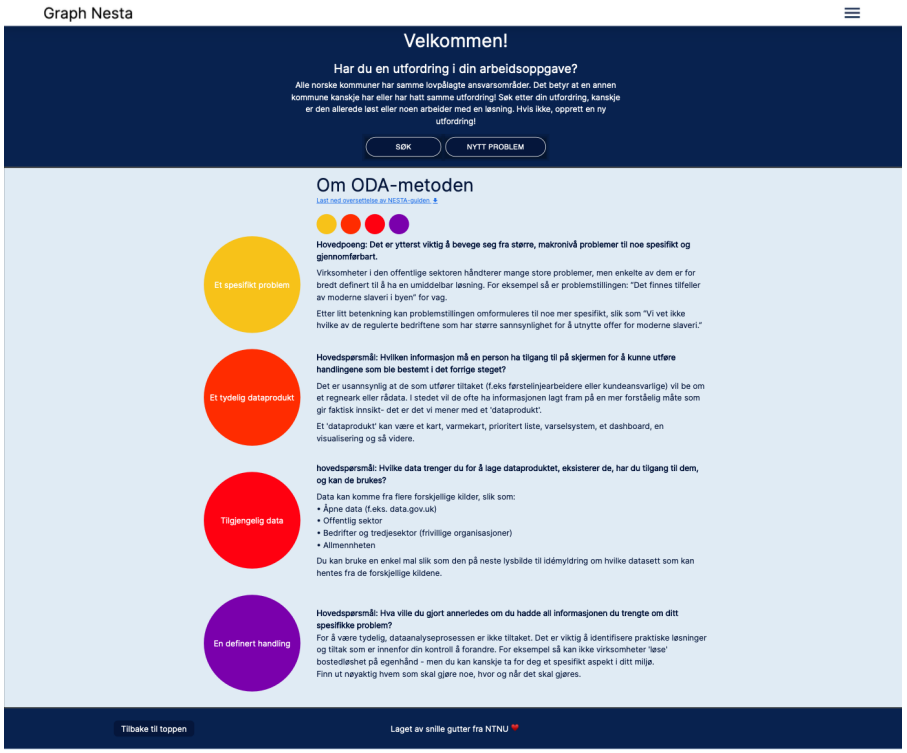


Figure 3.8 - Home page

Search Page

The “Search Page” is where a user may look for specific data problems that have been created and stored in the database. The user has the ability to filter their search by using the search bar, filter menu, and category buttons. A delay has been implemented, when typing in the search bar, to reduce requests and hence be more environmentally friendly. By clicking on one of the problems that appear, one is redirected to the “Inspect Problem page”, where all information about that specific data problem is displayed.

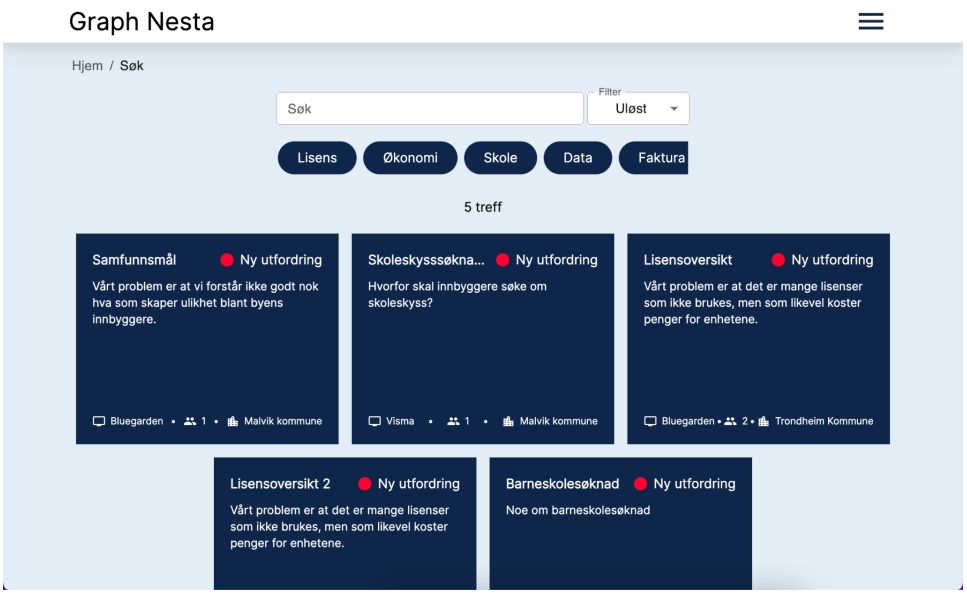


Figure 3.9 - Search page

Inspect Problem Page

This page displays all information about the chosen data problem. In addition, contact information is displayed, along with a list of subscribers to the data problem, and most importantly, a toggle functionality is available to show similar data problems. If the user either is an administrator or the owner of the data problem, an edit button appears, which makes it possible to edit the data problem.

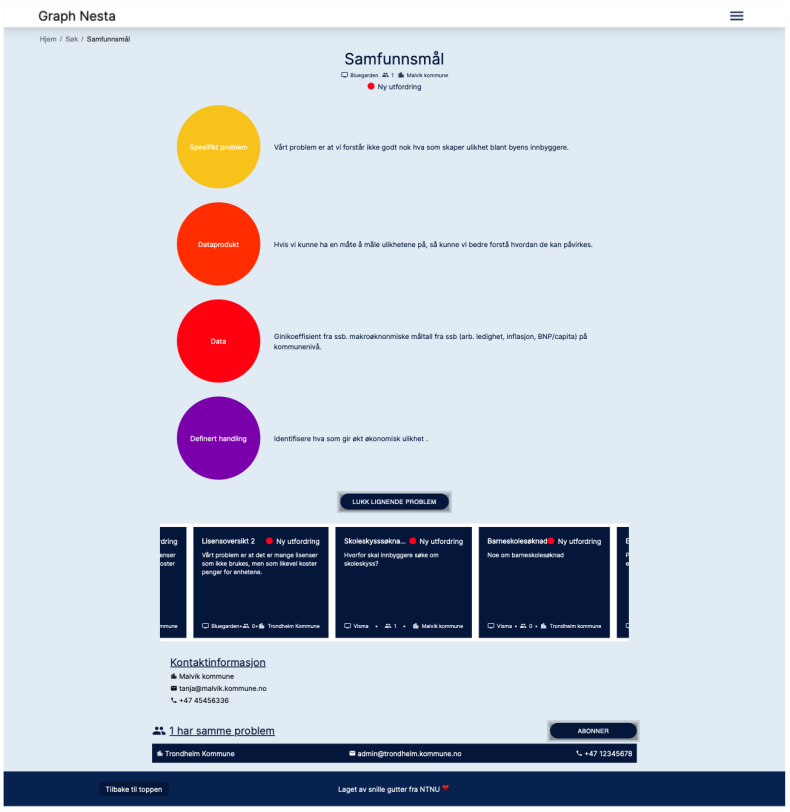


Figure 3.11 - Inspect problem page

New Problem Page

At the “New Problem Page” a user may create a new problem to be added to the database. The user must choose a title and the system in which the problem occurs. Furthermore, the problem must be described according to the ODA method. Therefore, the user has to fill out a text field for *Specific Problem*, *Clear Data Product*, *Accessible Data,* and *Defined Action*. There is implemented a tooltip (question mark) in the upper right corner of each text field to help the user insert text correctly according to the ODA method.

When the user and all the requirements are satisfied, a “Send” button may be clicked, upon which the problem is ready for admin review, which is part of the staging process.

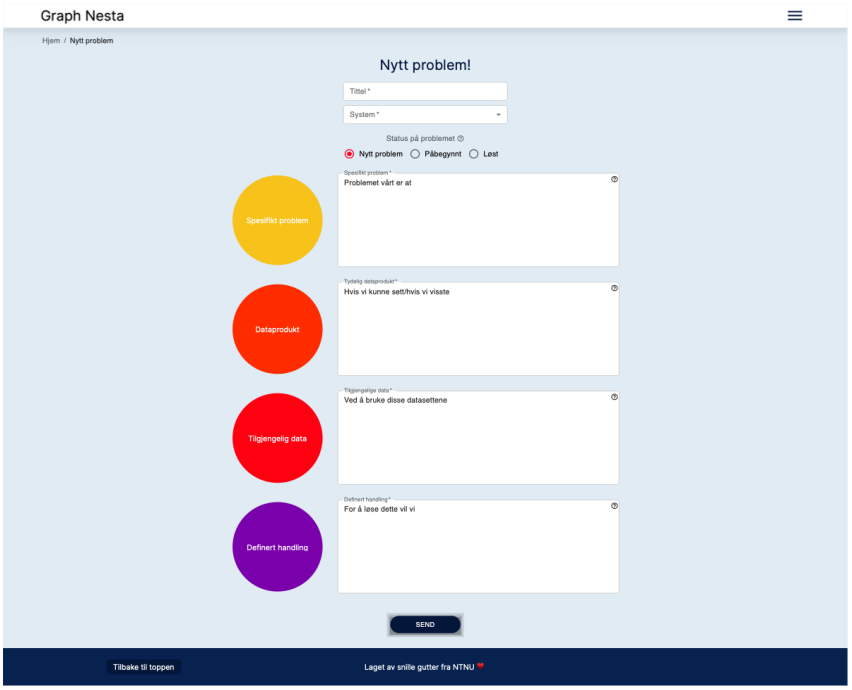


Figure 3.10 - New problem page

My Problems Page

The “My Problems” page is where users can see the problems they have added to the system and the problems they have subscribed to. Below the navigation bar, there are two buttons; one for displaying each of the two categories. When one of them is clicked, the relevant problems appear on the screen in the form of “Problem cards”.

Figure 3.12 - My problems & subscribed problems page

Admin Panel

If the user has admin privileges, the user interface will provide access to some additional functionality. From the HomePage, an admin user will be able to access the “Approve Problems” page. This page is where the admin reviews problems that have recently been created by regular users and have not yet been approved. The problems appear in the form of “Problem cards”, and are clickable. When clicked, the web browser is redirected to the “Approve Problem” page, which shares mostly the same layout as the “New Problem” page. The admin can edit the required fields, add categories (important for inference) and approve or delete the data problem, to ensure that it satisfies the requirements of the ontology. Once a problem is approved by an admin, it becomes visible to all users.

Figure 3.13 - Admin panel

3.6 Future work

3.6.1 Inferencing

One of the major goals of the ontology was to include inference so that we could relate ODA problems by their similarities. In our proof-of-concept version of the product, we have achieved this by relating them by similar categories. This is a one-to-one relationship, meaning there is no grade of similarity to say that one is more alike than the other. We consider this a general issue with ontologies, as the inferencing is mathematically based on the relation either being true or false.

A solution we discussed with our customer was to find similarities in the text of the problem, such as in the title or the specific problem part. From early in the project, we considered this to be an advanced functionality less relevant to the prototype, and we were unsure if we would have enough time to implement it. However, a possible solution was found using the similarity index functionality of GraphDB. This allows us to create a similarity index based on specific data in the database such as all the specific problem fields. You can then run a search on one specific problem returning an ordered list with a number from 0 to 1, listing the other specific problems that are most similar first.

3.6.2 Backend Improvements

Despite the fact that most of the important functionality has been implemented in the backend, there are still multiple areas of development that should be taken into consideration.

First of all, the backend lacks proper validation of data coming in from the frontend in its current state. It also lacks some proper feedback when a query doesn't return successfully. This still has the potential to crash the system or fail to provide the user with feedback in case of a problem not being posted to the database due to an error. The SPARQL queries still have a problem in which a user cannot pass in quotation marks as part of a text, or take in a multiline string. This is not a limitation of the database, but a formatting issue of the data that needs to be taken into account in the future. Functionality-wise, the backend still needs to implement ordering of ODA problems. The SPARQL query for getting ODA problems supports this but hasn't been prioritized so far.

The backend would also benefit a lot from implementing some form of testing, which has been deprioritized in the project. Specifically, having the ability to test all queries in a safe test environment before merging different branches would save a lot of work when something stops working. A possible solution to testing the backend endpoints would be to use the Jest and *Supertest* frameworks.

An important aspect of the backend in a production environment is security, which we have not taken into consideration for our proof-of-concept. This would need to be a major prioritization in the continuation of the project. The group was aware that in case the system was to be further developed, Trondheim Municipality would implement its own login and authentication system. The group would also have to consider the sensitivity of different data being sent, and if it is currently being sent securely.

3.6.3 Frontend Improvements

The frontend has implemented all core functionalities that satisfy the requirements of the project scope, but there is still room for improvement. As of now, the search page does not support pagination. The search page was supposed to have infinite scrolling implemented, but this was not a requirement from the customer and was therefore deprioritized. However, in the code we have arranged the possibility to add pagination, and it should therefore be a simple task to implement in the future. As described in the backend part of future work, the search page also lacks the ability to order its data problems from solved to unsolved.

The frontend has no “edit user” feature. This was also not prioritized because it is unimportant for the prototype. As written in *3.6.2 Backend Improvements* above, implementing Trondheim municipality's own login and authentication system is future work. Hence, all user-related data would be handled by that system, rendering an “edit user” page redundant.

The application's authentication check is executed by using protected routings from *ProtectedRouting.tsx*, which minimizes duplicate code. However, it is not an optimal solution since it renders a page for all users, including those who should not have access to certain pages when not logged in. In this case, the user will quickly be redirected to the login page. While this happens quickly and is not a significant issue, it is not the best solution. Security was not a concern in regards to this project, and was therefore not prioritized either.

The application’s authentication will therefore also redirect administrators to “/LoggInn” when refreshing the page at “/GodkjennProblem” and “/RegistrerBruker”. This should be fixed in future work using Trondheim municipality’s own authentication system.

The admin view currently does not have an overview for administrators to see all user profiles, see a specific user's ODA problems, and delete and edit users. This was also not implemented because it was deprioritized for this prototype.

The Web Content Accessibility Guidelines (WCAG) is an international standard for web accessibility published by the Web Accessibility Initiative (Henry, 2023). As a governing body Trondheim municipality needs to comply with the WCAG standard, but it was deprioritized in this prototype. We have verified color contrasts, which meet the requirements, but we have not done any more testing in regard to the standard. Future work will therefore be necessary to meet all requirements set by the WCAG standard.

4 Process

4.1 Project Management

4.1.1 Project Plan

Early in the project a development plan was set up. Here important dates and days were set up for deliveries, meetings with the customer, and meetings with the supervisor. As well as displaying meeting times for the group throughout the week. All of this can be seen in *Figure 4.1* underneath. To ensure that the group would stick to the planned meeting times, a group contract was formed. In this contract, penalties were defined

which members would receive if they were not able to abide by the rules regarding meeting on time. This group contract may be seen in *Appendix D* under *Group contract*.

Week	Monday	Tuesday	Wednesday	Thursday	Friday	Comments
3					Customer	
4	Customer				Supervisor	
5			Customer			
6		Supervisor	Customer		Preliminary	
7			Customer			Monday meetings dropped
8		Supervisor	Sprint 1 | Customer			
9			Customer			
10		Supervisor	Midterm | Customer		Sprint 2	
11			R&SA | Customer			
12		Supervisor	Customer		Sprint 3	
13						Excursion
14						Easter vacation
15			Customer			
16		Supervisor	Customer			
17		Sprint 4	Customer			Validation testing
18		Supervisor	Customer			
19			Customer			
20		Supervisor	Final			

Sprint start / end
Deliverables
Customer meeting
Supervisor meeting
Group meeting
Vacation / Excursion
Group meeting times (core)
Tue: 10.15 - 17.00
Wed: 12.30 - 15.00
Thu: 12.15 - 15.00
Fri: 10.15 - 14.00

Figure 4.1 - An overview of the project timeline.

4.1.2 Process Model

Early on in the project, we held a meeting to determine which work practices would be implemented. Since most of us have experience with agile development, we had a thorough understanding of its practices and discussed their advantages and disadvantages. Eventually, we agreed on several practices, with the most important being listed below.

Daily meetings

Most agile development teams conduct daily meetings (*16th Annual State of Agile Report.*, n.d.), but the length and content of the meetings can vary widely. We decided to conduct daily meetings because we had experienced from previous project work that they facilitate communication within the development group; by bringing us together and sharing information so that we become aware of what other members are working on and can adjust our work accordingly. Moreover, it makes the meetings more structured, saves time, and keeps them professional.

We chose to start by following the recommendations from Kniberg (2015) in our implementation of daily meetings. Among other things, we wanted to follow the recommendation to keep the meeting duration to a maximum of 15 minutes, answer 3 basic questions, and not devote time to discussing solutions to programming problems. Later on, we thought about experimenting and trying to adapt the daily meeting to better fit the group. For instance, we thought about devoting some time to discussion of possible solutions. Group members often have different skills, and can offer new perspectives and solutions that you would not have come up with yourself. However, this idea remained a mere thought and was never implemented in the daily meetings.

Once the actual coding started, and the group split up, the daily meetings became increasingly important. Group members often tended to focus on specific areas of the project and could thus lose track of the current state of the tasks assigned to other group members. Daily meetings ensured this became less problematic.

Backlog

Our group recognized the importance of a well-implemented project backlog, drawing on positive experiences from previous projects. The backlog enabled us to effectively plan sprints, prioritize important tasks, and ensure that important project features were developed. It was essential given the project's extensive scope and helped to maintain an overview of the project. The backlog was also a useful tool for aligning the project with the customer's needs and expectations, ensuring that the end product would meet the requirements.

Sprints

During the sprint planning phase, we decided on group partitioning for the duration of each sprint. We also utilized pair programming for new or challenging tasks, such as the implementation of the ontology. Without a clear picture of the requirements and the workload that followed from an ontology project, it was deemed reasonable to delay the decision of sprint lengths. In the end, sprints of approximately two weeks each were decided, which started after the preliminary research phase.

During our sprints, we prioritized effective planning by creating a backlog of development tasks with varying priorities. We then selected tasks based on factors such as available time, task importance, and dependencies.

Retrospective

Given that the group was developing an ontology from scratch and working with new technology for the first time, we expected to make some initial missteps. To improve the group's performance and avoid repeating mistakes, we planned to hold retrospective meetings. These meetings would offer an opportunity for the group to reflect on their work and identify what went well, what could have been improved, and what mistakes were made. This practice promotes learning and keeps the group from making the same mistakes repeatedly. For our retrospective meetings we decided to use the free agile retrospective tool *Metro Retro*[10] and write down under the following points, 10 minutes for each point:

- What went well?
- What can be improved?
- What actions can we take to improve areas we face problems in?

Kanban board

To visualize our group's workflow, we utilized a “Kanban board” integrated within GitLab (Gitlab Issue Boards). The board helped the group prioritize getting certain tasks done, by limiting the number of tasks that can be worked on simultaneously. As mentioned before, because of the extensive size of the project and our group, multiple layers of planning are required to ensure a smooth and efficient development process. The Kanban board acts as one of those layers.

[10] https://metroretro.io/

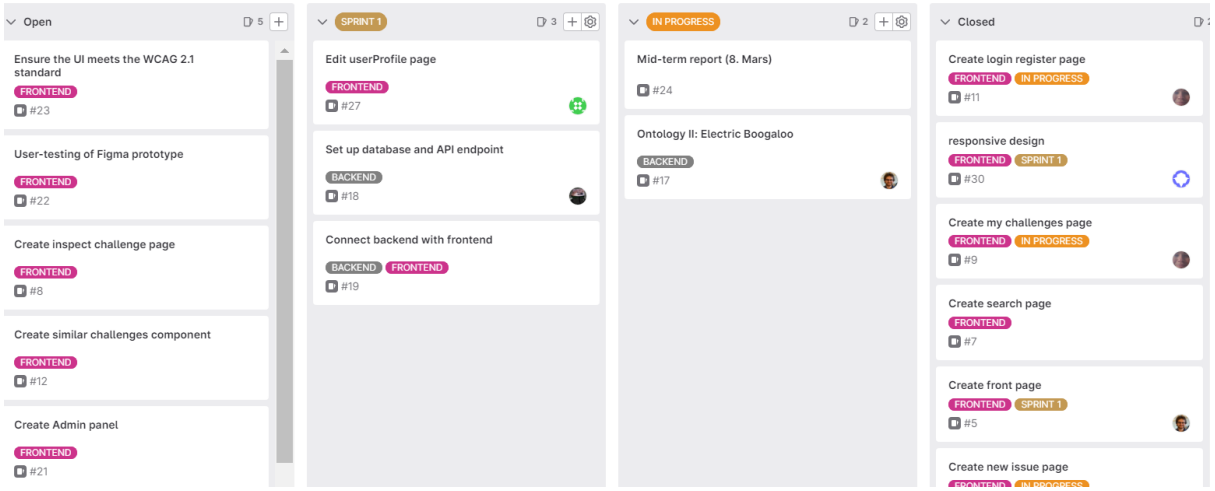


Figure 4.2 - Screenshot of the Kanban board from Sprint 1

4.2 Risks

The group conducted a risk assessment to proactively identify any possible complications that could arise with the product or within the group while working on the project. These assessments were assigned a probability of occurring and the magnitude of their consequences. Both the probabilities and consequences were categorized into four levels: low, moderate, high, and very high. These probability-consequence pairs allowed a 4x4 risk matrix to be constructed to create a more visual representation of the project risks. This matrix has color coding that represents which risk value a matrix cell has, i.e., green is *low* risk, yellow is *medium* risk, orange is a *high* risk, and red is *very high* risk. For each risk, the group developed measures to reduce their probability and/or consequences, and as such an additional risk matrix was constructed with updated values. The risk assessment can be found in *Appendix A*, while the risk matrices can be seen in Figures 4.3 and 4.4.

One of the more severe risks we have acknowledged as a group was the risk of us not being able to properly implement our ontology for the system (R11). As mentioned, developing an ontology is complex, so the likelihood of having to remake the ontology multiple times is high. Additionally, the project ontology is a vital part of this project, and failing to create a sufficient ontology will make this project infeasible. To mitigate the risk of this issue we concluded that thorough research of ontologies, careful planning, and frequent discussions about the implementation would help reduce that risk.

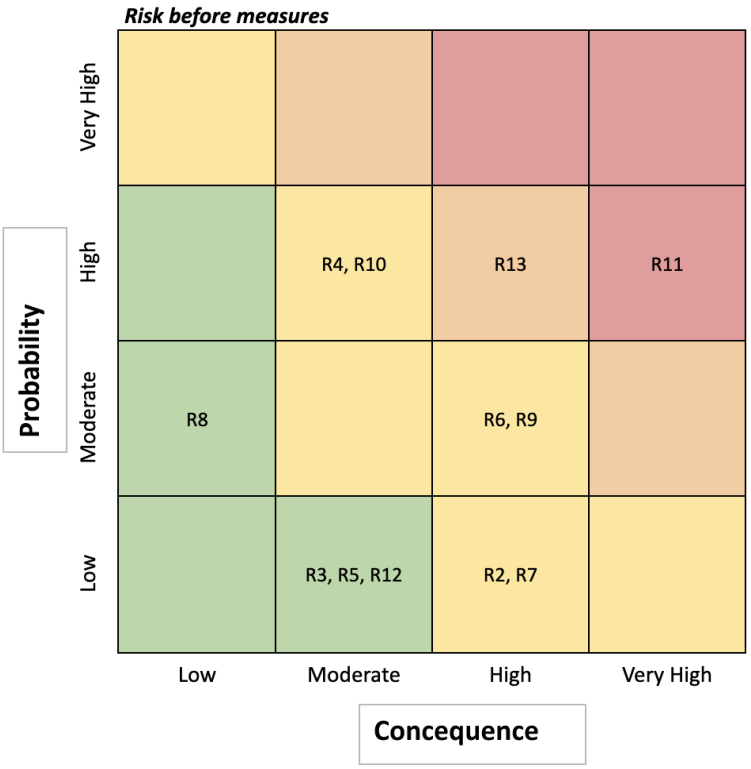


Figure 4.3 - Risk matrix evaluations before measures

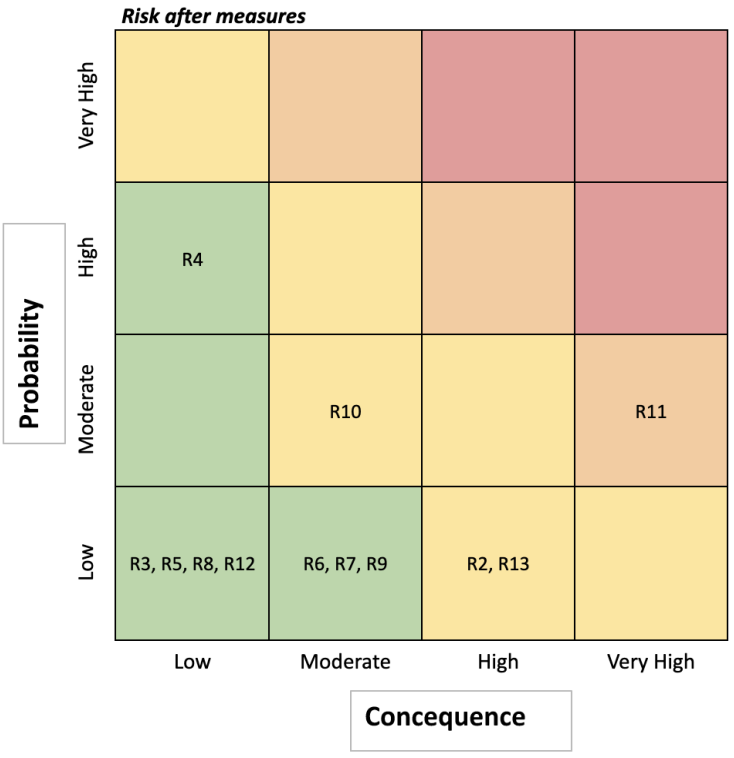


Figure 4.4 - Risk matrix evaluations after measures

4.3 Group Organization

4.3.1 Roles and Responsibility

Initially, during the preliminary research of the project, the group members did not have specific roles assigned. It was deemed more advantageous for the group to work together on key parts, for example when learning about ontologies, which was a key part of the project. The group planned to divide itself into several smaller groups based on roles when the first sprint started after the preliminary research had finished.

The group was divided into three smaller groups: frontend, backend/database, and ontology. These groups were fixed, but towards the end of the development process, group members would move between them if needed to effectivize working hours. Three people were primarily placed on the frontend, two on the backend/database, and two on Ontology during the first sprint. In the later sprints, two people worked on the frontend, one on the backend/database, two on ontology, and two on the project report. As in the first sprint group members moved between the groups as needed underways.

4.3.2 Group Collaboration

Throughout the project the collaboration between the group members was largely frictionless. The group focused on open and transparent communication from the start. This communication was facilitated through meetings and the usage of the communication application *Slack*[11].

However, some minor issues with collaboration occurred throughout the first sprint of the project. Because of fluctuating meeting times as well as meeting changes on short notice, some on the same day, members of the group had forgotten where and when to meet. This created some irritation in the group, and after the retrospective meeting, the group made some clarifications about meeting times and meeting locations. After these clarifications, the issue disappeared.

The group created a Google Sheet early on in the project to log working hours. In this sheet, the group logged hours spent on the project per day and what they had worked on. By doing this the group got a transparent overview of what other members were working on. Plus every member got a place to see how much work they had done per week, with a goal of between 15 to 20 hours. Timesheet information can be seen in *Appendix D* under *Timesheet*.

4.3.3 Customer Collaboration

Once the group had been formed, immediate contact was made with the customer who agreed to meet soon after. It was decided that communication between the parties outside of meetings would take place on Slack. Additionally, weekly meetings were scheduled to ensure both parties remained up-to-date on the project's progress. Both parties also agreed on canceling the weekly meeting if there were no significant updates to share for that specific week. This was to remove unnecessary meetings that would just be a formality and would not add significant value to the development of the product.

[11] https://slack.com/

The customer was helpful in the beginning and provided tips for how to understand ontology concepts. He sent a plethora of articles on the topic and suggested creating a “pizza ontology”. These tips were of immense value to the group in the early phase of the project.

Regular communication with the customer helped establish trust and ensured that the requirements and expectations were fully understood and met throughout the development process. This allowed the customer to provide feedback on the product as it progressed, enabling potential issues and opportunities for improvement to be identified early on. This, in turn, encouraged proactive problem-solving and resolving issues before they could become major roadblocks.

4.3.4 Supervisor Collaboration

Claudia Maria Cutrupi was assigned as the group’s supervisor for the project. The communication between the group and the supervisor was through emails and meetings. Meetings with the supervisor were less frequent than with the customer, only every other week. Since there were no noteworthy issues with the collaboration in the group, the meetings were for the most part related to the report. The supervisor's feedback on the report's structure proved valuable, enabling the group to refine the document's structure and emphasize its important aspects.

4.4 Development Tools and Standards

4.4.1 Coding Standards

With a group consisting of seven developers it is important to have a common coding standard. To ensure that all members followed a consistent standard, we implemented *ESlint*[12] for linting and *Prettier*[13]for code formatting. These widely used tools, which some of the group members were already familiar with, helped speed up the development. The group did not require a specific development environment. The only requirement for the development environment used was that they supported *ESlint* and *Prettier*. In the end, most of the group ended up using *Visual Studio Code*[14] and the rest *WebStorm*[15] as their IDE.

4.4.2 Git Standards

Early in the project, the group established Git usage standards. To ensure that code was peer-reviewed and maintained adequate quality, committing to the main branch was prohibited and merge requests had to be approved by at least one other member of the group.

To ensure clearness between branches and issues on *GitLab* the group decided on calling branches using the following format: *[issue number]-name-of-issue*. This made it clear what issue a branch belonged to. The group also agreed on making clear and concise commit messages to make it easier for other members of the group to look at changes made.

[12] https://eslint.org/
[13] https://prettier.io/
[14] https://code.visualstudio.com/
[15] https://www.jetbrains.com/webstorm/

4.4.3 Documentation

The group agreed early on in the project that code should be commented on. By using *Typescript* the group had to use strict typing, which made the code less vague. With strict typing and good coding practice documentation was mostly unnecessary. However, the group agreed that if the code parts are ambiguous they should be commented on. DigiTrøndelag wishes to complete a production version of the product, and thus extensive documentation simplifies their task of taking over development. Documenting issues, experiences, lessons learned, and more in the report was also part of the documentation requested by the customer.

4.4.4 Ontology Editor

The group discussed using either *WebProtégé*[16] or *Protégé*[17] as the ontology editor and framework. In the beginning, the group used *WebProtégé* as it enabled online collaboration. However, we quickly discovered that *WebProtégé* was unfit to use as an editor and framework. It was missing several crucial features needed to make a working ontology, like inference, properties, and helpful plugins. The group, therefore, changed to the *Protégé* desktop application that included more advanced functionality.

When the group changed to a local editor, collaboration became more difficult. Therefore the ontology was mainly created by a few members of the group, through pair programming or using screen sharing.

4.5 Preliminary Research Phase

During the preliminary research, we devoted most of our attention to researching ontologies and their potential use in our project to facilitate a shared understanding of information related to problems across municipalities. Additionally, we delved into Resource Description Framework (RDF) and Web Ontology Language (OWL). To gain practical experience, we spent the first week familiarizing ourselves with Protégé and working on the pizza ontology. For a detailed overview of the technologies we learned during the initial weeks, please refer to section 2.3.

We also addressed project logistics, such as meeting times, meeting locations, communication platforms, and the use of sprints. At first, our meetings were scheduled every weekday from 10.15 to 14.00. However, as we entered the development phase, we chose to cancel our meetings on Mondays and allocate more time on the other weekdays. Our primary objective in doing so was to have longer work days where we were actively programming, reducing “overhead”.

Our client did not have any particular preferences regarding software repositories, as long as we could grant him access to the application either by sharing it on *GitLab* or sending it as a zip file. Since every group member was familiar with GitLab, we decided to use it as our software repository.

[16] https://webprotege.stanford.edu/
[17] https://protege.stanford.edu/

4.6 Development of the product ontology

After conducting initial research and gaining a solid understanding of ontologies, the group split into smaller, specialized sub-groups. One such group, consisting of two members with expertise in ontologies, focused exclusively on this area. This division of labor aimed to enhance the end result and avoid the "jack of all trades, master of none"-problem.

Initially, the ontology group engaged in a brainstorming session to determine the classes and subclasses necessary for categorizing ODA problems within the ontology. The group drew inspiration from Nesta's ODA guide to help identify the appropriate classes. Although identifying the classes was straightforward, implementing the inference to detect potentially similar ODA problems proved challenging.

Throughout the ontology development process, multiple iterations were undertaken. For each new iteration, the group members worked independently before coming together to compare their individual ontologies. This combination of individual and pair programming effectively avoided groupthink, encouraged a diverse range of solutions, and facilitated constructive discussions to identify and address potential weaknesses. Given the potential for cascading errors resulting from a single flaw in the ontology, this aspect was crucial.

Due to the complexity of ontologies and the risk of cascading errors, the customer, an expert in the field, recommended rebuilding the ontology from scratch with each new iteration. While the group found that some iterative development was necessary to save time, major changes always involved reconstructing the ontology from scratch. Additionally, simplified "dummy ontologies" were created for testing new concepts, enabling easier evaluation and integration into the full ontology.

A significant breakthrough occurred with the discovery of the SWRL-plugin[18] (Semantic Web Rule Language), which allows for advanced inferences which can be used for detecting similar ODA problems. The group used this extension to create the first functional ontology draft, primarily following the rule format: *If problem A belongs to category 1 and problem B belongs to category 1, then problem A and problem B are related*. However, after undergoing an extensive overhaul of the ontology and integrating it with SWRL rules, an incompatibility issue arose between the plugin and the GraphDB database. Despite several days of troubleshooting, the group had to abandon the SWRL plugin.

Looking back, it seems evident that conducting more comprehensive research on SWRL's compatibility with GraphDB before incorporating it into the ontology would have been beneficial. Such a proactive approach could have helped the group avoid significant amounts of wasted work. However, this might have been easier said than done. Ontologies are a complex and niche subject, and finding resources online had proved difficult throughout the development. Previously, the group had experienced spending more time reading cryptic documentation to figure out an ontology concept, than what would have been spent learning the concept via hands-on experimentation. Nonetheless, this experience provided valuable insights, and the group will undoubtedly be more thorough in checking for compatibility in future endeavors.

Ultimately, the group discovered a method called “property chaining” which could be used for creating inferences able to detect similar ODA problems. We had previously read about this method during the early stages of ontology development, however, it had been discarded due to our lack of understanding and thus

[18] https://www.w3.org/Submission/SWRL/

was mistakenly believed to be irrelevant. For more details on how this inference works in practice, please refer to *3.5.3 implementation.*

4.7 Development Process

The first part of the project consisted of preliminary research and learning the necessary software. The group familiarized itself with the ontology editor and framework Protégé.

The group planned to use an agile approach to development. Issues that were made from the requirements approved by the customer were placed in a prioritized order in the backlog and Kanban board. To solve these issues the group mostly worked in pairs and did pair programming. For each sprint, there were sprint meetings and retrospective meetings to further facilitate the learning and making improvements.

4.7.1 Sprint 1 (22.02 - 08.03)

Once the group had completed the initial draft of the project ontology and reviewed it with the client, the first sprint commenced.

Sprint 1 - Work

The sprint was divided into three main focus areas.

Firstly, we focused on frontend development, which progressed rapidly, with several pages and elements from the Figma prototype being implemented. When working on the frontend the group added several new issues that arose under development.

Secondly, the group continued to work on the project ontology, making some changes to the first draft by removing redundant classes such as *DefinedAction* and *Municipality*, which were deemed unnecessary and would overcomplicate the inference process. The group also discovered a powerful tool called 'SWRL', which helped to define semantic rules in the ontology and significantly aided inferencing. However, issues arose when exporting the ontology as GraphDB did not support SWRL. Work to convert SWRL rules to another format supported by GraphDB was continued in sprint 2.

Finally, work began on setting up the GraphDB database for local development in Docker.

Sprint 1 - Retrospective

After completing the first sprint, we conducted a retrospective to discuss the positives and negatives. Some of the positive points highlighted were the effectiveness of the group dynamics, work organization, daily standups, and productive discussions with the customer. However, we also identified areas for improvement, such as late arrivals to meetings, group distractions, more relevant development tasks, sharing of knowledge, and updating the timesheet more regularly. We also acknowledged that the report could have been in a better state. Additionally, making issues for the backend prior to implementing the ontology was problematic since two backend issues could not be finished in the sprint. After assessing and voting on the issues raised, the group collectively decided on six action points to prioritize during the upcoming sprint:

1. Change the rules for late arrivals.
2. More clearly defined pauses and work sessions.
3. Remind each other to update their timesheets.
4. Work more regularly with the report. Even more so when you have challenges or solutions from the project to write about.
5. Change the meeting time for Thursday meetings. This was based on group members often having other obligations during these meetings.
6. Group members should get better at speaking out when they have nothing to work on.

The full list of points can be found in *Appendix C*.

Sprint 1 - Backlog

Table 4.1 - Sprint 1 Backlog

Issue #	Issue	Tags
18	Setup database and API endpoint	Backend
19	Connect backend with frontend	Backend, Frontend

4.7.2 Sprint 2 (10.03 - 23.03)

For sprint 2, the main goal was to progress further into the project, mainly starting to shape the backend so it could be connected to the frontend. Finishing the ontology was the most important goal of the sprint, as it was a major hurdle to create a proper backend. Members found out that the ontology needed a class like *DefinedAction* that was removed from the first draft in sprint 1 and thus reimplemented it. Another goal was performing a major restructuring of the project report as it was not in line with the syllabus guidelines.

Sprint 2 - work

The group commenced work on the project's backend, which did not require inferencing to be fully functional for implementing most of the queries. Fortunately, the ontology group discovered “property chaining” which made inferencing functional with GraphDB. This was a major boost for the group, as this had been a major hurdle for our proof of concept.

The general structure of the backend was then decided, and considerable progress was made on the needed queries. Although we managed to progress far in this aspect, we did not have time to connect it to the frontend in any significant way. With regards to the frontend, most of the main pages had already been created, so our main focus was on minor revisions and changes.

For the report, we restructured the format to be more clearly divided into a preliminary research part, a product part, and a process part.

Sprint 2 - Retrospective

The group identified several positive aspects of the second sprint, such as continued good collaboration, the benefits of changed meeting times, and the effectiveness of specified pauses. However, the group also acknowledged the need for more quality control of code merged into the main project branch and the importance of daily standups, which had fallen out of routine during the last week of the sprint. The complete list of points can be found in *Appendix C*. The retrospective resulted in several action points, including restructuring some roles, changing break routines, and assigning some members to a "facilitator" type role. The group unanimously agreed to appoint Erland as the facilitator.

Sprint 2 - Backlog

Table 4.2 - Sprint 2 Backlog

Issue #	Issue	Tags
19	Connect backend with frontend	Backend, Frontend
34	Update frontend based on new ontology	Frontend

4.7.3 Sprint 3 (24.03 - 21.04)

For sprint 3, the goal was to further connect the backend with the frontend, along with adding the remaining critical functionality in the frontend. This sprint was four weeks long because of the yearly one-week excursion, as well as the easter holiday. The excursion started on the 25th of March, so the first day of the sprint was spent discussing what the group should do when everybody would be back on the 10th of April.

Sprint 3 - Work

After completing the majority of the frontend in sprint 2, the group shifted focus to testing the application. Unit tests and Snapshot tests were created and finished during this sprint, and work started on end-to-end testing.

The group achieved substantial progress on the backend, having completed most of the routing and querying. We also successfully connected the backend and frontend to create a fully functional application, with only minor tasks remaining to be completed in Sprint 4.

The group ramped up work on the report compared to earlier sprints, with several members dedicating their efforts solely to it. As a result, the report's progression sped up significantly.

Toward the end of the sprint, the group presented and demonstrated the product to Trondheim municipality, along with other TRDK groups. The presentation was well-received, and the group received high praise and quality feedback for their work.

Sprint 3 - Retrospective

In the third retrospective, the group agreed that the project had progressed well and that communication within the group had been effective. However, the facilitator became ill for a week due to an infection, but the group was able to adapt and maintain their workflow.

The group had observed a recurring trend of members arriving late, with delays ranging from 5 to 15 minutes. Moreover, the group encountered challenges in establishing a consistent break schedule, which resulted in misaligned breaks that led to distractions and reduced overall productivity. Daily standups were also disrupted during the sprint due to the absence of the facilitator. However, since most of the group's work was done during physical meetings, there was a low threshold to ask what people were working on.

The group identified two important action points during the retrospective: firstly, to arrive at meetings before their scheduled start time to ensure promptness and efficient use of time; and secondly, to prioritize taking consistent breaks during work sessions, particularly for report writers who were experiencing fatigue and reduced productivity due to long and unfocused work sessions.

The full list of action points of the retrospective can be found in Appendix C.

Sprint 3 - Backlog

Table 4.3 - Sprint 3 Backlog

Issue #	Issue	Tags
21	Create Admin panel	Frontend
22	User-testing of prototype	Frontend
23	Ensure UI meets WCAG 2.1 standard	Frontend
45	Continuous refresh in “Inspect Problem”	Bug
76	ODAproblem query length	Backend
77	Approve problems and show similar problems	frontend + backend
78	cypress test	Frontend

4.7.4 Sprint 4 (25.04 - 08.05)

Sprint 4 aimed to accomplish several objectives, including creating an admin panel, staging ODA problems, displaying similar ODA problems, resolving encountered bugs, cleaning up code, and finalizing application testing. Most of the group members worked on the report in the sprint, which resulted in large leaps of progress.

Sprint 4 - Work

The group focused on resolving the remaining frontend issues and dedicated a significant portion of sprint 4 to fixing bugs of varying degrees of severity that were identified during development and testing. The most noteworthy bug discovered was that ODA problems could not share the same title. The group fixed these issues and also refactored the codebase. Furthermore, we updated the tests to ensure that the new implementations were working as expected.

Sprint 4 - Retrospective

In the fourth and final retrospective, the group expressed overall positivity towards the sprint. We unanimously agreed that the product looked impressive and felt that the project had come together nicely.

The motivation in the group was elevated after the validation test and all the positive feedback from the customer. However, the group acknowledged that they could have been more supportive in updating the end-to-end tests. This task was assigned to only one member, who had a relatively heavy workload.

Sprint 4 - Backlog

Table 4.4 - Sprint 4 Backlog

Issue #	Issue	Tags
23	Ensure the UI meets the WCAG 2.1 standard	Frontend

5. Summary

This chapter presents a summary of the report and its key findings. As per the *project's vision and goals*, the group collaborated with Trondheim Municipality to develop a prototype for systematizing and storing recorded municipal data problems and facilitating cooperation between municipalities, utilizing ontologies and the ODA method.

As the group was unfamiliar with these topics and technologies, an extensive *preliminary research phase* was conducted. During this month-long endeavor, the group gained proficiency in *ontologies* and a solid understanding of the *ODA method* which is a specification for how municipal data problems should be formulated and recorded. Moreover, the group became familiar with the *semantic web*, and *inferencing*. Other

representation methods, including *taxonomies* and graph databases, were also researched during this phase. However, the customer's strong preference for ontologies became evident after evaluating all options.

Formal *project requirements* were quickly established to gain a clearer understanding of what functionalities would need to be implemented and prioritized during product development. The technology stack was also decided, relying on the popular and well-supported *React* library and *Typescript* language for the frontend development. React was chosen because of its popularity, customization options, the group's previous experience with it, and because it was already being used by Trondheim municipality. Typescript was chosen as its static typing enhances the debugging and testing experience.

RDF4J Java was intended for the project's *backend* to extract information from *Ontotext GraphDB*, but *TypeScript* and *SPARQL* were eventually chosen to connect the backend and database through the *RDF4J REST API*, allowing highly customizable queries with less boilerplate code. *Graphdb.js* was considered but deemed too general. Fewer dependencies were preferred to simplify future updates. The backend was connected to the frontend using the widely-used *Express* framework, allowing for streamlined API creation. Using TypeScript in the backend allowed for a consistent programming language across the frontend and backend, facilitating collaboration among group members. *GraphDB* was chosen as the database management system for the project due to the customer's specific requirement to use it.

The architecture of the prototype was developed and is described in *section 3.3* using the 4 + 1 architecture model.

Both manual and automated testing techniques were utilized to ensure the quality of the product, with manual testing being conducted initially and automated testing tools like *Jest* and *Cypress* being used later. To evaluate the application's usability, a *validation test* was performed with test subjects being facilitated by Trondheim municipality.

To put emphasis on the end user, *user stories* were developed based on the functional requirements previously mentioned. A *Figma prototype* was created to visualize the user interface without writing code, saving time and providing a common understanding of the expected interface for the group and customer.

Following this initial work, the actual product was developed. For a full specification of implementation details, check *section 3.5.3*. As this is a prototype, the product is not fully developed, and *section 3.6* on future work outlines the necessary steps to implement a production version of the product.

The success of the project was greatly attributed to the group’s effective organization and collaboration. As the group had previous experience with agile development from earlier project work, we all had a common understanding of what methods worked and decided on several *agile practices* to be part of the process. A *project plan* was also established, and a *group contract* was signed to clarify expectations, improve accountability, and enhance productivity.

The group conducted a *risk assessment* to identify potential issues with the product or group and assigned probability and magnitude levels to each. This allowed us to create a visual 4x4 risk matrix with color coding to represent the risk levels. The group also developed measures to reduce the probability or consequences of each risk and created an updated risk matrix.

After conducting initial research and gaining a solid understanding of ontologies and the ODA method, the group reorganized into specialized sub-groups. The group was divided into three groups: frontend, backend/database, and ontology, with group members moving between them as needed during each sprint. This approach aimed to enhance the end result by avoiding the common issue of spreading oneself too thin and not mastering any one skill, which can happen when taking a less specialized approach. After two sprints the group appointed a dedicated facilitator.

The customer was contacted early on in the project, and weekly meetings were scheduled to keep him updated on the project's progress. Communication outside of meetings was established on *Slack*. Regular *communication with the customer* allowed for feedback on the product, and identification of potential issues, and encouraged proactive problem-solving to prevent major roadblocks during the development process.

The group communicated with the supervisor through emails and meetings every other week, which focused on the report as there were no major issues with group collaboration. The supervisor's feedback helped the group refine the report's structure and emphasize its important aspects.

To maintain a common *coding standard*, the group implemented *ESlint* for linting and *Prettier* for code formatting. Git usage standards were also established to maintain code quality and peer review, disallowing commits to the main branch and requiring merge requests to be approved by at least one group member. The group adopted a naming scheme for GitLab branches and issues to improve clarity. Although strict typing and coding practices minimized the need for documentation, the group agreed to comment on ambiguous code parts. These standardizations, some of which group members were already familiar with, helped accelerate development.

The group split into sub-groups during development, with one group focused exclusively on the *development of the ontology*. The ontology underwent multiple iterations, and a combination of individual and pair programming was used throughout this development. The group encountered incompatibility issues, but eventually discovered a method called "property chaining" which made it possible to have inferences that detect similar ODA problems.

The product development was divided into four sprints, which you can read more about in *chapter 4.7*.

In conclusion, the prototype that was developed went above what the group initially expected. The final product contains an extensive array of features that would not have been implemented without the group's extraordinary collaboration effort. The group is thankful that the product may be developed further into a production version, and has benefited greatly from the learning outcome of working with a customer-oriented project.

Bibliography

1. *16th Annual State of Agile Report.* (n.d.). Retrieved February 9, 2023, from https://stateofagile.com/#ufh-c-7027494-state-of-agile
2. Allemang, D., & Hendler, J. A. (2009). *Semantic web for the working ontologist: Modeling in RDF, RDFS and OWL* (1nd ed). Morgan Kaufmann.
3. Copeland, E., Symons, T., Simpson, H., & Dragicvic, N. (n.d.). *Public Sector Data Analytics—A Nesta Guide*. https://media.nesta.org.uk/documents/Public_Sector_Data_Analytics_-_A_Nesta_Guide_byCwKTI.pdf
4. Cypress. (n.d.). *Open-Source JavaScript Test Runner | cypress.io testing tools*. Retrieved April 14, 2023, from https://www.cypress.io/app/#e2e
5. Definition of Done. (2015, December 17). *Agile Alliance |*. https://www.agilealliance.org/glossary/definition-of-done/
6. Dingsøyr, T., Dybå, T., & Moe, N. B. (2010). *Agile Software Development: Current Research and Future Directions* (1st ed.). Springer Science & Business Media.
7. Docker. (2021, November 11). *What is a Container? | Docker*. https://www.docker.com/resources/what-container/
8. Eaton, M., & Bertoncin, C. (n.d.). *State of Offices of Data Analytics (ODA) in the UK*.
9. Eclipse, R. (n.d.). *The Eclipse RDF4J Framework · Eclipse RDF4J™ | The Eclipse Foundation*. Eclipse RDF4J. Retrieved March 1, 2023, from https://rdf4j.org/about/
10. Ehimwenma, K. E., Crowther, P., Beer, M., & Al-Sharji, S. (2020). An SQL Domain Ontology Learning for Analyzing Hierarchies of Structures in Pre-Learning Assessment Agents. *SN Computer Science*, *1*(6), 335. https://doi.org/10.1007/s42979-020-00338-1
11. *Express—Node.js web application framework*. (n.d.). Express. Retrieved March 14, 2023, from https://expressjs.com/
12. Gruber, T. R. (1995). Toward principles for the design of ontologies used for knowledge sharing?

International Journal of Human-Computer Studies, *43*(5), 907–928. https://doi.org/10.1006/ijhc.1995.1081

13. Henry, S. (2023, April 19). *WCAG 2 Overview*. https://www.w3.org/WAI/standards-guidelines/wcag/#top
14. *JSDoc: Home*. (n.d.). Retrieved April 28, 2023, from https://ontotext-ad.github.io/graphdb.js/
15. Kniberg, H. (2015). *Scrum and XP from the Trenches* (2nd ed.). lulu.com.
16. Kruchten, P. B. (1995). The 4+1 View Model of architecture. *IEEE Software*, *12*(6), 42–50. https://doi.org/10.1109/52.469759
17. Noy, N. F., & McGuinness, D. L. (2001). *Ontology development 101: A guide to creating your first ontology*.
18. *OWL 2 Web Ontology Language Primer (Second Edition)*. (2012, November 12). https://www.w3.org/TR/2012/REC-owl2-primer-20121211/
19. *RD4J REST API - OpenAPI spec*. (n.d.). Retrieved April 28, 2023, from https://rdf4j.org/documentation/reference/rest-api/
20. solid IT. (n.d.). *DB-Engines Ranking of RDF Stores*. DB-Engines. Retrieved January 31, 2023, from https://db-engines.com/en/ranking/rdf+store
21. Solms, F. (2012). What is Software Architecture? *Proceedings of the South African Institute for Computer Scientists and Information Technologists Conference*, 363–373. https://doi.org/10.1145/2389836.2389879
22. UserTesting. (n.d.). *When Should You Do Usability Testing?* UserTesting. Retrieved April 20, 2023, from https://www.usertesting.com/resources/topics/when-should-you-do-usability-testing
23. Uutilsynet. (n.d.). *WCAG-standarden*. Retrieved April 10, 2023, from https://www.uutilsynet.no/wcag-standarden/wcag-standarden/86

Appendix

Appendix A

Figma prototype

[Link to Figma prototype]

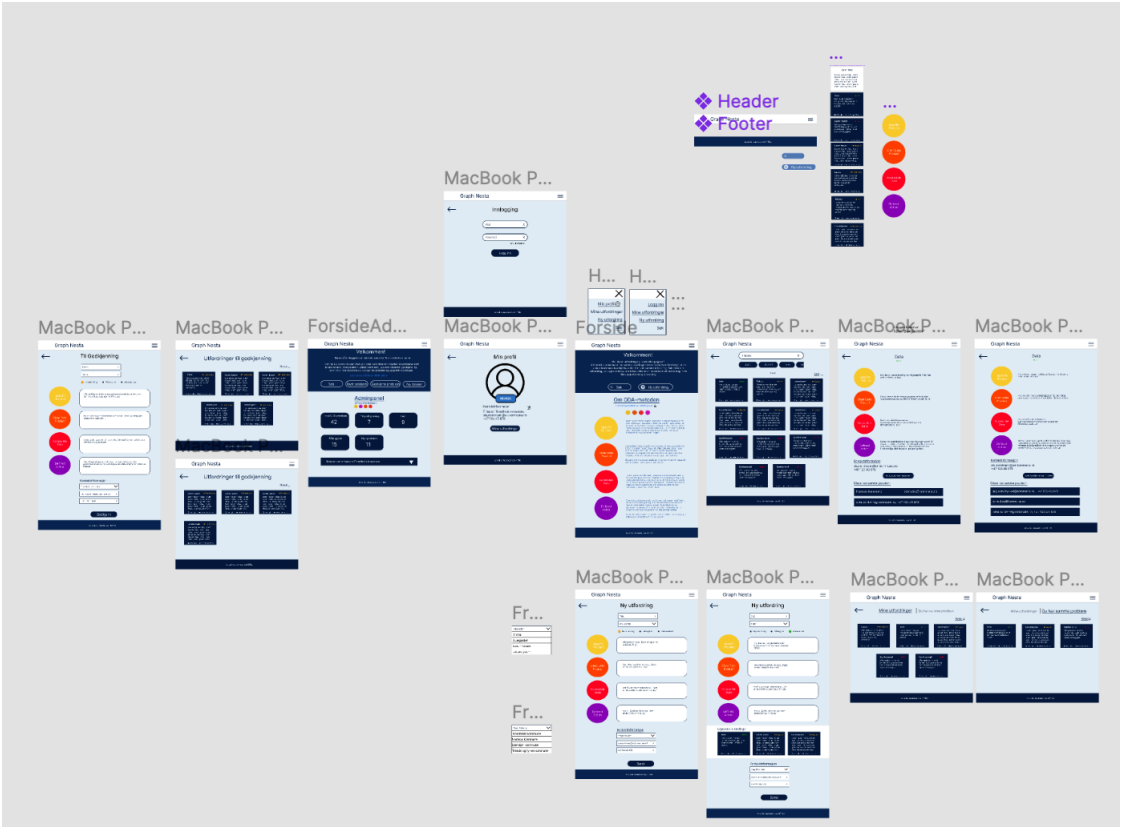
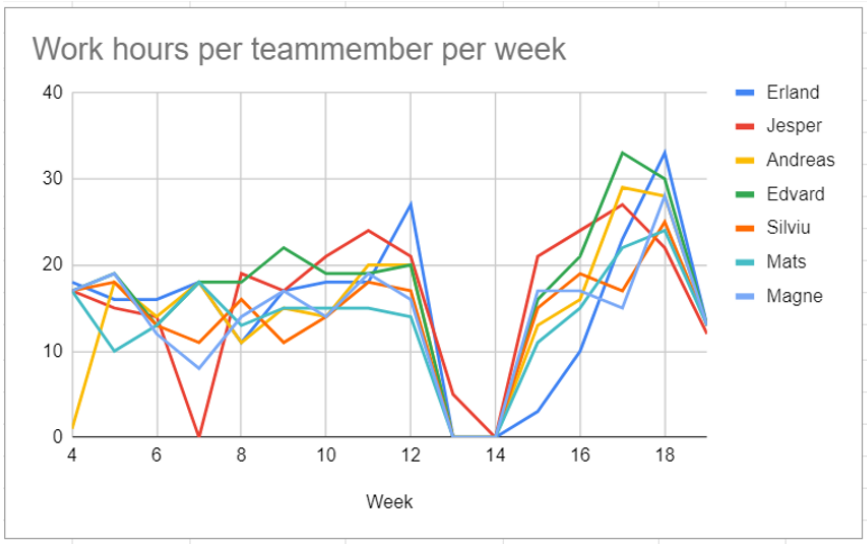


Figure A.1 - An overview of the Figma prototype

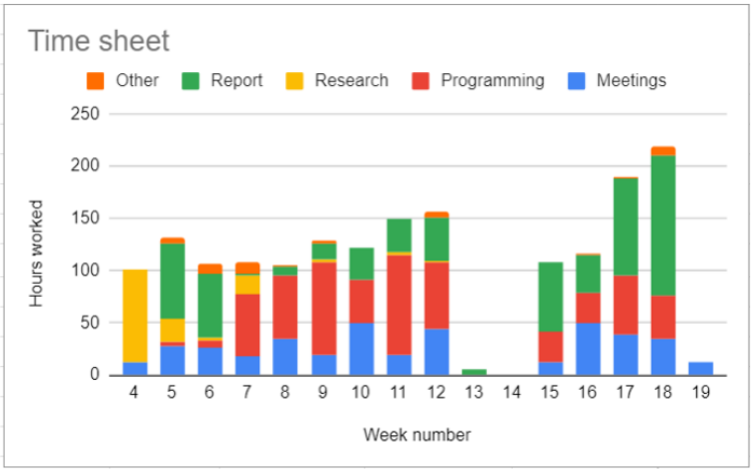
Appendix B

Timesheet



Name	Total hours
Erland	239
Jesper	258
Andreas	228
Edvard	276
Silviu	225
Mats	215
Magne	227

Figure B.1 - A graph showing total hours worked each week, and a table showing total hours worked by each team member



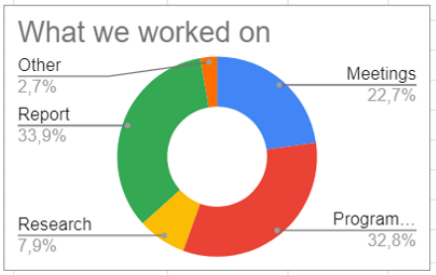


Figure B.2 - A graph depicting the amount of work completed by category for each week of the project, accompanied by a donut chart showing the percentage of work by categories

Group Contract TRDK1_Graph-NESTA

The contract applies to members of the TRDK1_Graph-Nesta group. The members are: Andreas Ringereide Berg, Jesper Barfod, Silviu Catalin Mihai, Edvard Bjørnevik, Erland Lie Amundgård, Mats Thoresen Nylander and Magne Slåtsveen.

Working hours:

The Group has working hours, which are defined as core time for the project, on the following days and hours:

- **Tuesday**:
  - 10.15 - 14 (A4-132)
  - 12.00 - 13.00 (Supervisor meeting, IT-bygget meeting room 122)
  - 14 - 17 (VM-paviljongen)
- **Wednesday**:
  - 12.30 - 14.00 (Trondheim Torg)
  - 13.15 - 17.00 (Kalvskinnet)
- **Thursday**:
  - 12.15 - 15.00 (Kalvskinnet)
- **Friday**:
  - 10.15 - 13.00 (A4-138)
  - 13.00 - 14.00 (VM-paviljongen)

It is expected that each group member works approximately in total **17.75 hours per week**.

Penalties

The group agrees on a system of penalties to ensure that each member complies with the expectations for work and reliability (for meetings and work) that one should expect from a member of a project of this scale.

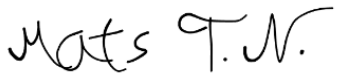
Rules:

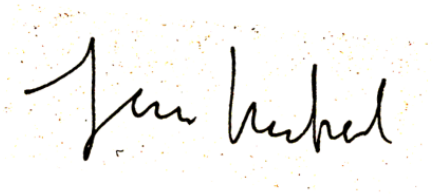
- 5 minutes delayed - 1 penalty
  - Every 10 minutes after - 1 penalty
- A Maximum of 4 penalties per day
- 1 day/afternoon advance notice when due for a meeting/work gives no penalties.
  - Shorter notice - 2 penalties
- Notification of late arrival (missed the bus, etc.) - 1 penalty

Penalty values:

- 1 penalty equals 1 beer in value (30 Norwegian kroner)
- 4 penalties equals one homemade cake

Signatures:





Appendix C

Risk Assessment

For a more detailed and higher resolution image, check this link

ID	Risk description Keywords about: (1) initial events (2) the information security breach (3) the undesirable consequences that may arise	Reason for impact assesment	Consequence	Probability	Reason for assesment of probability	Risk level	Assesment of risk management	Consequence after measures	Probability after measures	Residual risk
R1	Risk 1 example: Team member drops the course	Important parts of the project can become missing with the team member.	High	Mode rate	NN has been unsure with finishing the project.	Moderat e	NN has promised to save NN´s part of the project on a shared project area and will notice the team far in advance before dropping out.	Mode rate	Mode rate	Moderat e
R2	Unfair work distribution	Some members may end up doing too much work, or some may not be sufficiently involved.	High	Low	It seems that all members are equally motivated to work.	Moderat e	Important to discuss such cases in the team. Find out what is causing the uneven workload, and eventually an agreement that suits everyone	High	Low	Moderat e
R3	Disagreement	We are a large group, where everyone have their own opinions and preferences. These may collide, and cause decision making to take longer than necessary.	Mode rate	Low	Everyone in the group get along, and respect each others opinions and preferences.	Low	We can make decisions through voting, which works well, seeing as the number of group members is uneven.	Low	Low	Low
R4	Availability	Can slow the flow of the development. If, for instance, a member is unavailable for a certain amount of time and other members are dependent on that person in order to move forward.	Mode rate	High	All team members have other courses and outside obligations to attend to. Not every members will be available every day.	Moderat e	Agree on a flexible plan that allows members to attend to their obligations without having to slow down the development process of the project. Have team members notify the rest of the group in a timely manner whenever an absence may occur.	Low	High	Moderat e
R5	Uneven motivation	Affects the commitment each members has in the project and thus the work they put in. This can lead to some members being overworked	Mode rate	Low	The members of the group have similar ambitions regarding this project. This can allways change in the future but as of now the group is balanced	Low	In the event a team member decided that he wants to put less effort into the project then a meeting will have to be scheduled where the team and the member agree on a workload that suits both parts.	Low	Low	Low
R6	Group member only working alone and misinterprets current issue/task	Risk that one or more members of the group start working with something that opposes what the rest of the group is doing . This may result in git problems, having to redo parts of the project or code that mismatches the rest.	High	Mode rate	Some members of the team prefer to work alone	Moderat e	Plan as a group and keep each other informed on where we stand, e.g., using sprint planning, daily standup	Mode rate	Low	Low
R7	Group member leaves the project	Would change the group dynamics, and require restructuring of our work	High	Low	All group members are seemingly highly motivated for the course, and are all equally participating in the group work. Likely factors for a group member would rather be an external unexpected factor.	Moderat e	Have to agree on how to distribute the workload and fill holes in the work. Mostly based on which role the leaving group member had	Mode rate	Low	Low

Figure C.1 -Risk Assessment Table Part 1 of 2

R8	Communication with the customer	The customer works full time at Trondheim kommune, and could be very busy from time to time.	Low	Moderate	We can't expect the customer to always be available for us. This is something we have to expect and plan around	Low	It doesn't seem to be a big issue. So far it has been easy to communicate with the customer. In the case that he is not available, it should be easy to plan around it	Low	Low	Low
R9	Fluid requirements and expectations	The customer has given the group freedom to chose the requirements for the project. These requirements will take time to develop and most likely will have to be changed/adapted as the project advances	High	Moderate	As there are many approaches the group can take, in order to find the right one the group will have to test its way forward. This will include many trial and error cases.	Moderate	The group will try to maintain good communication with the customer, as well as work consistently with trying different approaches early on in the development. This will help us chose solid requirements from the start as well as giving us time to change things if needed	Moderate	Low	Low
R10	Constant changes	To develop an ontology is a constant and time-consuming process. The ontology will most likely need to be changes and/or updated as more requirements are added/changed to the project	Moderate	High	It is very unlikely that the group designs the perfect ontology in the first try. The process will involved a lot of changes and trial and error as the the task is very open, with multiple posible solutions.	Moderate	Work consistently to ensure that we have enough time to fix problems that may arise during the development	Moderate	Moderate	Moderate
R11	Getting stuck on creating ontologies - problems	Designing an ontology is going to be a difficult task considering it will be the first time we have touched on the concept and related technologies, as well as it just being difficult to implement.	Very High	High	Creating our first ontology will be a difficult task with a lot of incremental improvement. It could also end up being a lot of trial-and-error as there can be a lot of pitfalls when designing one.	Very High	Careful planning, lots of discussion about the implementation, and a good understanding of ontology implementations will help reduce the risk of not being able to deliver on the ontologies	Very High	Moderate	High
R12	Equipment failure - A team member's Personal Computer stops working	There is always a chance that a computer stops working, either through physical damage, extensive use or bad luck.	Moderate	Low	All members use relatively new equipment, which they treat carefully.	Low	Equipment failure will require reparation or replacement. This may cause a team member to be without a working computer. The consequence of this event is moderate, as it may slow down the development process, or cause loss of unsaved code.	Low	Low	Low
R13	Remote work and communication / sickness	Some team members may hang behind the rest. The group has chosen to work together i the same room. If one team members can not physically attend, then that person may lose some important information discussed in the meeting. Having one team member attend the meeting over the internett while the rest	High	High	As the group has a considerable size, the probability that some member is unable to attend the meeting physically, eg. he is sick or is traveling, and has to work remotely is quite high.	High	The team member that could not attend can ask to get briefed on important events or discussions that happened during the meting. The daily standup is a good time for that.	High	Low	Moderate

Figure C.2 - Risk Assessment Table Part 2 of 2

Appendix D

This appendix covers the retrospective boards that were produced during the retrospective meetings that the group had. Sticky notes that overlap indicate that they share a common subject, e.g., ontology-development. There are also links to each board for easier readability.

Sprint 1 retrospective board - Link to Sprint 1 board

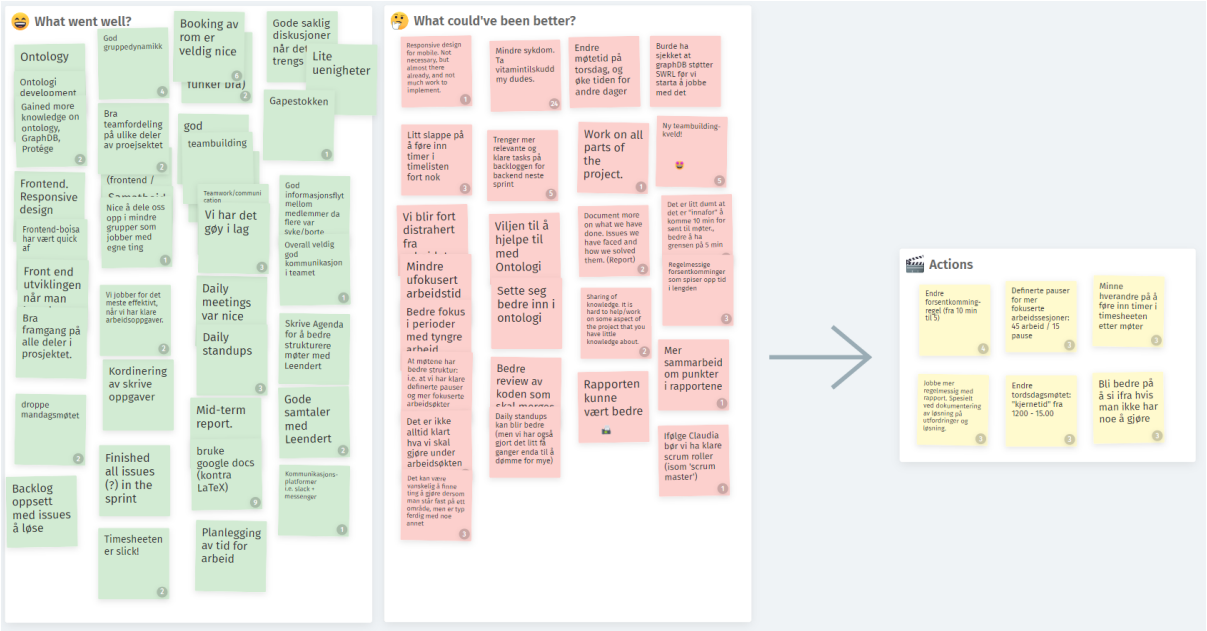


Figure D.1 - Retrospective Board for Sprint 1

Sprint 2 retrospective board - Link to Sprint 2 board

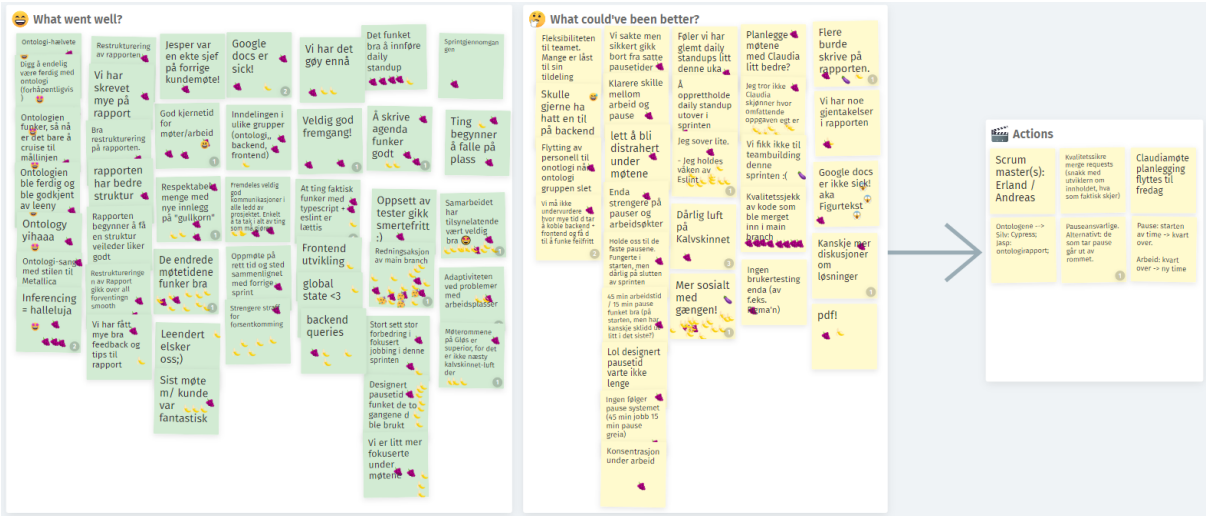


Figure D.2 - Retrospective Board for Sprint 2

Sprint 3 retrospective board - Link to Sprint 3 board

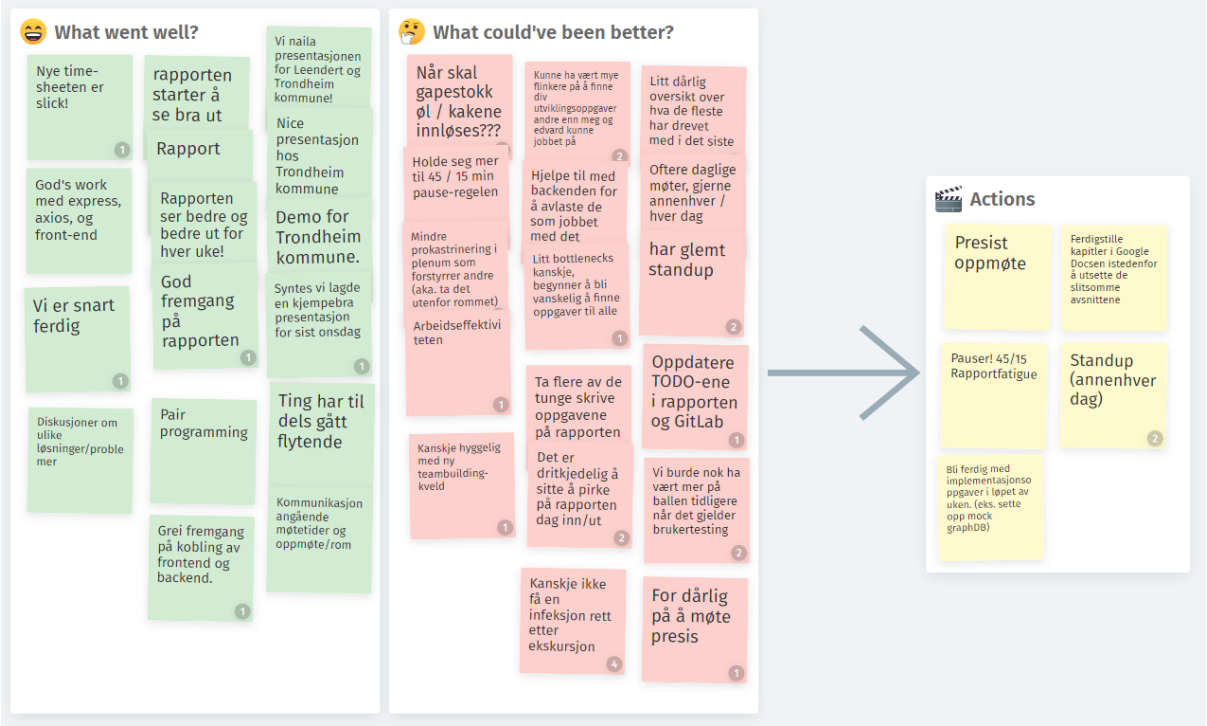


Figure D.3 - Retrospective Board for Sprint 3

Sprint 4 retrospective board - Link to Sprint 4 board

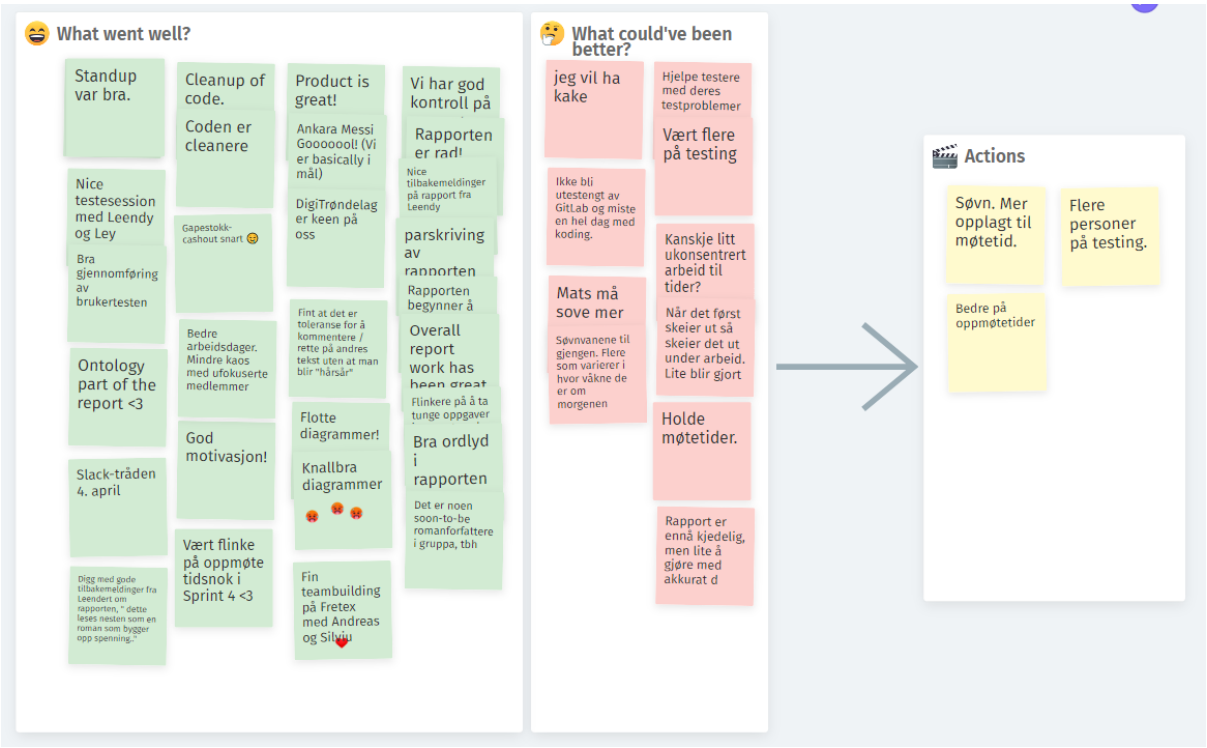


Figure D.4 - Retrospective Board for Sprint 4

Appendix E

Unit Testing and Snapshot Testing Results

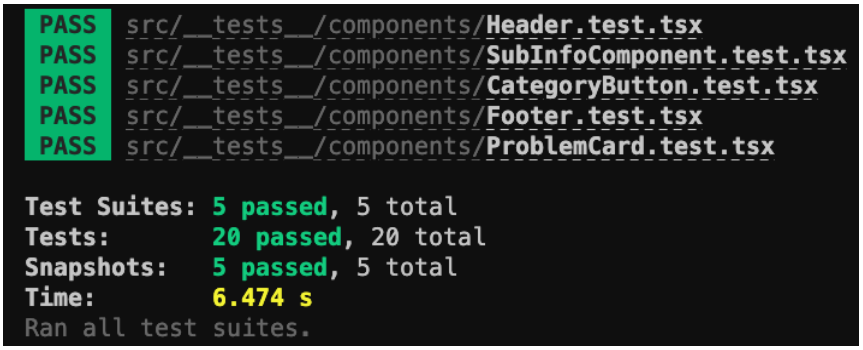


Figure E.1 - Unit Test and Snapshot Test Results.

End-To-End Testing Results

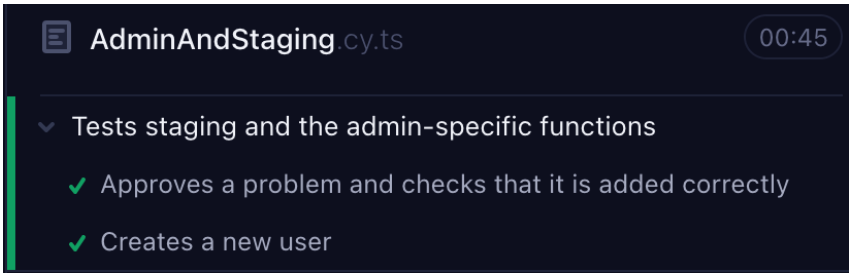


Figure E.2 - E2E Admin and Staging Test Result.

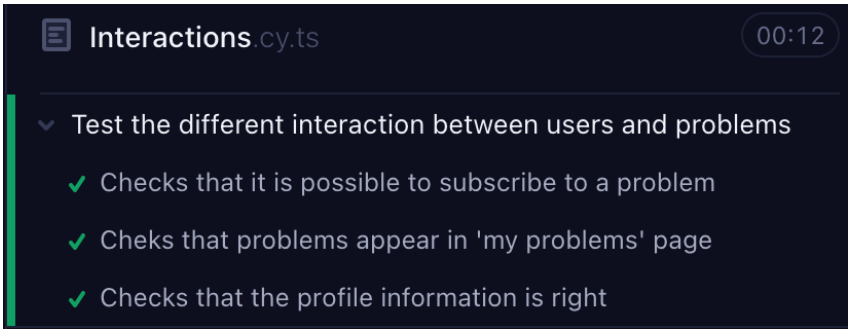


Figure E.3 - E2E Interaction Test Resul

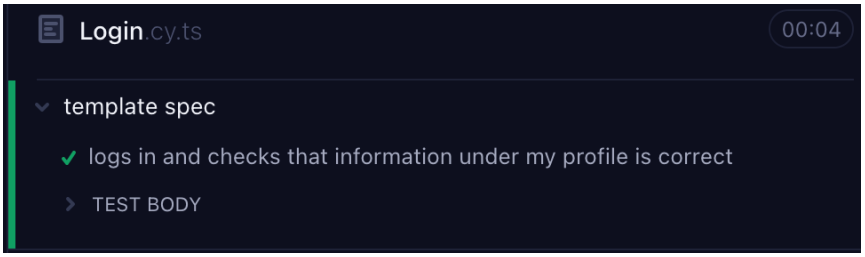


Figure E.3 - E2E Login Test Result

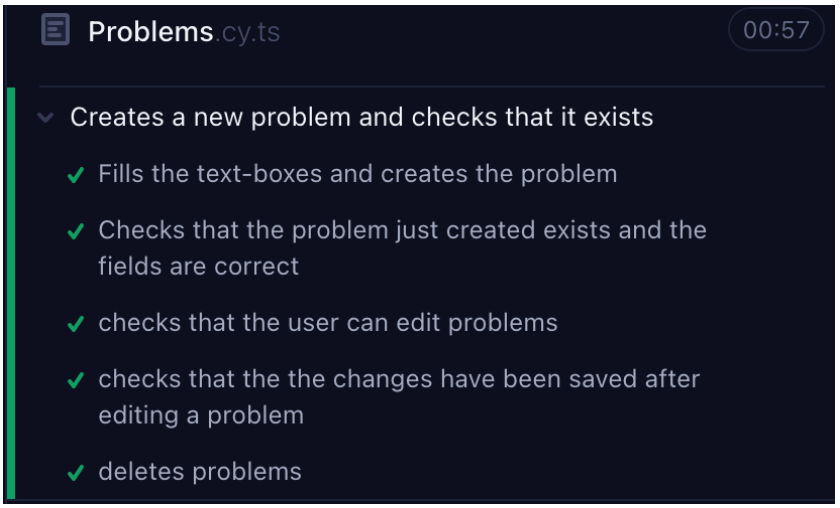


Figure E.4 - E2E Problems Test Result.