Group 5

TDT4290 Customer Driven Project

Development of a Data Integration Tool for a Mobility-as-a-Service (MaaS) Platform

Authors
Eivind Yu Nilsen
Elise Nordal
Markus Lossius Opdahl
Eivind Aksnes Rebnord
Regine Ruud
Åsne Stige
Sebastian Vittersø

Supervisor
Yujie Xing
Customer
SINTEF Digital

November 19, 2020

Executive Summary

This report describes the process of developing a software solution for SINTEF Digital during the Customer Driven Project course at NTNU. SINTEF Digital approached NTNU with a project that was expected to deliver a data integration tool for a Mobility-as-a-Service (MaaS) platform. With the goal of offering tailor-made mobility solutions for individuals MaaS represents the future of transportation. MaaS is depending on data integration to efficiently combine data from different platforms. The project was assigned to group 5, hereafter referred to as "the team". The report presents the planning, research, process, and methods practiced by the team.

The project involved an extensive phase with preliminary studies. The team needed to familiarize themselves with the concepts of MaaS to understand what functionalities the data integration tool should have. The team had to understand how to use the existing source code developed by the customer, in the form of several algorithms used for semantic matching to be able to specify the requirements of the solution. Implemented correctly, a data integration tool will save data providers countless hours of manual work needed to establish MaaS. This report documents how the team analyzed the current situation and the solution space and gives a justification for the architecture and technologies used in the choice of solution.

The report also describes the team's implementation of Scrum as the development methodology of choice. As a new team, the defined framework of the methodology helped to establish an effective work environment and resulted in effortless cooperation that kept improving throughout the project. Scrum was used to manage the process, to make it as time effective and quality-oriented as possible. The team had regular meetings internally and together with the supervisor and the customer. These meetings ensured that all stakeholders were updated on the progress and facilitated for a good relationship between the team and the customer throughout the project. Following the Scrum methodology made it easy to keep an overview of the project, and to adjust the requirements and expectations along the way in cooperation with the customer.

By the end of the course, the team accomplished all of the goals within the scope of the project, and the customer was satisfied with the solution and process. The final product resulted in a data integration tool in the form of a web-application, extending the existing source code developed by the customer in Java, and linking it to a single-page application framework with an API. The final product fulfilled the requirements specified by the customer.

Contents

1 Introduction

This report is documentation of a customer driven project at NTNU during the fall of 2020. It describes the given problem and the proposed solution, as well as other relevant aspects regarding the process or the solution.

1.1 The Customer

The project is owned by SINTEF Digital, represented by Audun Vennesland, hereafter referred to as the customer. SINTEF is a research organization based in Norway and carries out multiple projects for their customers each year. The given project is part of a bigger collaboration called ReiseNavet and is funded by the National Research Council.

1.2 The Background for the Project

Recently, the transport industry has undergone a pronounced technological transformation. This has brought forward new modes of transportation, like car sharing and micro mobility services such as electric scooters. This transformation has also affected the more traditional modes of transportation, of which providers now have a considerable online presence. A byproduct of these technological changes is an increasing amount of data being available online, which is a potential source for future innovation in the transport industry. This project seeks to investigate how a digital Mobility-as-a-Service (MaaS) platform can facilitate the implementation of MaaS in Norway. Briefly explained, MaaS implies the integration of transport services of various forms into a single mobility service. The MaaS service ReiseNavet investigates represents a subscription service for door-to-door travel using various means of transport offered by different transport providers.

1.3 The Task

Such a platform requires an integration of data from several data providers. Each of those might describe their data using different syntax and semantics, which makes it challenging to create a platform that is compatible with all providers' platforms. Before integration can occur, the data formats used by the data providers must be semantically aligned with those used by the MaaS platform. E.g. if the public transport service in Oslo, Ruter, calls a bus stop "bus stop" while Entur calls it "stop place", an alignment between them will need to be made before the data provided by Ruter can be added to the MaaS platform. The aim of this project is therefore to develop a data integration tool that in an automated or semi-automated manner can facilitate that process. This will help the data providers when integrating with Entur. The proposed solution serves as a prototype, and the goal is to show that the aforementioned task can be solved by offering a user-friendly data integration tool.

The team was provided with an existing repository of source code containing a variation of matching algorithms. as implementing these were not part of the problem definition. In addition to creating a graphical user interface and a data access layer, the project involved an improvement of the existing code, especially problems related to the running time of the algorithms. To implement the solution Javascript and Vue was chosen for the front-end and Java with Javalin for the back-end. The solution makes it possible to match ontologies from multiple file formats and get a result containing the relation and confidence interval. An ontology is a way of showing the properties of a domain and how they are related, by

defining a set of concepts and categories that represent the subject [1].

1.4 The Stakeholders

To make sure the team prioritizes correctly and takes care of the interests of everyone involved, it is important to identify the stakeholders of the project.

The first stakeholder in this project is the team itself. The focus of the team is to learn as much as possible, and deliver a good product that satisfies the other stakeholders' requests.

The supervisor and course administration are also stakeholders whose interests are to see the team get relevant experience from the project, and ensure that the project is a satisfactory experience for both the team and the customer.

The customer, SINTEF Digital, represents a third stakeholder. The customer's concerns of the project are for the team to develop a solution that fulfills all requirements. The customer is also the project leader in Reisenavet and is responsible for the development of solutions for MaaS. The project owner of Reisenavet is Entur, which is a company owned by the Norwegian government and one of the first MaaS providers in Norway. Entur is currently developing a digital platform that offers transportation services from numerous transport service providers in Norway. Entur is therefore another stakeholder of the project and has interest in using the solution developed by the team to make data integration more efficient.

1.5 The Solution

The deployed webpage is located at `http://dataintegrasjon.reisenavet.no/`. The code and wiki can be found at `https://github.com/Kundestyrt-ReiseNavet`. The installation guides can be found in Appendix A.1, together with user guide and what file formats are allowed in Appendix A.2 and A.3.

2 Planning

When working together in a new team where every team member has with different time schedules, it is important to make a clear and well organized plan for the project. After grasping the extent of the project, the team set up a schedule, defined internal routines, created templates, and decided on resources for different purposes. This section presents the planning phase of the project and gives an overview of the planned routines to ensure high quality of the worked produced.

2.1 Project Schedule

In cooperation with the supervisor, a timeline was defined for the project. As there was decided on an agile development methodology, presented in Section 4, the project was taken on in sprint iterations.The team decided to break all project work into sprints, regardless of what kind of work would be conducted in that iteration. The breakdown of the project work resulted in a total of five sprints, opposed to the typical two or three sprints suggested for the project. This enabled the team to get familiarized with the development methodology before embarking on the development sprints, and by that maximize the possibility to improve the collaboration through retrospective meetings. It also ensured consistency in the communication with the customer and facilitated a good framework for the project from day one. The first sprint is called Sprint 0 due to it not being a phase of implementation. The team planned to use six weeks to implement the system, while documentation of the process was planned to be done throughout the whole project to assure correct details in the documentation. Figure 1 shows an overview of the initial project timeline. Appendix B.4 shows a more detailed schedule of the project that was used by the team to plan meetings, set deadlines, and generally keep an overview of the project.

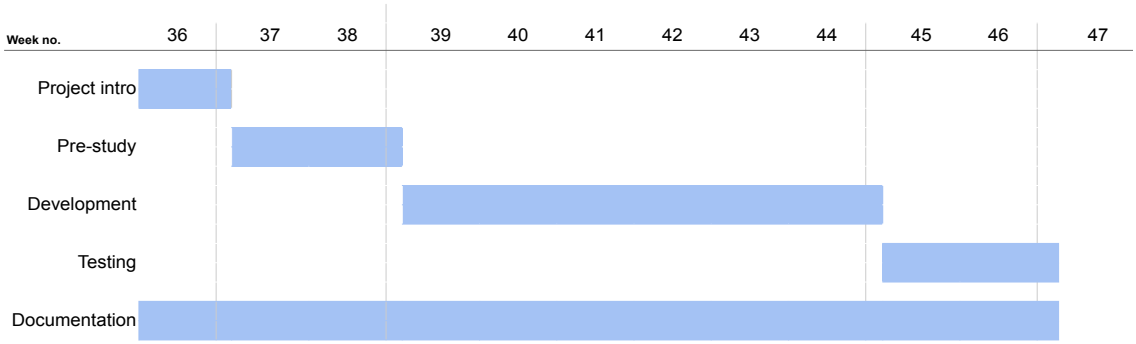


Figure 1: An overview of the project timeline.

The work hours were divided into four different categories in addition to meetings and lectures; architecture, development, report, and project planning. Figure 2 presents the planned distribution of work hours for the team, based on the schedule and the product backlog, which is presented in Section 4.2.1. The figure will be used to compare the planned distribution of work hours to the actual distribution registered at the end of the project, which is further described in Section 7.6.

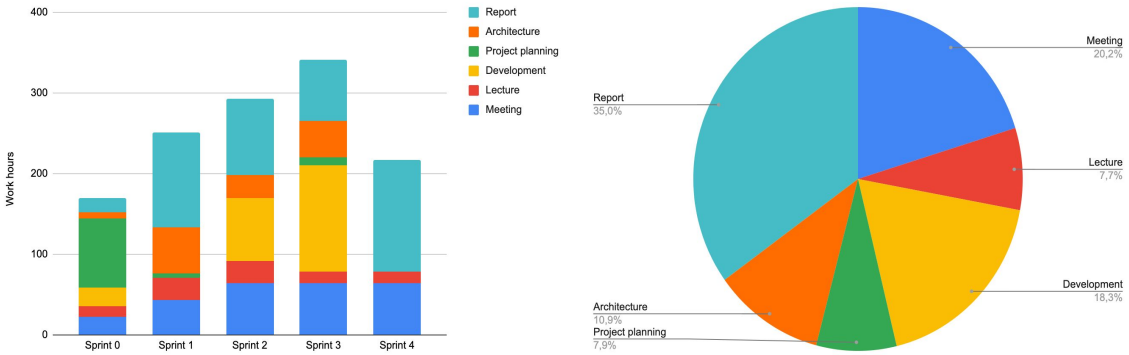


Figure 2: Planned distribution of work hours.

2.2 Team Organization

As a starting point of the project collaboration, the team developed a team contract, presented in Appendix C.2. Here, we discussed the expectations of the project and established routines for collaboration, e.g. the process of making important decisions. The team members also had to write down their previous experiences, such as internships and the technologies they were proficient with.

The distribution of responsibility was based on the team members' previous experience and their preferences mentioned in the team contract. This helped to ensure that members got the roles that they were most equipped and motivated for. Table 1 shows the description of the various roles defined and how the roles were allocated.

Table 1: Description and assignment of roles.

Role	**Description**	**Responsible**
Team leader	Main coordinator for team activities. Responsible for customer contact, attending team leader meetings, making the meeting agendas and motivate the team.	Stige
Deputy team leader	Stand-in for the team leader. Responsible for supervisor contact and writing weekly summaries and leads and plans supervisor meetings.	Rebnord
Scrum master	Responsible for the scrum backlog and daily stand-ups. Should host scrum meetings and confirm task assignments.	Ruud
Secretary	Responsible for writing the meeting minutes and assuring good documentation of all important discussions.	Nordal
Report responsible	Responsible for the layout and content of the report. Ensure continuous work on the report throughout the project.	Opdahl and Nordal

Quality manager	Setup quality assurance process. Defining testing requirements. Monitor testing.	Opdahl
Tech lead	Manage pull requests. Cooperate with Scrum Master on converting backlog epics into development tasks. Have an overview of the technical project, and make decisions for respectively the back-end and front-end.	Nilsen and Vittersø
UX-designer and front-end-developer	Designing and user testing the solution. Developing the front-end of the project, implementing the visual design.	Stige, Nordal and Vittersø
Back-end-developer	Developing the back-end of the project, implementing the ontology-matching process.	Opdahl, Ruud, Nilsen, Rebnord and Vittersø

2.3 Project Work Organization

This section presents the meeting structure of the project. To have a steady workflow, the team had weekly team meetings to update each other and organize work. Additionally, we scheduled supervisor meetings, customer meetings, and team leader meetings.

2.3.1 Team Meetings

The team plan to have common work hours Tuesdays from 8:00 - 18:00 and Thursdays from 12:00 - 18:00. These hours contain the weekly team meetings where the project process is discussed and worked at. The team members are free to choose whether to do the work collaboratively or individually. When necessary, the team members also work outside the work hours.

2.3.2 Supervisor Meetings

The supervisor meetings are planned as part of the common work hours on Tuesdays from 10:00 - 10:30. Here both parties will discuss progress and questions related to the project. The supervisor is also available through Slack to answer questions that arise during the week.

2.3.3 Customer Meetings

The customer meetings are held on Tuesdays from 15:00 - 16:00. This is a platform for the team to ask questions and demonstrate the progress made on the project. The sprint review

for an ending sprint and the sprint planning for an upcoming sprint is also held during these meetings, both described in Section 7. The customer is also available through email, where it is possible to ask more urgent questions.

2.3.4 Team Leader Meetings

The team leader participates in a team leader meeting from 13:00 - 14:00 on Wednesdays every other week. During the meeting, the team leaders for all teams discuss suggestions for improvements to the course, or present issues faced.

2.4 Version Control Procedures and Tools

This section presents the tools the team has chosen to use to communicate, plan, and develop the project. It will also present how version control is ensured.

2.4.1 Version Control

Version control is used to record changes to files over time, and in this course it is required to create a systematic procedure for all textual documents and source code.

Overleaf When writing the report the team uses Overleaf's latex editor. Overleaf supports version control and ensures that the report has a clean and professional look.

Google Drive The team decided to use Google Drive for all internal documents, since the service included all the functionality we needed, e.g. text editing, spreadsheets and file storage. The team shared a Google Drive folder with the supervisor and the customer, which contained meeting minutes, weekly reports and time sheets. Google Drive supports version control, and makes it easy to revert to an older version if mistakes are made.

GitHub The team decided to use GitHub as a Git repository because it was known by all the team members. GitHub has a server containing the repository and a local copy on the clients' computers. This ensures everyone has a backup if data is lost. GitHub also has version control, allowing the users to roll back to an earlier version if bugs are found. This can be crucial in a development setting. To keep track of the progress, GitHub Projects were also used as a platform for a Kanban board. The implementation and use of this are further described in Section 4.2.

2.4.2 Communication

Microsoft Teams and Discord Microsoft Teams, as the customer's choice of communication tool, is used to hold customer meetings. Due to the ongoing pandemic, we are not able to meet the customer in person, and do all live communication with them over Teams. We use Discord for remote work internally, because it facilitates collaborative work and is an easy way to contact team members.

Slack As the main communication channel, the team decided on Slack. With the ability to divide the workspace into themed channels and use private messages, it is easy to find back to written information at a later point. The platform is available on computers and smartphones, which makes it easy for the team members to reach each other.

2.4.3 Development

Visual studio code We decided to go for Visual Studio Code as the editor for both the back-end and front-end. Using a single editor instead of different editors for back-end and front-end makes the development process easier to set up and to learn.

2.5 Quality Assurance

This section presents the efforts made to ensure quality in the project. Quality assurance is important to ensure good collaboration within the team and with the customer, to be able to deliver according to expectations. Quality assurance can also be helpful to prevent and minimize mistakes throughout the project.

2.5.1 Time of Response

The team agreed with the customer on how to communicate, and how fast it was required to respond on a request during the week.

- Approval of minutes for customer meeting: 24 hours
- Feedback on documents the customer would like for review: 48 hours
- Approval of suggested project decisions: 48 hours
- Answer to questions on email: 24 hours
- Sending over agreed documents: 48 hours

2.5.2 Templates and Standards

To have consistent and predictable meetings with customers, supervisor and within the team, we made templates for all meetings. The templates also helped to have effective and good sprint events. The team also agreed on standards for both code and report writing. These templates and standards can be found in Appendix C.3 and E.

2.6 Risk Management

To ensure customer satisfaction it is important to be aware of problematic scenarios that might arise during the project. Risk management became even more relevant due to the fact that the project was initiated and completed during the Covid-19 pandemic, which made the execution of the project more unpredictable. The team dedicated a meeting to discuss different means to prevent the risks, and formulate strategies and specific actions to deal with them. The results from the risk assessment meeting are illustrated in Table 2.

It takes into account the different risks the team might encounter, their consequences, and how likely they are to happen, ranked with a probability from Low to Medium to High.

Table 2: Risk analysis.

Category	Risk	Consequences	Prob.	Means to prevent	Actions and strategy	Resp.
All	Covid-19	H: Delays, cancellations, quarantine	M	Follow infection control measures	Adapting to digital working environments. Communicate to ensure all team members are working efficiently	All
All	Team members leaving	H: Loss of knowledge resulting in delays	L	Follow up members of team	Address issues immediately. Keep track of work hours	Team leader
All	Team members not contributing	M: Reduced work ethics	L	Keep track of progress	Handle rapidly. Involve advisor if necessary	All
All	Team conflicts	M: Project delays	L	Information flow	Vote to resolve	Team leader
All	Not meeting the customer in person leading to bad communication and misunder- standings.	M: Delays, reduced customer satisfaction	L	Plan the digital meetings well, use the meetings efficiently. Contact the customer if unsure about something.	Have more meetings	All
Tech	Lack of experience with technology	H: Increased work hours	M	Make sure task are assigned to people with the right skills	Dedicate time for training in every phase. Seek consultation if the problem then precedes.	All
Tech	The technical skills on the team is highly unbalanced	M: Unbalanced workload for team members	M	Pair programming	Consider re-dividing tasks.	Tech resp.
Project management	Cannot finish deliverables on time	H: Delays, reduced customer satisfaction	L	Open communication with customer	Keep up with the schedule, monitor this through weekly meetings. Re-structure workload	All
Project management	Too much time-consuming overhead. Valuable time is spent providing unnecessary documentation	H: Increased work hours	M	Be concise.	Avoid writing more than necessary. Write reports that are useful to read. Consider re-dividing tasks and efforts.	All
Project management	Other subjects or spare time activities take up too much time	H: Do not get everything done	L	Good dialogue.	Discussex-pectationsin team contract. Use deadlines and let members structure their own work time as far as possible.	Team leader

2.7 Effort Registration

The effort registration will be managed in a spreadsheet, and an overview is presented in Appendix B.3. All team members will register the number of hours spent, and also which task it was spent on. The benefits of this are related to tracking how the team used their work hours compared to how they were planned, which is presented in Section 7.6.

3 Pre-study

Creating a solution for the customer requires the team to study the current situation carefully. The purpose of this chapter is to highlight the current situation as well as the wanted situation and specify which benefits and drawbacks that exist. This section will give an overview of the problem and solution space, and what challenges the team will face in the development phase.

3.1 Providing Mobility as a Service (MaaS)

To understand what the customer's wanted solution is, the team had to see the project in the context of establishing MaaS solutions. Modern societies need digital services that promote sustainable mobility, and the concept of MaaS is an important part of this. Many regions and counties in Norway are planning to realize MaaS through their data providers. Ruter and Kolumbus are examples of transport service- and data providers that are participants of the ReiseNavet project and are planning to realize MaaS. The customer will as the project leader in ReiseNavet work towards supporting operators that would like to establish MaaS solutions. This means offering:

- Digital services for MaaS such as travel planning, booking, ticketing, and payment.
- Configuration of MaaS, like what choice of services to be included in mobility packages and choice of business models.
- Semantic model and integration tools for interoperability between data providers and ReiseNavet. This makes it possible to exchange or transfer data between the various digital services and MaaS operators.
- A portal that publish and makes open data accessible for all kinds of transport services.

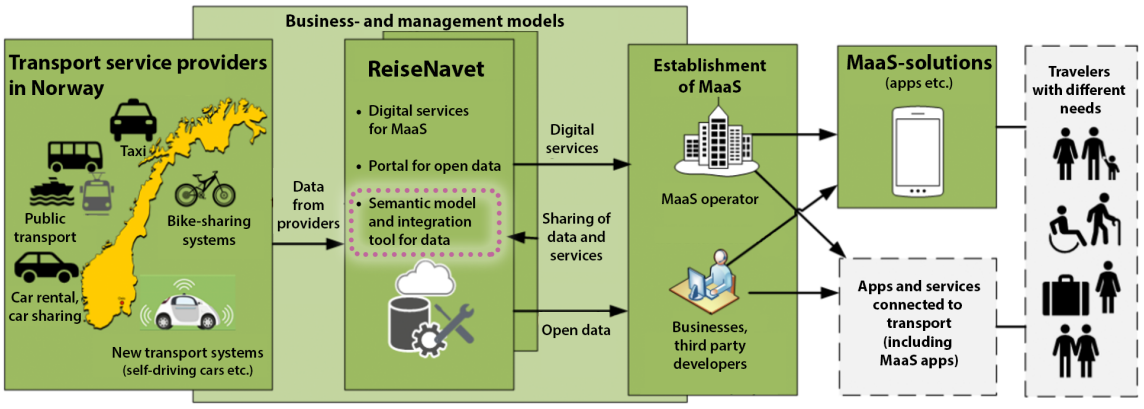


Figure 3: Data integration in the context of establishing MaaS.

MaaS depends on accessible data through open, defined, and standardized interfaces. One of the main activities of ReiseNavet is developing a semantic model to correctly transform between data formats used by transport providers and formats used by Entur. How a semantic model fits into the context of establishing MaaS is shown in Figure 3.

3.2 Situation and Solution of Today

Public transport data are currently delivered to and offered by Entur as standardized formats (e.g. NeTEx for route data and SIRI for real-time data), while data from other transport services have proprietary formats. Entur demand the transport providers to convert to the standardized data format upon sending data to Entur.

Relying on a specific format for exchanging complex transport data among distributed systems has its weaknesses, especially when this format is being updated or replaced. When the format that preceded NeTEx, called RegTopp, was replaced in Norway, this required extensive manual mapping. Considering that the semantic mapping between data formats contributes to 50-80% of the integration process [2], it is clear that a lot of resources can be saved by automating this process.

The transport service providers and other data providers use various data formats with different syntax and semantics to describe their data. An example of another data provider could be the Norwegian Public Road Administration (NPRA) offering traffic and weather data. Alignment of different kinds of data formats is crucial for the realization of a MaaS platform. Today, considerable human interaction is involved to be able to map data schemas element-to-element, so that integration can find place and heterogeneous systems can communicate. The current situation can also be formulated as the following scenario:

Scenario: Manual Alignment An employee of a data provider wants to align their XML-files containing transportation data with the format of the XML-files of the MaaS platform, which is Entur, that follows the NeTEx standard. The employee needs to manually go through the files in XML editors like XMLSpy, Oxygen, or Stylus Studio to align the formats.

3.3 The Wanted Situation

A semantic model can help translate between proprietary and standardized formats using generic concept descriptions [3]. Ontologies for related domains exist today, e.g. Smart Cities [4] and individual modes of transport [5], but there is no semantic model which harmonizes all modes of transport available to MaaS. Relationships between the model and the different formats must be identified correctly to provide good data quality. A lot of research is done on techniques to automate the process, so-called ontology-matching, where similarities in terminology, structure, and linguistic patterns are used to find semantic relations. One example of semantic relations is equivalence, which means that two concepts are synonymous. Another example is subsumption, which means that one concept is more or less general than another, for instance the word “stop” is more general than the word “bus stop”.

SINTEF Digital has existing source code that automatically aligns different data formats and has implemented semantic matching for OWL and RDF files. The customer wants the team to make a solution that extends the existing source code to support functionality for parsing GTFS, GBFS, NeTEx, and IXSI formatted data. The data integration tool should further make it possible to perform semantic matching between these file formats. The solution to be made will be available as an API to be used by the different data providers. It will also include a user-friendly web interface that enables humans to generate, view, and download a mapping between their data format, and the format of the MaaS platform. These functionalities can also be formulated into two different scenarios:

Scenario 1: Alignment with Web Interface An employee at a data provider wants to map data from their own format to the format of the MaaS platform. The employee opens the web interface and uploads their source schema and the file format of the MaaS platform as the target schema. The employee then selects what relations the matching algorithm will identify before getting the aligned file that shows the confidence of the matching.

Scenario 2: Alignment with API An employee at a data provider wants to use an API to automatically align the data files containing transportation data to the format of the MaaS platform. The employee wants to select between matching relations and chooses between equivalencies and subsumptions, or both.

The wanted situation is a data integration tool that will support data providers in semantically aligning their data with the data used by the MaaS Platform. The data integration tool will expand the number of file formats supported by the current solution. The file formats supported are shown in Table 3.

Table 3: File formats supported by the existing solution vs the new solution.

File format	**Current solution**	**New Solution**
OWL	✓	✓
RDF	✓	✓
GTFS		✓
GBFS		✓
NeTEx		✓
IXSI		✓

3.4 Business Requirements

The business requirements explains what the end-product should be like, but not how this will be achieved. Being aware of the requirements is important for all of the team members so that everyone works towards the same goal. The requirements are presented in Table 4.

Table 4: Business requirements.

ID	Requirement	Priority (1-10)
BR1	The system's interface should be user-friendly.	8
BR2	The system should provide an alignment between the source schema and target schema uploaded by the user.	10
BR3	The system should perform semantic matching with both equivalence and subsumption.	10
BR4	The solution should implement all of the existing matching algorithms provided by the customer.	6
BR5	The system should support the following input formats: GTFS, GBFS, NeTEx and IXSI.	7
BR6	The implementation of the system should be easy to understand so it easily can be further developed by the customer.	7

3.5 Evaluation Criteria

To accurately evaluate a proposed solution the team used the evaluation criteria defined in Table 5.

Table 5: Evaluation criteria.

ID	Name	Description
EC1	Business requirements	How well does the solution solve the task
EC2	Development cost	Total time spent implementing solution
EC3	Modifiability	How easy is it to continue implementing on the solution
EC4	Maintenance cost	How much does it cost to maintain the solution
EC5	Time to market	How soon can you release the solution
EC6	Usability	How easy is it for a customer to use the solution

3.6 Market Investigations

Prior to this project, the customer did a literature study of 119 research articles, with the conclusion being that there were no similar projects out there. Currently, there are a number of companies offering tools for manual alignment between different formats. XML

editors like XMLSpy, Oxygen, or Stylus Studio are all frequently used for manual alignment. Among other research projects currently trying to implement automatic ontology mapping are LogMap and AgreementMakerLight. None of these tools are available in any commercial way, and hence there are no other competing solutions on the market.

3.7 Possible Solutions

Before beginning to implement the system, the team discussed possible ways to solve the customer's problem. This section discusses the three approaches the team considered.

3.7.1 An All-Client Approach

A web page consisting of the technology stack HTML-, CSS- and JavaScript/WebAssembly-code is an all-client solution to the customer's request.

Advantages Firstly, multiple team members are already quite familiar with web development and have worked with the technology stack before. Secondly, deployment of a web-page with the aforementioned technology stack is quite simple. This will allow the team to use tools like Firebase hosting, which further simplifies deployment into a process taking only a few seconds, without much setup. Writing all the code in one repository and language simplifies integration.

Disadvantages The customer's existing source code is written in Java. To build an all-client solution, the team will have to translate this existing code into JavaScript or WebAssembly. Even though WebAssembly is built for computing performance, choosing this would be a disadvantage, as no one in the team has worked with it before. As the requested solution is quite computationally demanding, the performance of a client-side application also relies heavily on the performance of the client's device. This means users with low-end devices might spend more time on calculations, and may not be able to finish computations that require a lot of memory. Another disadvantage is that the requested solution currently does not require a database or data storage. If the customer eventually found that distributed storage is needed, e.g. if user authentication is added, then an all-client solution will no longer be sufficient. The solution therefore currently offers less modifiability than the other solutions presented. Lastly, the requested solution includes the wish for an API, or the possibility of having one in the future. By selecting an all-client solution, there is no existing backbone to latch that onto.

3.7.2 An All-Server Approach

An all-server approach includes a web-server running the client's existing Java-code, as well as serving browser-ready HTML-code. If the team is to build an all-server project, the Java framework Spring is the way to go. Spring supports HTML-template-languages that makes dynamic HTML-production, such as providing a table containing the calculated result, a possibility.

Advantages Having a server running Java-code allows the team to directly utilize the customer's existing source code, and implement it directly with the served HTML-code.

This means a lot of development time is saved from not having to re-implement this functionality. Another advantage is that by centralizing all the code on the server, the user interface can be extremely lightweight. The connection between the generated HTML and the calculating Java-code will also be tighter than the other approaches, making it simpler to implement. It will also simplify integration by writing all code in one repository and the same language. Lastly, if the customer needs an API, it is quite easy to implement that with this architecture.

Disadvantages A problem with using Spring is that no team members have experience with this framework or any other Java web-frameworks. This means the team will have to spend quite some time learning how Spring works before any actual work can be done. Spring is also a quite heavy web-framework, so it might feel like overkill to use it for a system that will only have a few views.

3.7.3 A Client-Server Approach

By separating the client-side and server-side code, the solution reflects a client-server architecture. For the project, this approach means building a client-side application in the HTML-, JavaScript- and CSS-stack, as well as a web-server with the customer's existing Java-code. To integrate the client with the server, the server will include a minimal REST-API. This can be done with a light-weight web-framework such as Javalin or Spark, and allows the team to utilize Single-page application (SPA)-frameworks, such as React, Angular, or Vue.

Advantages By splitting the project into two parts, there is a clearer partitioning between the front-end and the back-end. The computing part of the application will go in the back-end, and the user-interaction of the application will go in the front-end. This makes it easier for some team members to develop solely on one part of the application and causes the application's modules to have higher cohesion. As multiple team members already have experience with SPA-frameworks, and with using REST-APIs, the development process will go a lot faster than if everything had to be learned from scratch. This approach leads to less work required to implement the API mentioned in the customer's requirements. Another advantage of the client-server divide is that it is a very modular approach. If at some point in the future, some parts of the application were to be switched out, the REST-API connection offers a loose coupling, making it easy to replace either side of the application.

Disadvantages The client-server approach splits the code base in two, which gives some disadvantages. Firstly, it means the team will utilize multiple programming languages and frameworks, which makes it harder for inexperienced developers to get a good overview of the codebase. Secondly, it means some time will have to be spent on fixing interaction between the two ends of the application. By having two separate parts, the team will also need to deploy both front- and back-end to servers, possibly different ones, for it to be available on the web. This adds complexity to the application and its deployment.

3.8 Evaluation of Possible Solutions

After presenting the possible solutions, the team assessed each of the three approaches with respect to the evaluation criteria in Table 5. Each criterion would yield a score between 0

and 10. If the solution would not achieve anything, a score of 0 was given. If the solution was regarded as perfect for fulfilling that specific criterion, 10 points were given. As the EC1 is the most important criterion in the form of the business requirements of the project, this is weighted double, between 0 and 20. The assessment is presented in Table 6.

Table 6: Evaluation of possible solutions.

Possible solution	**EC1**	**EC2**	**EC3**	**EC4**	**EC5**	**EC6**	**Sum**
All-client approach	14	3	5	7	2	2	33
All-server approach	18	4	6	8	3	6	51
Client-server approach	20	10	8	8	5	10	61

There was not a huge difference in how well the different solutions would perform in terms of functionality. One of the most defining criteria was the development cost, which includes how much time the team would spend implementing the solution. Time to market was a less important criterion, since the length of the project was set, and was therefore given lower scores.

3.9 Choice of Solution

The evaluation of the possible solutions is presented in Section 3.8. Based on this evaluation the client-server approach was chosen. The decision was also based on the team's previous experience with the technologies and the architecture, as it will help them to quickly build the core of the application. It was also the preferred solution by the customer. This section will go deeper into detail on the choices of programming languages and frameworks.

3.9.1 Front-end

The customer's desired solution was described as a web-application, so with regards to the front-end, we decided on building a JavaScript single-page application. This also fits well with the criteria to minimize development cost (in developer hours), and maximize usability and availability to consumers.

Programming language Web applications force us to use technologies that are based on JavaScript, HTML, and CSS, as modern browsers read only these file formats.

Framework Because the system description fit the description of a single-page application, the team decided to use a JavaScript framework to accelerate the development process. This meant we had a few choices among mainstream, modern web-frameworks: Angular, React, and Vue.

No one in the team had any experience with Angular, so it was quickly dismissed as less practical than the two other alternatives. Between React and Vue, multiple members of the team had used either technology. The team discussed their experiences with the two

frameworks and decided on Vue due to its ability to remain systematic in large team projects, where the team agreed React was far less helpful.

Vue is a model-view-controller based JavaScript/Node.js framework, which means it follows a predefined architectural pattern, and "forces" the developer to follow them as well. Although this restricts some possibilities from the developers, it also means that the team's collective code will be more easily maintained. The team decided that this was very valuable in trying to keep the project tidy.

Component library To have an aesthetically pleasing design, the team decided to use a pre-built component library, with pre-defined styling. The team's front-end lead developer compared the top available libraries, and selected Vuetify for this task, as it follows Google's Material guidelines, implements accessibility rules, and is quite easy for the developers to use.

3.9.2 Back-end

Some of the functionality required in the solution was already implemented in a previous project made by the customer. Therefore, the team decided to interface this with the back-end application. We then determined that it would be most efficient to connect the existing project to a web server, to allow users to access the functionality through a REST-API.

Programming language The customer existing Java-project of the customer contained some prebuilt semantic matching-algorithms. Because of this, the team decided that it would be inefficient to begin writing the application in any other language than Java. Thus, we chose Java as the language for the back-end project.

Framework With the front-end being Vue, and the back-end being Java, we had to use a web framework to handle the communication between them. After discussing with the team we agreed that because the website would be a single page application and it would handle basic requests, it would be best to go for a lightweight web framework. A lightweight web framework has fewer concepts you need to learn and is easier to work with, which makes them more appealing. After researching we found out that there are two popular lightweight web frameworks: Javalin and Spark.

By researching, we found out rather quickly that Javalin was pretty much superior over Spark in every way. Javalin is a ground-up rewrite of Spark, it covers the same features, but with better documentation, is more robust, and is easier to learn. We therefore decided to go for Javalin.

4 Development Methodology

When developing a system in a team, it is important to settle on a methodology to manage the development process. A good methodology can help communication within the team and with the customer. It can also help with the structure and prioritizing of tasks, and generally streamlining of the process. In this section, we will discuss the choice of development methodology for the project and present the choice along with the teams' implementation of it.

4.1 Choice of Methodology

The team decided early on to go with an agile development methodology. It is generally known that agile is more effective in development teams than the traditional top-down approaches such as the waterfall method [6]. Especially in such an open and unpredictable task with a new team, we considered it an advantage to work agile and not have to plan all aspects of the project from the start. Some team members had prior experience with agile development methodologies, and the team considered it advantageous to choose a methodology the team had some familiarity with. This restricted the set of methodologies that we considered relevant for the project. Each of the methodologies will be discussed further in detail, along with some of their advantages and disadvantages.

4.1.1 Scrum

Scrum is an iterative framework for agile development that also includes a framework for project management. It contains a set of roles, events and resources, and divides the project progress into time-boxed iterations of development called sprints. Scrum teams consist of three distinct roles - the product owner/customer, the Scrum Master, and the development team members. The main events to fulfill a sprint include sprint planning meetings, daily stand-ups, the sprint review, and the sprint retrospective, as illustrated in Figure 4. Each of the events is further described in Section 4.2.5.

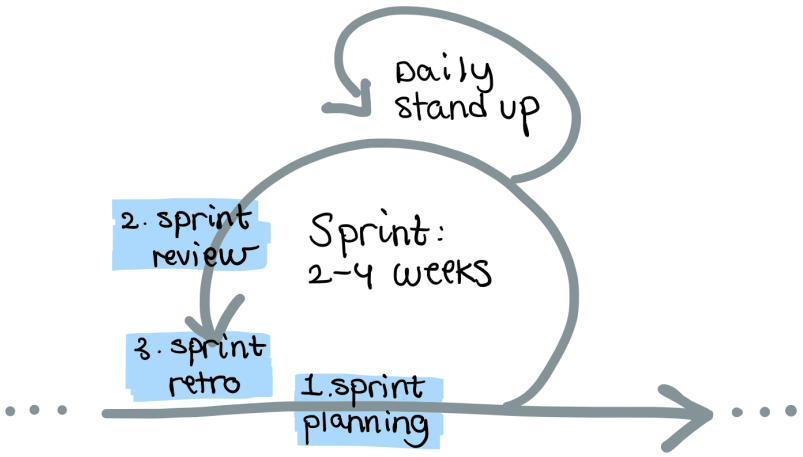


Figure 4: The workflow in Scrum.

The project work is managed in a *product backlog*. The team develops a list of tasks in cooperation with the customer consisting of all the tasks that need to be conducted in

order to finish the product. All work items should be included in the backlog: User stories (description of features told by the customer), bugs, design changes, customer requests, action items from the retrospective, etc. We then estimate how much time each work item will take to finish, and prioritize them in cooperation with the customer. During each sprint planning meeting, we select a set of tasks from the backlog based on aspects such as the tasks' time estimates and priorities, that will be fulfilled in the upcoming sprint. At the end of a sprint the customer is informed whether or not the sprint backlog work items are finished, and in that case, presented a demo of the system. We also arrange a sprint retrospect to discuss what went well and what could be improved to be able to optimize the next sprint. [7, 8]

Advantages A well defined and well documented framework for both task management and project planning, with defined events, roles, and templates. Easy to understand and adapt for a newly established team.

Disadvantages Once the sprint backlog is set, it can not be changed, and new items from the backlog must be handled in the next sprint planning. Can be effective in many projects, but might be too strict because of the team's lack of experience in time estimating. Therefore a more flexible approach to the backlog would be more desirable.

4.1.2 Extreme Programming (XP)

Extreme programming, or XP, is also an agile framework for development but focuses more on customer satisfaction, continuous planning, and delivery rather than the strict deadlines and iterations. Unlike Scrum, XP gives the developers the power to do immediate changes to the backlog, even though the developers are not in a dedicated planning phase of the process. Instead of strictly defined sprints with sprint events, XP promotes rapid feedback loops, continuous testing, continuous planning, and close teamwork to deliver high-quality software at very frequent intervals, typically every one to three weeks. The typical workflow is presented in Figure 5.

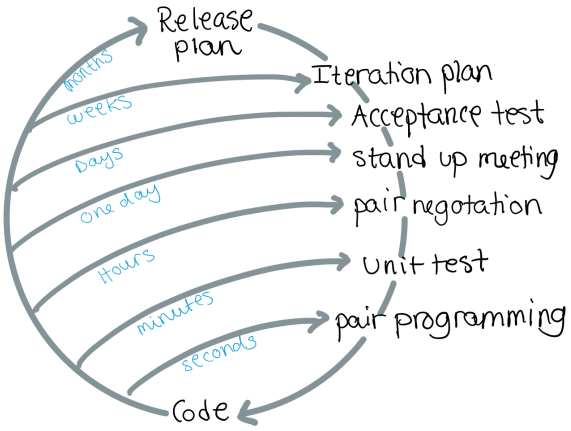


Figure 5: The workflow in XP.

Like Scrum, XP teams also consist of three roles; manager, customers, and developers. They are all equal partners in a collaborative team, which could guarantee the effectiveness of

development progress. Communication and respect are key words in the XP collaboration, and working frequently together in pair programming, negotiations, customer updates and so on are important to always meet the possibly changing requirements. XP is also a test-driven methodology that can secure good code and product quality along the way. [7, 8]

Advantages More flexible with the backlog and customer. Also a good way to secure good code with the test-driven focus.

Disadvantages Less defined events, higher focus on the software development rather than the whole project in total. We consider this a disadvantage as this project is managed by the team themselves and includes a huge management aspect. This includes the whole chain from defining, planning and designing, to development, testing, and documenting, in addition to the actual software development.

4.1.3 Kanban

Kanban is a methodology designed to help teams work together more effectively and focuses on just-in-time delivery and continuous delivery while not overburdening the team. The concept of Kanban is to manage the number of tasks in development by introducing the term WIP (Work in Progress). The team members continually pick tasks from the backlog based on priority and move them from the backlog to the WIP until finished. The number of tasks placed in WIP is limited to ensure the team gets things done, as well as to prevent tasks from accumulating. A feature of Kanban that visualizes this concept is the Kanban board. The Kanban board is commonly used in combination with other agile methodologies. It gives an overview of all the tasks every team member is assigned to, and in what stage in the process they are, to increase the collaboration and effectiveness in the team. [7, 8]

Advantages Since the progress of work can be seen explicitly, the project schedule and progress could be handled more easily.

Disadvantages Kanban does not include the same sprint mindset or project planning systems as Scrum does, which can make it harder to divide the project into different iterations or phases as we wish to do.

4.1.4 Rationale

When choosing a development methodology, the team prioritized the possibility to work in sprints, support in project management, and the existence of a good template to visualize and track the project progress. **Scrum** was chosen as development methodology as it enables us to deliver the customer's most wanted functionalities by prioritizing tasks. In addition to Scrum, the team decided to use some features from the Kanban methodology, more specifically the **Kanban board**. The Kanban board helped the team visualize both the sprints and the report writing, and made it easier to cooperate remotely and to track the project process.

The team agreed that collaboration would be more effective if we did not change the focus too much within the sprints. We therefore decided to aim towards not making changes to the backlog outside sprint plannings. But because we assume that the customer's expectations

and requirements will be modified throughout the project, we wanted the possibility to pick lower priority tasks from the sprint backlog during a sprint. We therefore considered Scrum as the natural choice of development methodology in this project. Although we saw the benefits of having the flexibility to change the backlog as we wanted during each sprint, the risk of miscommunication was considered higher if we allowed it to be as flexible as in XP. The quality and amount of documentation of Scrum available, the team-members' own experience, and the fact that it is frequently recommended by NTNU were also weighing in for this alternative. The team concluded that it would not be efficient for us to develop test-driven, even though it is a way to ensure code quality. We did not see the benefit of having a big test focus early on when the team could be spending more time on thorough planning and fully understanding the task. This contributed to excluding XP as methodology for the project.

A consideration when making these decisions was the fact that the team does not work full-time on the project, and that many work hours will be spent working remotely due to the pandemic mentioned in Section 2.6. It was therefore important to pick a framework that facilitated a good collaboration without following the optimal path of daily meetings and conversations. The solution was to take advantage of the strict sprints in Scrum to clearly define the expectations of each team member and the team as a whole per time interval. In that way, we can work efficiently both remotely and with different schedules within the team. The Scrum frameworks' strict deadlines, predefined roles, prioritized tasks, and regular events therefore suited this project well, as it laid the foundation of a concise set-up that everyone knows the rules and routines of, without having daily contact with each other.

4.2 Implementation of Scrum

The Scrum methodology comes with a set of defined resources and event templates that can be highly useful if used appropriately. This section describes the team's use and implementation of these resources, and the adjustments made to fit the team's preference and the task specifically. Resources outside Scrum that were used are also commented, and this section complements the justification of methodology choice to a certain extent. A written summary of each sprint is presented in Section 7.

4.2.1 Product Backlog

The product backlog in this project was maintained in a Google Sheets document and consisted of a set of superior tasks referred to as *epics* by the team. Each epic consists of a set of data fields, e.g. priority and start- and finish date, that gave the team high value in both the planning and progression tracking of the project. Appendix B.1 shows how the product backlog looked like at the end of the project with all its data fields. Note that several changes have been made during the sprint iterations and some changes are illuminated in the respective sprints' section in 7. All changes in the product backlog were made by the team during planning phases and approved by the customer according to Scrum methodology.

Frequent adjustments and continuous updating of the data fields made the product backlog a reliable source of real time information on the total progress of the system development. The product backlog was directly linked with the time sheets for effort registration (described in Section 2.7) to have the *time spent* column automatically updated when someone registered their work hours related to the epic. This connection guaranteed that the product backlog

was always consistent with the actual amount of hours spent on each epic by the team in total, and made it possible to assess the accuracy of the estimates of all tasks during the sprint retrospective meeting.

4.2.2 Story Points

In this project the story points were directly measured in hours *(h)*. All epics contained an estimate describing the number of hours the team predicted it would take to complete it, which is crucial information in Scrum methodology because it makes it possible to predict the total amount of work required to deliver a product. The backlog then works as a way of both planning each sprint with a logical distribution of the work hours, and continuously tracking the progress of a sprint relative to the project as a whole. A burn-down chart was used to visualize the completion of epics for each sprint, so the team could easily follow the progression, and is presented in the respective sprint's section in Section 7.

The burn-down chart was used during sprint plannings to discover misjudgments in estimates that affected the progression. A large difference from the estimated progression to reality can result in an overpriced project or an unsatisfied customer, so the hour estimation was an important parameter in order to always be able to justify choices and prioritization to the customer. In that way, the team could compensate for misjudgments and adjust the plan accordingly so that we always gave the customer a realistic expectation of the following progress. The sections for each respective sprint in Section 7 present a more detailed insight to the progression of the project and challenges related to it.

4.2.3 Prioritization of Work

Each backlog epic was marked with the sprint it was planned to be implemented in and were labeled from 1 to 3 based on importance, 1 being most important and 3 being least important. These two data fields constituted an order of priority in the product backlog for the customer and the team to have a common understanding of what epics should be brought into what sprint. The chosen epics to be prioritized in a sprint were brought from the product backlog into the *sprint backlog* and constituted as the customer's expectations of what will be done when the sprint is finished. The sprint backlog in this project consisted of a set of smaller tasks referred to as *issues* by the team. The issues originated from the epics in such a way that several issues constituted an epic, and had a clear definition of done. The issues were small enough to be finished by one team member in roughly a day of work. Scrum methodology states that the sprint backlog can not be changed during a sprint, but allows the team to choose lower priority epics when creating the sprint backlog. The team followed this standard throughout the project, with some exceptions in cases where the workload was underestimated in a sprint to make sure all team members had work to do at all times. In these specific cases the customer approved for more epics brought into the sprint, but a general rule was to minimize the number of changes to the backlog during a sprint.

4.2.4 Sprint Management

Each sprint was represented in a unique Kanban board in GitHub Projects, along with a board for report writing. An example is presented in Figure 6. All related issues for a sprint were included in the sprint's board, and each issue included a reference to the epic it originated from. The team members assigned themselves and further followed the Git Flow

standard for version control presented in Appendix E.4. The Kanban board made it easy to keep track of the progress and to see which team member was working on what issue. It also helped to ensure that no part of the project was overlooked during the implementation of a sprint.

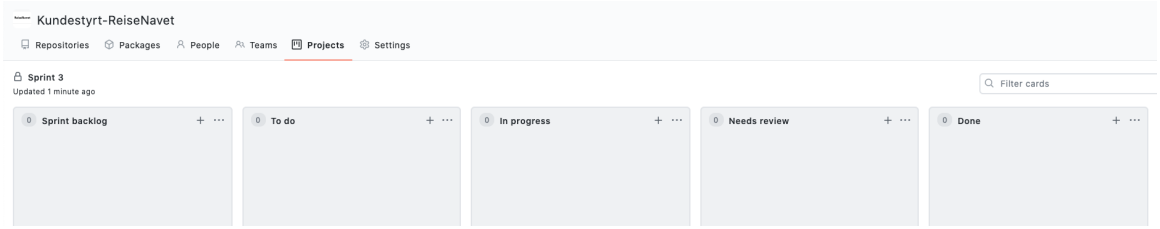


Figure 6: Example of a Kanban board in GitHub Projects.

An important aspect of the Kanban methodology is the limited amount of work in the "in progress" stage of the workflow. The team chose not to follow this because we wanted to make sure that all team members had work to do at all times and did not want to take the risk of outreaching the limit, with only a few members' workload counted due to an uneven distribution at some point in the project. Although we had the general rule of finishing one issue before starting a new one which limited the work in progress column in the Kanban board at all times to the number of team members in work.

Definition of Done The team operated with a common definition of done for all epics. The definition stated that for an epic to be moved to the "done" column in the board, it required review by one of the tech leads or report responsible. Then it either had to be merged into the *development branch* (called dev) or put into the Overleaf document, dependent on whether it is technical or report related. The tech leads were responsible for all issues to contain a checklist of what should be implemented for the issue to be done and required branches to be deleted after merging with *dev* to keep the GitHub repository well organized.

4.2.5 Sprint Events

The following events were implemented in the project as part of the sprints in accordance to the Scrum methodology. See Figure 4 for an overview of the sprint events in Scrum.

Sprint Planning All the sprints started with a planning meeting referred to as the sprint planning, attended by the team and the customer. The customers were presented with the team's suggested epics to bring into the sprint backlog, their priority, and a defined goal for the sprint. The customer had the option to deviate from the suggestion and change the sprint backlog. Eventually, an agreement would be made and the outcome of the sprint planning would be a sprint goal and a sprint backlog to be implemented within the end of the sprint.

Daily Stand-up The daily stand-up is a short meeting, with a typical duration of 15 minutes, attended by the whole team to keep each other updated on the progress. While the Scrum methodology requires daily stand-up meetings the team did not have the possibility and did not need that frequency of updates because of reasons mentioned in Section 4.1.4.

The team arranged stand-up meetings two days a week as part of the team meetings. The stand-up meetings consisted of a short update from each team member according to the teams *PPP template*:

- **Progress** What has been done since last stand-up?
- **Plan** What will be worked on towards the next stand-up?
- **Problems** Is there anything that prevents the team member from fulfilling the task?

Sprint Review All sprints ended with a sprint review attended by the team and the customer. The sprint review consisted of a demonstration of the work that was done on the solution, and a review of the backlog to see if all the epics were implemented and if the work done fulfilled the sprint goal. The customer typically had some feedback regarding the work and some questions for the team. The sprint review typically laid the foundation for the next sprint planning meeting which came right after.

Sprint Retrospective After the customer meetings, the team wrapped up each sprint with a retrospective meeting. This is an internal meeting to consider the team's collaboration, work, and routines during the previous sprint, and consists of a workshop activity with discussion. The team pointed out what went well and what could be improved from the sprint and came up with action points to improve the non-optimal parts. Post-its were used on a whiteboard to make it a more participating activity, and the template of the activity can be seen in the retrospective section of each sprint in Section 7. This was an arena for sharing thoughts and the team was able to make it a safe space to ventilate what was on the team members' minds.

Internal Technical Course This event is not a part of the Scrum methodology. It is included in the implementation description because it was the team's way of obtaining the knowledge required for the development sprints. As an introduction to the development sprints there was arranged a back-end and front-end course by the respective tech leads at the beginning of Sprint 2. The team was given an intro to the programming languages and frameworks to be used and were shown the customer's existing code base. This technical course assured a common minimum level of knowledge before developing and all team members got the help they needed to get started on a technical issue.

5 Requirements Specifications

Establishing a set of requirements for the project is crucial to make sure the team works towards delivering a solution that fulfills the expectations of the customer. Defining requirements early made sure the team would be on the same page as the customer from the beginning, and thus defining these specifications was of top priority in the initiation phase of the project.

Through close dialog with the customer, the requirements were formulated as technical constraints, business constraints, use-cases, functional requirements, and quality attributes. In the early phase of the project, we received a set of user scenarios from the customer, which can be found in Appendix D.1. In this section, we formulate these user scenarios into use-cases and functional requirements.

5.1 Technical Constraints

The technical constraints describe the unchangeable technical design decisions of the team and are defined in Table 7.

Table 7: Technical constraints.

ID	**Description**
TC1	Use of Vue as front-end framework
TC2	Use of JavaScript, CSS and HTML for front-end
TC3	Use of Java for back-end
TC4	Use of Javalin as API framework
TC5	Use of virtual server at ProISP

5.2 Business Constraints

The business constraints formulate the fixed business decisions that could restrict the team, and are shown in Table 8.

Table 8: Business constraints.

ID	**Description**
BC1	The project must be delivered in the middle of November
BC2	The team can not get paid by the customer
BC3	Intellectual property rights issues are handled according to NTNU and Norwegian regulations

5.3 Use-Cases

The team formulated specific use-cases based on the user scenarios provided by the customer. The use-cases are defined in Table 9.

Table 9: Use-cases.

ID	**Description**
UC1	An employee at a data provider has to map their data from their own format to the format of the MaaS platform, and wants to use a web interface that takes a source and target schema, and aligns them.
UC2	An employee at a data provider wants to use an API to align the format of their files containing transportation data with the format of the MaaS platform
UC3	An employee at a data provider wants the data integration tool to identify equivalencies in the semantic matching.
UC4	An employee at a data provider wants the data integration tool to identify subsumptions in the semantic matching.
UC5	An employee at a data provider wants to check the confidence of the different alignments computed by the data integration tool.
UC6	For a user to integrate new data into the MaaS Platform, the data integration tool should be able to automatically identify equivalence and subsumption relations between a source ontology and a target ontology

5.4 Functional Requirements

The functional requirements defines the basic system behaviour, and are presented in Table 10.

Table 10: Functional requirements.

ID	Description	Complexity
FR1	The user pushes a button “Source schema” and gets a dialog asking the user to choose a file from disk	Low
FR2	The user pushes the button “Target schema” and gets a dialog asking the user to choose a file from disk	Low
FR3	The user can choose “subsumption”, “equivalence’ or both, via checkboxes before submitting the form. (Semantic relations)	Medium
FR4	The user can start a matching process by pressing the “Compute Alignment”-button, and clear the page by clicking on a “Clear”-button	High
FR5	The user can view the resulting alignment on screen when it has been computed	Medium
FR6	The user can download the resulting alignment by pressing the “Download Alignment”-button	Medium
FR7	The user can choose to make another alignment by pressing the “New Alignment”-button	Low

5.5 Quality Attributes

The quality attributes, also known as non-functional requirements, describe the performance of the system, and hence do not explain any of the specific features. Defining the quality attributes is important in order to define what the core principles of the system should be. The quality attributes were formulated early in the process, and confirmed and revised together with the customer.

5.5.1 Scalability

The system has to perform well as the input schemas for matching become larger and more complex. The solution is based on nine different matching algorithms, all of which have different running times. As the input size grows, the running time should not grow more than expected.

5.5.2 Modifiability

Since we are building a system using ReiseNavets already existing source code, it is useful for the customer to be able to easily expand and modify the system. A modifiable system is important both for the team itself, but also for the customer to further build new functionalities for the system.

5.5.3 Modularity

The customer specified a machine-to-machine interface as one of the features the data integration tool should include. An API is a modular approach to the architecture, and ensures flexibility for the system, as it offers use for both human-to-machine- and machine-to-machine interaction.

5.5.4 Availability

The system should provide reliability for end-users. The system must serve every user with the expected functionalities at all times and hence be available for them to fulfill all requirements.

5.5.5 Usability

It was specified by the customer that the solution should fulfill the principles of Universal Design[9]. Fulfilling these principles would mean for the team to respect the variation in functional abilities of the end-users, including people with disabilities. When making a universally designed solution, the usability of the solution should reach all users in the target audience, regardless of their abilities.

6 Architecture

To lay the foundations for an efficient development process, it is helpful to plan out the overall structure of the project. The team does this by exploring the process needed to solve the customer's problem, and then designing an architecture that fits the process. It is crucial that this architecture and process meet the requirements put forward by the customer.

The architecture decided on is an implementation of the client-server design pattern, which implies that the team has separated the code executed on the client's device from the code executed in the server environment. This decision is directly connected to the choice of solution in Section 3.9. The internal architecture in the front-end/client-side code strictly follows the natural structure of the framework (Vue) [10], while the back-end/server-side code builds on principles from procedural programming and component based development.

6.1 Architectural Views

In order to better visualize this architecture and the processes involved, the team has created architectural views. A popular way to describe the many facets of a complex system is with the 4 + 1 architectural model [11], consisting of the Logical-, Process-, Physical- and Development views, plus scenarios. The views are shown in this section, while the scenarios are shown in Table 9 in Section 5.

6.1.1 Logical View

Figure 7 shows a mid-level logical view of the classes and interfaces needed in the back-end architecture. The front-end architecture is not class based, and does not behave in an object oriented fashion, so it was excluded from the diagram.

The arrows represent references from one class to the next. There are many classes used in the project which are not described in the diagram. These classes were excluded because they do not impact the architecture in any significant way, but simply offer utility to the classes that do impact the architecture.

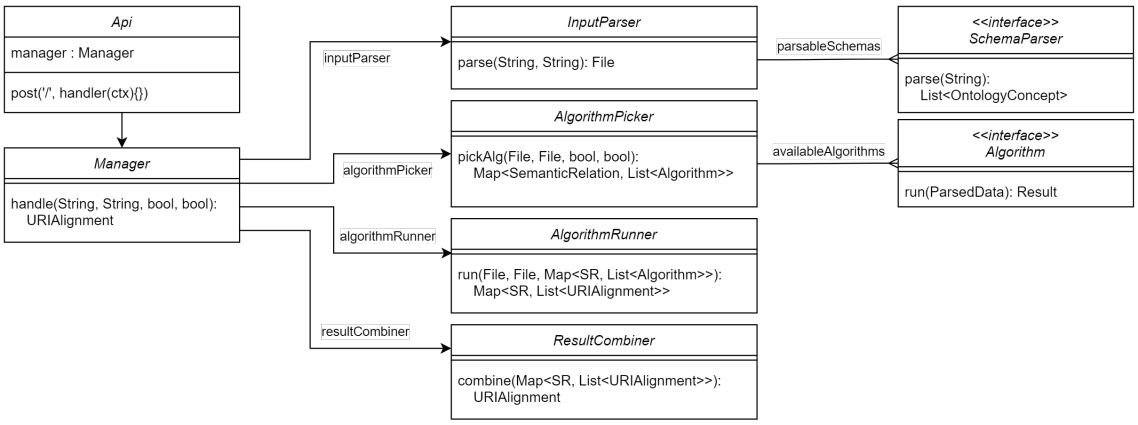


Figure 7: Class diagram representing the back-end architecture.

6.1.2 Process View

Figure 8 describes the flow of data from the user's request, through the solution's "pipeline", and back to the user as the result. The data flow diagram is read top-to-bottom, with the exception of the curved arrow from Manager to API endpoint. This process view gives the best understanding of what the back-end does with the user's request and the order of operations. It is very useful for understanding the task of each component because it describes both input and output.

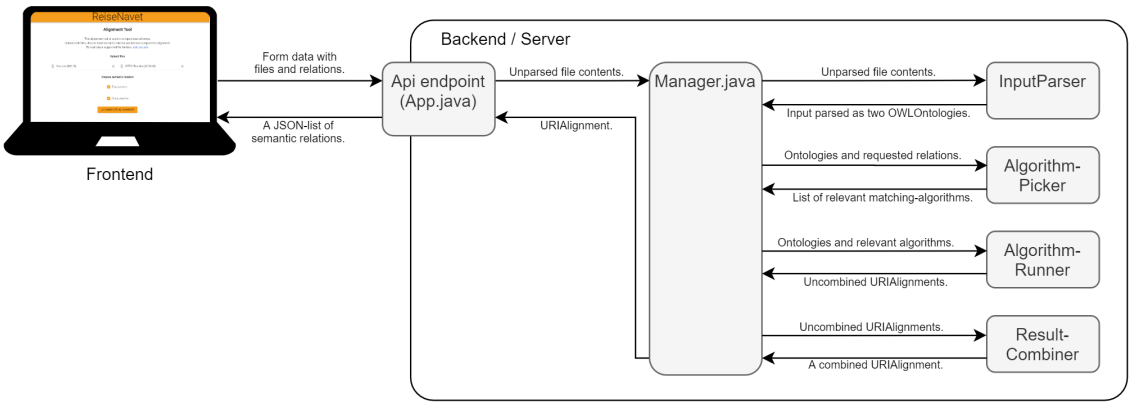


Figure 8: Data flow diagram showing the back-end process.

6.1.3 Physical View

Figure 9 is a physical view showing the single physical component in the deployment architecture: The server. The diagram also shows what will be deployed onto it; web server software hosting the website and the alignment software with the API.

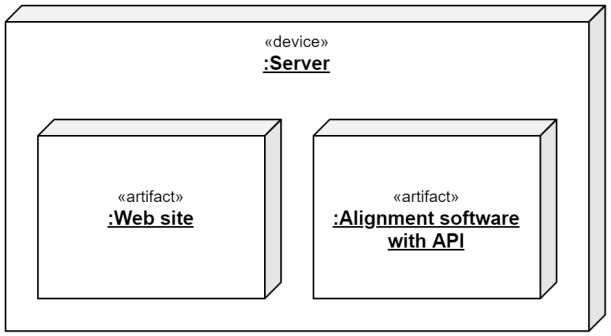


Figure 9: Deployment diagram describing the server hosting the application.

6.1.4 Development View

Figure 10 shows the development view for both front-end and back-end. As previously discussed, the front-end is quite simple and isn't highly divisible. The back-end, however, is divided in multiple ways. The white rectangles symbolize classes contained in the team's code, located in the "services"-package. The dotted rectangle within the back-end shows the packages relevant to the ontology matching process. Finally, we see that the "Matcher-

Algorithms" implement the "Algorithm"-interface, allowing us to integrate with them quite well.

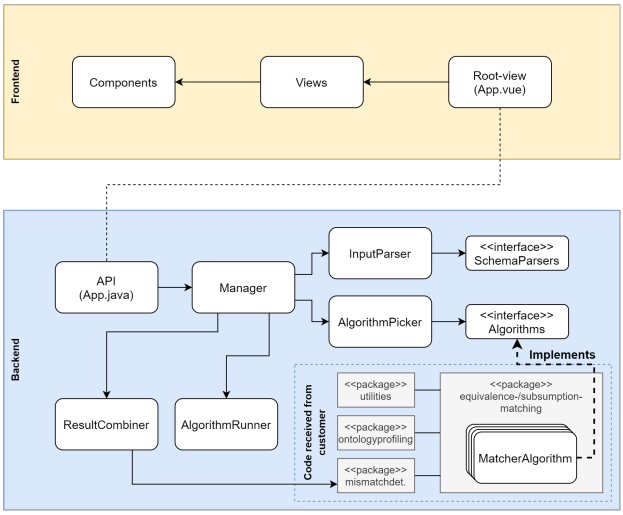


Figure 10: A variation of a package diagram, showing the developer how the different classes and packages interact.

6.2 Rationale

The choice of architecture is tightly knit together with the choice of solution. A client-server approach is presented in Section 3.7.3, and the argumentation for this solution is found in Section 3.9. This section will further discuss the architecture of the solution.

The team uses principles from procedural programming. In Java's case, procedural programming implies minimizing object initialization and making use of static methods [12]. The reason is that the solution can be easily separated into smaller steps, and the problem is clearly defined with specifications that won't change. This makes it an ideal situation for procedural programming [13].

The solution is a part of a larger project, and it is likely going to be developed further. The project has been designed with modularity and modifiability in mind allowing for future use of its modules in different settings or for a different purpose. These concepts are discussed as quality attributes in Section 5.5.

To achieve a modular and modifiable system, the team follows the practice of Component-based software engineering (CBSE). CBSE is the approach of defining, implementing, and composing loosely coupled independent components into systems [14]. Additionally, related code are structured together to make the system highly cohesive. High cohesion and loose coupling is an important focus as it decreases the cost of implementing new features, changing the existing code, and testing the system [15].

7 Documentation of sprints

The project consists of five sprints starting with Sprint 0 which was conducted in week 36. This section presents a written summary of each sprint according to the defined implementation of Scrum presented in Section 4. Appendix B.2 shows a visualization of the work implemented in each sprint represented as a Gantt chart.

7.1 Sprint 0

The first sprint is neither a development sprint nor a pre-study sprint, but more of an introduction phase for getting started on the project. The team figured it would be helpful to work through it as if it were a sprint even though there was no existing backlog yet, just to get some practice. This gave the team an opportunity to get some experience with the methodology and to get used to working with each other as a team. In that way, we could early identify possible difficulties related to the execution of the project.

7.1.1 Sprint Planning

Sprint 0 began with the team's first sprint planning meeting with the customer. The customer was introduced to the essence of Scrum methodology, and how the team imagined the implementation of it. The goal of the sprint was discussed, and we spent some time establishing routines and expectations.

Sprint goal The sprint goal presented and approved by the customer was as follows: *Create a first draft of the backlog, choose and document choice of technologies and come up with a detailed plan for the pre-study sprint, Sprint 1.*

Effort estimation Since one of the goals of the sprint was to make a draft of the backlog, there were no defined epics brought into the sprint during planning. While the upcoming hour distribution was still quite diffuse to the team at this point, most hours were expected to be spent on further planning and resource set-up. This involved creating the backlog, and documentation of all decisions in the report. Figure 11 shows formalization of the epics that had already been implemented during this first sprint, as part of the now defined product backlog. All epics are marked as priority one due to the fact that they had already been implemented and did not need to be prioritized.

	Epic name	Category	Priority	Hours estimate
1	Choose technologies	Architecture	1	8
2	Set up project environments	Development	1	15
3	Write readme for project	Development	1	8
4	Define team contract, roles, meetings etc.	Project planning	1	40
5	Choose development methology	Project planning	1	5
6	Create templates for meetings, documentation, ...	Project planning	1	15
7	Set up google drive and define use and structure	Project planning	1	10
8	Project - and sprint planning (sprint overview)	Project planning	1	10
9	Risk analysis	Project planning	1	5
10	Create report in Overleaf and define structure	Report	1	4
11	Document technology choices (section in planning)	Report	1	4
12	Document Introduction	Report	1	10

Figure 11: Sprint backlog for Sprint 0.

7.1.2 Sprint Review

Demo for customer The customer was presented with the product backlog draft which included a plan for the pre-study sprint with a set of defined epics to be done. The presentation of the product backlog also revealed a detailed plan for the development sprints, showing when functionality are estimated to be in place. In addition to the product backlog, the tech leads presented for the customer the team's choices of technologies, which were all subsequently approved and can be read about in Section 3.9.

Progress and unfinished work Eleven out of the twelve epics from the backlog were finished during the sprint, while one epic was still in progress; *E11: Document technology choices*. This epic regards the documentation of project choices for the report and does not affect the solution. Time estimation was not done in a valuable way for this sprint, mainly due to the fact that the product backlog did not exist until the end of the sprint. In addition, there was no definition of done for the sprint work, so the team struggled to define the amount of work that could be done during the sprint.

Feedback from the customer The customer was happy with the demonstration and was satisfied with the progress on the planning and setup of the project. They also agreed on the overall plan implied by the product backlog and pointed out their trust in the team's assessments and choices at all stages.

7.1.3 Sprint Retrospective

At the retrospective meeting, the team concluded that the communication with the customer was going great. Everyone agreed that the customer meetings, as well as the team meetings, were highly effective work hours during the sprint, but the downside of this was that the efficiency was gone as soon as these meetings were over and the team members were on their own. The main problem was the lack of a common understanding of the scope of the tasks, and what was expected from each member. This was probably a result of beginning a sprint without having a backlog with time-estimated epics and well-distributed tasks, so the whole sprint was a little disorganized despite an effort to define each task that had to be carried out. Table 11 presents the outcome of the retrospective meeting after Sprint 0.

It was an important lesson for the team to understand that all tasks required a clear explanation and definition of done, as well as a good distribution among the team members in order for us to be able to work on the project separately. We also longed for a way of tracking the progression during the sprint and were therefore set on using a burn-down chart for all upcoming sprints to solve this problem.

Table 11: Retrospective analysis for Sprint 0.

What went well	**What could be improved**
• Good communication within the team and with the customer and supervisor • Coming together in smaller groups to work on a problem was effective • We had a clear agenda for each team meeting • There was created concrete defined task for the sprint	1. Lack of defined hour distribution or agenda for the common time outside meetings specifically made the days as a whole less effective 2. Unclear who is responsible for what task resulting in overlapping work 3. Lack of things to do between meetings because of the unclear division of responsibilities 4. Too much discussions in plenum that were not required to be overheard by the whole team 5. Lack of updates on what everyone is doing (not optimal use of stand-up), especially if a team member misses a meeting 6. Not all todo's from team leader were followed up

Action points Naturally, there was room for improvement due to the sprint being the first time working together as a team and on the project. The team chose three to five notes from the sprint retrospective meeting that was the most important to the team or that could make the biggest impact on the team's work in a positive manner if were improved. We asked ourselves: "How can we make these improvements?". Table 12 shows the selected notes and corresponding action points and the team member assigned as responsible for fulfilling it.

Table 12: Action points created from retrospective meeting Sprint 0.

Chosen Note	**Action Point**	**Responsible team member**
1	An agenda for the whole day on Tuesdays and Thursdays are made in advance and posted in slack. Everyone is expected to read the agenda and fulfil their todo's prior to the meeting.	@team leader and @dep. team leader are responsible for an agenda for all common time
2	Delegate more tasks specifically during sprint planning and be clearer on who is responsible for what	@scrum master with @tech lead and @rapport responsible for their respective fields
4	Have a chairman for each meeting that overviews the discussions and makes sure discussions not requiring the whole team are scheduled outside meetings	@team leader will work as a chairman
5	Update each other on slack if a task is done and everyone should prepare more for stand-up meetings	@everyone

7.2 Sprint 1

In this sprint, the team took a deeper dive into the actual problem that the team was appointed to solve. Sprint 1 therefore served as a pre-study sprint and as an opportunity for the team members to get a deeper understanding of the fields involved, the existing code base, and the customer's expectations of the solution.

7.2.1 Sprint Planning

Sprint goal The sprint goal presented and approved by the customer was as follows: *Define architecture, define result format for the matching of ontologies, and create a non-technical design suggestion for the web-application.*

Effort estimation The backlog consulted a suggested set of epics brought into the sprint backlog for Sprint 1, shown in Figure 12. In addition to these, the unfinished epic from Sprint 0 was already started on and was planned to be finished as soon as possible.

	Epic name	Category	Priority
13	Decide and define result-format	Architecture	1
14	Define architecture	Architecture	1
15	Create design-suggestion	Architecture	2
16	Guidelines for coding	Project planning	2
17	Document Development Methodology	Report	1
18	Document Planning (minus technology choices)	Report	1
19	Document Pre-study (minus alternative solutions)	Report	1
21	Document Architecture: Create architectural views	Report	2
38	Document Requirements	Report	1

Figure 12: The epics brought into sprint backlog for Sprint 1.

The customer approved the suggested sprint backlog. The report-related epics were converted into issues in the report Kanban board, while the rest were delegated in the sprint board. The epics in Sprint 1 mainly consisted of documentation of choices regarding the solution, in addition to overall architectural planning and research in order to make those decisions.

7.2.2 Sprint Review

Demo for customer First, the customer was presented with a review of the product backlog to be able to see if the team were in accordance with the expected result for the sprint and to get an update of the progress. Then, the customer was presented with the suggested architectural solution for the technical project, which included a set of UML diagrams that can be seen in Section 6. The team also presented the initial design suggestions in paper format and gave the customer the chance to comment on them.

Progress and unfinished work The burn-down chart in Figure 13 shows the progress of finishing epics during Sprint 1 and reflects the misjudgment in time estimates.

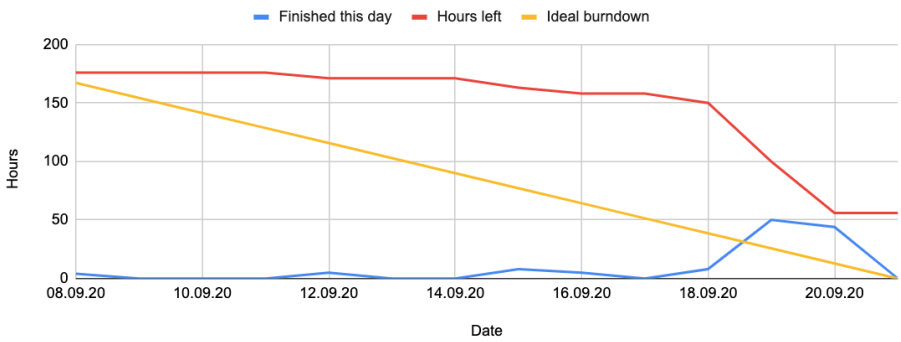


Figure 13: Burn-down chart for Sprint 1.

Six out of nine epics from the sprint backlog were finished. As for the unfinished epics, the team had intentionally postponed E13 due to the discovery of the relation this epic had to the actual code that was not implemented yet. The team therefore decided, in cooperation with the customer, for this epic to be shifted onto the first development sprint. The two remaining epics were report related documentation (E19 and E38) and did not affect the solution specifically. The reason for these epics not to be finished was a huge underestimate of the time needed to obtain detailed insight required in order to make architectural decisions. The team had a harder time defining the scope of the expected solution than estimated, which prompted us to acquire more information and answers from the customer.

Feedback from the customer The customer was satisfied with both the architectural design suggestion and the design and had no further comments. They were impressed with the team's understanding of the scope, considering the fact that it is based on a niche field.

7.2.3 Sprint Retrospective

Table 13 presents the outcome of the retrospective meeting after Sprint 1, which was done similarly to the retrospective meeting for Sprint 0. In the discussion at this meeting, the team concluded that a huge improvement in the distribution of tasks had been made, and the team had felt an improved opportunity to work on their own. The Kanban board and burn-down chart helped us track the progression during the sprint and the fact that the sprint was a week longer than the previous gave more continuity. Generally, the work hours were more effective both together and separately, and a lot more outcome was produced during this sprint. Despite these improvements, and although the burn-down chart does not reflect that all the report sections were started on, there was still a lot of unfinished work in the backlog. Some of it can be explained by an increase in non-backlog related work such as a whole day dedicated to reviewing report sections without having a good system for documenting the work. The team also experienced that the more demanding epics were hard to estimate correctly, and struggled with defining and estimating the tasks well enough without having a deep understanding of the solution yet. This resulted in a still unclear definition of done, and trouble finishing one task at a time. Unfortunately, a by-product of the increased amount of individual work was a more sloppy effort on hour registration, which has to be improved in further sprints.

Table 13: Retrospective analysis for Sprint 1.

What went well	**What could be improved**
• Everyone is taking responsibility for their role-related tasks and the distribution of tasks and delegation flow works well. We trust each other's effort and work • The use of Kanban boards is a huge improvement from the previous sprint and the tasks are now distributed in a good way so that everyone knows what to do at all times and can work independently during the week. • We have come a far way with the report due to good structure from the responsible and great effort from the team. We also held a great review-meeting to read through each other's parts. • Agenda for meetings and days works great. Everyone knows what to expect for the day and are able to distribute their own time better. • The group dynamic is perfect; no discussions or bad mood and everyone is in team spirit.	1. People are not finishing a task they started on according to expectations. We are missing a clear definition of done. 2. There are too many tasks in circulation at the same time because we are not good at prioritizing tasks and finishing one before starting another. 3. We are not following the PPP template during stand-up in an optimal way and it leads to some unnecessary discussions and waste of time during stand-up. 4. Limited contact with the customer this sprint because they were not at work. Even though we knew about it prior and prepared, it is still worth mentioning. 5. Bad hour registration in the backlog because the setup requires double registration in both time sheet and backlog. That also makes it hard to follow up on what is started on and finished, especially on the project planning and architectural tasks.

Action points Table 14 shows the selected notes and corresponding action points and the team member assigned as responsible for fulfilling it.

Table 14: Action points created from retrospective meeting Sprint 1.

Chosen Note	**Action Point**	**Responsible team member**
1	Write down definitions of done for report work and tech issues to have a common understanding. Responsible writes a checklist on each issue describing the expected requirements to be done.	@tech leads and @report responsible for respective definitions of done and boards
1	Implement code/write freeze before the last team meeting in a sprint to be able to review and fix up during the last common day instead of finishing work.	@scrum master and @team leader planning freeze
2	Be harder in prioritization of issues and assign them according to the priority in the backlog. If an epic is finished, ask to contribute on another high priority before starting on a lower priority.	@everyone
5	Add functionality for automatic updates of hours in the backlog. Make sure the Kanban boards are always up to date and a trusted source of progress information.	@scrum master on automatic update, @tech leads and @report responsible on board management and @everyone to follow up

7.3 Sprint 2

Sprint 2 was the first out of two development sprints. The team aimed the sprint work towards presenting the customer with the first iteration of a working system, called a minimum viable product (MVP). In this sprint, the team made use of all the Scrum resources, put most of the report work on hold, and dedicated three weeks to software development and UX-design.

7.3.1 Sprint Planning

Sprint goal The sprint goal presented and approved by the customer was as follows: *Implement the design suggestion in Vue and user-test to be able to iterate on the design solution for the final product. Have a product in place that works for matching minimum two existing algorithms.*

Effort estimation Since there were still two epics of 30 hours unfinished from the last sprint, the team did not want to make the same mistake by planning too much work for this sprint. Therefore we made two specific changes in order to make a more accurate estimation; the team decided to decrease the total amount of story points relative to the number of weeks, and to reduce the size of each epic. Naturally, the main load of the epics differed in category compared to the previous sprints. The epics went from being mainly yellow and red, representing report writing and architectural planning, to mainly blue, representing development. The sprint backlog for Sprint 2 is shown in Figure 14.

	Epic name	Category	Priority	Hours estimate
13	Decide and define result-format	Architecture	1	8
39	Make design suggestion in Figma and user-test design	Architecture	1	20
28	Implement design suggestion in Vue	Development	1	15
27	Send and display result using defined result-format	Development	1	5
32	Merge semantic matcher to backend	Development	1	5
25	Allow uploading of source- and target-schemas	Development	1	10
52	Implement all classes in the class diagram	Development	1	5
24	Set up subsumption algorithm	Development	2	10
26	Selection of semantic relations	Development	2	5
53	Clean and refactor semantic-matcher	Development	3	5
20	Document Implementation/sprints (except figures)	Report	1	20
42	Document Appendix for sprint 0 to section 8	Report	2	5
50	Document Appendix for sprint 1 to section 8	Report	2	5
51	Document Appendix for sprint 2 to section 8	Report	3	5

Figure 14: The epics brought into sprint backlog for Sprint 2.

The customer approved the suggested sprint backlog. They were curious about the team's prioritization of support for different file formats, which was defined as an epic in Sprint 3. The team allowed the customer to re-prioritize the epics in order to make this a part of the sprint. The customer declined and specified their trust in the team's priority skills and that they just wanted to make sure there was a plan for this feature.

7.3.2 Sprint Review

Demo for customer First, the customer was presented with a review of the product backlog to be able to see if the team was in line with the expected result for the sprint and to get an update of the progress. Then, the customer was given their first demonstration of a working system, consisting of a functional back-end for matching algorithms and a simple but sensible front-end. The last thing presented to the customer was the results and evaluation of the user-tests done in the sprint. The customer was informed that the results will be used to further improve the design of the solution.

Progress and unfinished work All epics from the sprint backlog were finished according to plan. In the initial sprint goal, the team targeted a back-end with functionality for matching with at least two of the algorithms but ended up having working functionality for all the existing algorithms. The reason for this was that that it did not take as much additional work as expected to include more algorithms after supporting the first two, and the extension was natural to implement in the same branch. In addition to the epics from the sprint backlog, the team finished four epics that were originally planned for Sprint 3. Figure 15 shows the product backlog as it looked during the sprint review.

	Epic name	Category	Priority	Hours estimate	Planned sprint	Started date	Done
13	Decide and define result-format	Architecture	1	8	Sprint 2	29.09.2020	☑
39	Make design suggestion in Figma and user-test design	Architecture	1	20	Sprint 2	24.09.2020	☑
28	Implement design suggestion in Vue	Development	1	15	Sprint 2	30.09.2020	☑
27	Send and display result using defined result-format	Development	1	5	Sprint 2	02.10.2020	☑
32	Merge semantic matcher to backend	Development	1	5	Sprint 2	22.09.2020	☑
25	Allow uploading of source- and target-schemas	Development	1	10	Sprint 2	01.10.2020	☑
52	Implement all classes in the class diagram	Development	1	5	Sprint 2	28.09.2020	☑
24	Set up subsumption algorithm	Development	2	10	Sprint 2	24.09.2020	☑
26	Selection of semantic relations	Development	2	5	Sprint 2	24.09.2020	☑
53	Clean and refactor semantic-matcher	Development	3	5	Sprint 2	29.09.2020	☑
49	Fix Java-warnings in project	Development	3	5	Sprint 2	29.09.2020	☑
30	Downloading of results	Development	1	5	Sprint 2	01.10.2020	☑
41	Testing of system	Development	2	8	Sprint 2	06.10.2020	☐
20	Document Implementation/sprints (except figures)	Report	1	20	Sprint 2	06.10.2020	☑
42	Document Appendix for sprint 0 to section 8	Report	2	5	Sprint 2	08.10.2020	☑
50	Document Appendix for sprint 1 to section 8	Report	2	5	Sprint 2	13.10.2020	☑
51	Document Appendix for sprint 2 to section 8	Report	3	5	Sprint 2	13.10.2020	☑

Figure 15: The backlog as it looked at the end of Sprint 2.

The decision of editing the backlog was made in collaboration with the customer after discovering that we now had underestimated the amount of work to bring into the sprint, and the epics were chosen based on their relation with the ones for this sprint. We also were able to speed up the testing process because the team had a stable enough functionality to start writing tests. The testing was not finished and remained as the only unfinished epic for this sprint in the backlog. This is according to plan because there is still unwritten code to be implemented and tested during Sprint 3, so the epic will be brought along into the next sprint, where it was originally planned.

In total sixteen epics were finished during Sprint 2. The burn-down chart in Figure 16 shows the progress of finishing epics during Sprint 2 and reflects a resulting increase in the time estimate compared to Sprint 1. Although there were still improvements to be made, as the original estimates were an underestimation and adjustments had to be made.

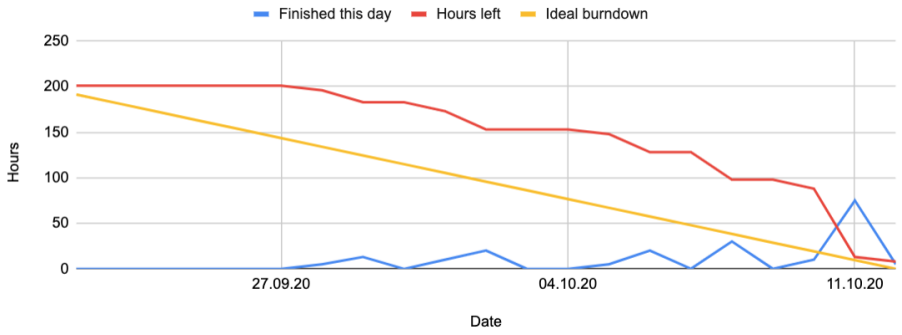


Figure 16: Burn-down chart for Sprint 2.

Feedback from the Customer The customer was happy with the overall demonstration of the system and they agreed on the feedback about the design (see documentation of user tests in Section 9). They were pleased with the format of the result-file and preferred the chosen format regardless if it matched the industry standard format. They had some questions regarding the degree of difficulty of implementing support for more algorithms, and the team confirmed that it will be an easy job due to the code cleaning and generalization of methods that had been done.

7.3.3 Sprint Retrospective

In this sprint, the team had a great advantage from the technical course from the tech leads. Everyone was able to contribute to the technical epics, and the distribution of responsibilities made the collaboration effective. At the end of the sprint, the code became more and more complex, and the team had a discussion during the retrospective whether it was optimal to further include everyone in the system development, or if we would benefit from dividing the backlog categories between the team members in the next sprint. Table 15 presents the outcome of the retrospective meeting for Sprint 2.

Table 15: Retrospective analysis for Sprint 2.

What went well	**What could be improved**
• Communication with the group is going well and the meetings are nice sessions with snacks • Merging and workflow are easy to follow • Division of work went well most of the sprint • Action points from last time were followed up and went well • Tasks well conducted with good resulting product • Work-flow in Github working well, and good reviews from the tech leads there • Everyone is contributing and can be trusted with their work and responsibilities. Meetings are really nice and something to look forward to • Good use of the backlog and it works great as a source of information and communication with the customer	1. It is hard to get more than a superficial understanding of other people's tasks and work done due to low effort use of stand-up and weekly reports. 2. Hard to follow the progress on report writing during the development sprint and lack of action when tasks are set to the review section in the report section because no one is assigned to reviewing or resolving comments in Overleaf. 3. The working week becomes really short when someone is not assigned to an issue between meetings. That results in only having work to do at the common time and not throughout the whole week. 4. Not optimal distribution of tasks at the end of the sprint 5. People being late on registering hours makes it problematic to write weekly reports 6. Complex back-end makes it hard to get a good understanding of the whole project

Although the initial planning resulted in an underestimate of time, the team was able to handle the challenge and restructure the backlog and the plan together with the customer. This flexibility resulted in a well-implemented sprint, despite a violation of Scrum methodology's principle of a locked backlog during sprints. It must be said that the last days of the sprint were hectic, and resulted in some sloppiness in updating each other and registering work-hours. This was discussed during the meeting. The fact that the whole team was able to notice the difference in effectiveness between when the process is followed and not, gave us an assurance that the collaboration in general was working well, and that we actually benefited from following the implementation plan as carefully as possible.

The main challenge this sprint was to optimize the progression tracking between team members. Due to an increase in complexity and a big variety of tasks, the team noticed an increase in the need for good updates from each other. Before developing, most team members were working on similar epics or collaborating, but during this sprint, it was harder to follow up on what was happening in the project in total. Even though everyone was trusted with their work, the team member needed a broader understanding of the progression at all times, so this was the main focus in this sprint's action points.

Action Points Table 16 shows the selected notes, the corresponding action points, and the team member assigned as responsible for fulfilling it.

Table 16: Action points created from retrospective meeting Sprint 2.

Chosen Note	**Action Point**	**Responsible team member**
1 and 2	Everyone is expected to spend some time preparing for the stand-ups and write better weekly reports. At the end of every work day we should write down a summary of what has been done as preparation for the next stand-up.	@everyone
2	Convert report board into issues to be able to assign each other and to require review	@report responsible
3	Never leave a work day without being assigned to a new issue. In that way everyone will always have something to work on in individual work hours.	@everyone

7.4 Sprint 3

Sprint 3 was the second of two development sprints, aimed at presenting the customer with a finished product meeting that met their expectations. The sprint was set up similar to Sprint 2 with three weeks of development, although there was also prioritized some report writing in order to finish the project in time.

7.4.1 Sprint Planning

Sprint goal The sprint goal presented and approved by the customer was as follows: *Extend solution to work with support for more file formats. New design implemented from user-test results.*

Effort estimation After having one sprint of overestimating the amount of work, and one sprint of underestimating, the team aimed to make more accurate time estimates in the last development sprint. The team felt much more confident in understanding the scope of each task after getting some experience and could compare new estimates to actual effort registration from previous epics to make the estimates more accurate. The team included report- and project planning-epics in addition to the development to get an overview of the report, to ensure that we did not underestimate the hours that would be spent writing on the report. Experience said the report-related epics tended to be worked on in pairs, needed

review, and was hard to know the scope of during planning, and therefore often require a lot of hours to conduct. The sprint backlog for Sprint 3 is presented in Figure 17. From Sprint 2, testing was still in progress and had to be finished within this sprint.

Being close to the deadline, the team wanted to increase the efficiency. After the discussion from the last retrospective meeting regarding if every team member should code or not, the team decided on dividing the workload between the team members by categories they specialize in. Two team members were made responsible for the architectural epics and front-end design, three members for back-end related epics, and the last two members for the report. This meant a shift in the technical load onto a few of the team members, while the rest handled the project management and documentation. The choice enabled us to let the members work with what they preferred and were the most skilled at to decrease time spent on each task throughout the sprint. The planned result of this was a higher production than in Sprint 2, which would probably be at the expense of the learning outcome. Another challenge of specializing team members is the lack of common understanding of a part of the project which can result in the team being too dependent on a few team member's knowledge. We therefore decided to specialize, but never have only one team member working alone on a part of the project.

	Epic name	Category	Priority	Hours estimate
45	Find out what more algorithms (file formats) we need	Architecture	2	5
43	Make final design suggestion in Figma based on user tests	Architecture	2	10
29	Set up a server at SINTEF	Architecture	2	30
44	Finalize product, both frontend and backend	Development	1	20
61	Test that the solution follows universal design and make necessary changes	Development	3	16
54	Implement support for more file formats	Development	1	50
47	Implement error-handling (e.g. with broken input files)	Development	2	15
57	Estimate algorithm run-time	Development	2	20
55	Minimize running times	Development	3	11
59	Setup in overleaf and issue board for all report sections for sprint 3 and 4	Project planning	1	10
60	Create Figures for section 8	Report	1	14
52	Document Appendix for sprint 3 to section 8	Report	1	2
33	Document Security	Report	2	25
34	Document Testing	Report	1	35

Figure 17: The epics brought into sprint backlog for Sprint 3.

7.4.2 Sprint Review

As Sprint 3 was the last development sprint, it was also time to handle the project over to the customer. In addition to reviewing the progress from Sprint 3, this review meeting was therefore spent demonstrating for both the customer and a representative from Entur.

Demo for customer and Entur At the project demo, the customer and a representative from Entur were presented with the final product. Instead of only demonstrating the newest features added to the project, the team wanted to show the customer how the solution fulfilled the initial task description, the product requirements, and the user scenarios the customer gave the team at the beginning of the project.

Progress and unfinished work All epics related to the system development were finished this sprint, and the progress is presented in the burn-down chart in Figure 18. Even though it is not reflected in the burn-down chart or the backlog, some changes were made regarding the tasks within the sprint. After discussing with the customer half-way through

the sprint, the team decided to downscale some of the epics, as they did not give a high enough value for the customer for them to want the team to spend time on it. This regarded the estimation of the run time and adding the ability to download the result in another file format. The plan was to carefully estimate the running times, but as we realized that this would not give the system much more value than an approximate estimate, we decided to go with the latter. That means that the estimates implemented were only based on the size of the input file, and did not take other parameters like e.g. the server stability into account. The customer agreed that this was a good enough estimate.

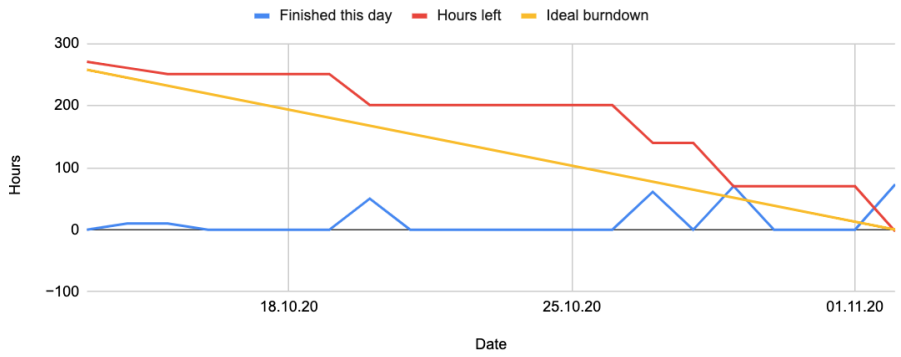


Figure 18: Burn-down chart for Sprint 3.

Feedback from the customer The customer clearly expressed their satisfaction with the system and stated that the team had fulfilled SINTEF Digital's expectations. The team was also pleased to hear that Entur was impressed with the final product. They said that this solution targets a frequent challenge in their daily work and that such a system would be valuable to them. Since Entur has not been directly involved in the project but is an important stakeholder in the ReiseNavet project, this was a confirmation that the solution was relevant for actual use.

7.4.3 Sprint Retrospective

The team all agreed that Sprint 3 was the best executed sprint yet, and all planned work was finished *just in time*, according to Scrum methodology. The collaboration and effectiveness were going better than ever, which is reflected in the amount of work done from the backlog. All the team members noticed an increased flexibility from each other, which probably originates from getting more comfortable with the team and the task. People were able to meet on short notice and took on each other's tasks to help finish them. We succeeded with accurately estimating bigger epics better than we had earlier, as predicted in the planning, because we better understood the scope of the tasks. We handled all challenges and were flexible for the customer, without making a negative impact on the work. Even though a lot of work hours were spent separately during the three weeks of the sprint, the team managed to keep each other updated on the individual work, which was the main action point from the last sprints retrospective meeting. Table 17 presents the outcome of the retrospective meeting.

Table 17: Retrospective analysis for Sprint 3.

What went well	**What could be improved**
• The collaboration went great despite the dividing of responsibilities. Everyone did a great job of updating each other through stand up and weekly reports, and it was easy to track the progress. • Use of the boards was excellent. Everyone assigning themselves to tasks frequently moving them across the board gave a good project overview. • Arranging a review meeting of the report was still a good idea. People left good comments and it was easy to follow up afterwards. • Generally good work morale, and increased flexibility among the team members. When someone has trouble finishing a task, another team member is there in a second to help them at any time. • We have become a good team! Good routines, getting more comfortable with each other after some social activities.	1. Time estimates are still hard to do right. Even though we had experienced a development, it is difficult to be accurate on the work to bring into a sprint, especially in the development sprints. 2. Hard to keep an overview of the report due to increased complexity. Specializing tasks makes the content high quality, but also makes it hard for the others to be critical due to lack of detailed insight in a section. 3. Not seeing the wanted consistency in the report due to the distribution of tasks. Lack of following rules in writing and being consistent with the text.

Since no further development would be conducted after this sprint, a few points were noted to be improved regarding the report writing in Sprint 4. The team struggled with an increased complexity of the report that made it hard to keep an overview of the other team member's sections, and an increased amount of text lead to the need for more common standards and rules to make sure the language was consistent between the authors.

Action points Table 18 shows the selected notes and corresponding action points and the team member assigned as responsible for fulfilling it.

Table 18: Action points created from retrospective meeting Sprint 3.

Chosen Note	**Action Point**	**Responsible team member**
2	Read through the whole report together to get an overview and remove eventual overlaps, leave comments and generally start raising the level	@report responsible
3	Define rules to be followed throughout the report regarding e.g. choice of words and section introduction format	@report responsible and @everyone to follow up

7.5 Sprint 4

Sprint 4 was the last sprint and served as an internal sprint for the team to finish the documentation of the project. The team was given a set of deadlines from the course staff to be taken into account, regarding the delivery of the project and the report. During the project, the team was assigned a date for a final presentation. As the initial project schedule had the last day of the course as the end point, we realized that Sprint 4 would not actually extend over three weeks as we originally planned. The team decided to divide the number of weeks planned for Sprint 4 in two. First a week and a half of implementing what would now be called "Sprint 4", and then after finishing the report, a week would be spent to prepare for the presentation.

7.5.1 Sprint Planning

Sprint goal The goal of the sprint was to finish the report in time. This was a team goal with no need for approval by the customer. The team set internal deadlines to fulfill the goal, e.g. report freeze, due to the success we had with report freeze followed by a review meeting from Sprint 1. The deadlines and schedule for Sprint 4 can be viewed in the detailed project schedule in Appendix B.4.

Effort estimation The customer was not involved in this sprint, and so the team managed the sprint backlog themselves. Because of the decision to include report work in all sprints were well followed up by all team members, a lot of the report were already finished at this point, but still some sections were missing in addition to reviewing and so. All epics this sprint regarded finishing the report, and the sprint backlog for Sprint 4 is shown in Figure 19.

	Epic name	Category	Priority	Hours estimate
35	Document Documentation	Report	2	5
36	Document Evaluation	Report	2	25
37	Document Summary and finish introduction	Report	2	10
53	Document Appendix for sprint 4 to section 8	Report	1	4
23	Document Architecture: Tweak architectural views and document with text	Report	1	20
65	Document unit testing	Report	2	5
63	Review report	Report	3	70

Figure 19: The epics brought into the sprint backlog for Sprint 4.

After prioritizing a review of all the remaining sections in the report as an epic during Sprint 3, the team had a good overview of the work required to finish the report. There was also a "definitions of done" for all sections defined by the report responsible as part of E59 in Sprint 3, which helped us understand the scope and estimate epics this sprint planning. A big share of the hours was dedicated to reviewing the report in total, and we planned for a whole day of review attended by all team members like we had in Sprint 1 as well.

7.5.2 Sprint Review

Progress and unfinished work All epics were finished during Sprint 4, and the report was delivered at the end of it. The burn-down chart in Figure 20 shows the progress of finishing epics during Sprint 4. It reflects the reduction of the implementation time of the

sprint. One can also see the effort made to review the report between internal report freeze November 7th, and delivery deadline November 14th.

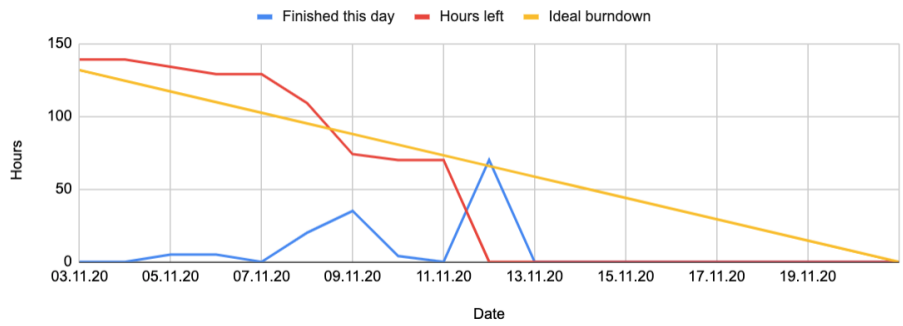


Figure 20: Burn-down chart for Sprint 4.

7.5.3 Sprint Retrospective

Due to stricter national measures in the ongoing pandemic, we were forced to do most of the work remote during Sprint 4. As this was the last sprint of the project and no action points were to be made, the team decided that it would be efficient to have a discussion rather than the post-it activity we usually had at the end of the sprints. We therefore held a remote retrospective meeting where we discussed the implementation of Sprint 4.

The increased amount of remote work turned out to work well, due to the good routines we had acquired during the project. To optimize the stand-up meetings and to have an effective flow of information has been an action point for the team throughout the project, and during Sprint 4 we got rewarded for it. Both team meetings, cooperation in pairs, individual work, and reviewing of the report were conducted efficiently over Discord. The Kanban board for report work was used frequently and worked great as a source of information. There was never an unassigned issue on the board, and in that way we were able to have the expected work finished in time for the deadlines.

The team spent a lot more time than planned on work that was not related to any epic, such as technical adjustments in overleaf, moving around text, or adjusting the appearance of the report. As the sprint ended up lasting barely two weeks, the last days before delivery got a little hectic, and reviewing the whole report required a lot of hours. Luckily the team had held several review meetings in previous sprints and had learned that we needed to estimate a great number of hours to this. We were overall satisfied with the conduction of the plan and were able to finish with perfect margin.

7.6 Final Distribution of Work Hours

Based on the registration of hours and categorization of work in the product backlog, there was generated a visualization of the team's work hours. Figure 21 shows the distribution throughout the project, and clearly reflects the main focus of each sprint. One can see the transition from planning focused sprints in the beginning, over to the two developing sprints, and further onto finishing the report in Sprint 4.

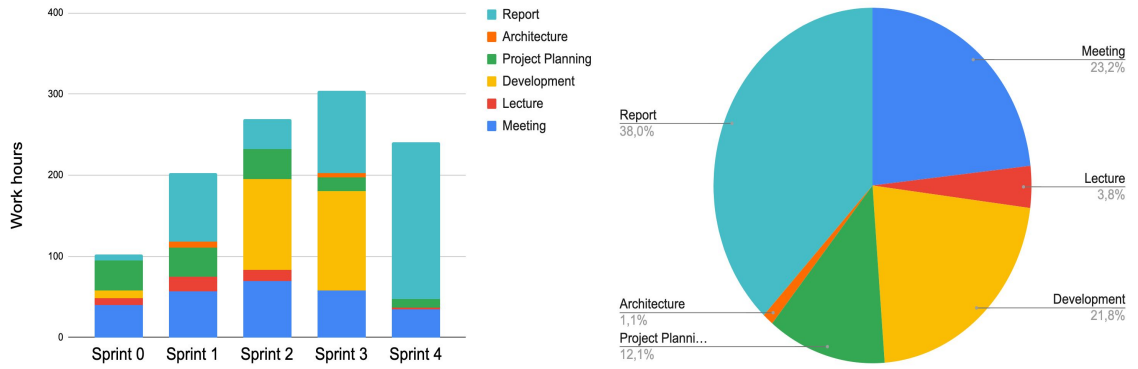


Figure 21: Actual distribution of work hours in the different categories based on time sheets.

Figure 21 can be compared to Figure 2 in Section 2.1 to see if the team spent their work hours as planned. Note that the sprints are of different lengths and that the figures only reflect the hours spent on specific epics from the backlog, in addition to lectures and common team meetings. All hours to be spent on e.g. administrative work throughout the project, non-epic related report work such as the overall design, or technical challenges, are not reflected. The figures therefore gives a somehow incorrect picture of the total distribution of work, but as they are both generated from the product backlog, they give a correct comparison relative to each other. The team considered it as a valuable insight, as the rest of the work hours are presented in the time sheets in Appendix B.3, and were also negligible in amount compared to the work hours defined in the product backlog.

A comparison of the two figures proves that the work hours were distributed quite as planned. Thus, there are some differences caused by either limitations in the registration system or by misjudgments in the planning phase. There is a relatively big difference in hours between Sprint 2 and Sprint 3, knowing that the sprints were of the same length. This is a result of the underestimate of work brought into Sprint 2, where we decided to start the testing of the system earlier than it was originally planned. The relevant epics for this were therefore started the implementation during Sprint 2 but were finished during Sprint 3, and since the figure reflects the number of finished epics, all these hours are registered in Sprint 3. There is also a big difference in the amount of time spent on architecture compared to what was planned. It was hard to define the scope of the architecture related epics, as the team did not know how much research they required. Many of the architectural decisions were also made, or at least discussed, during meetings, which result in the work hours being registered in the meeting category instead. This conflict of how to categorize a work hour correctly most likely applies to the work registered in all categories, which is natural due to the fact that they sometimes overlap. As a result, the meeting category ended up with more hours than planned, where some of them probably includes common work in some of the other categories.

8 Security

Security is important in all aspects of software development and includes both network security and software security. This section gives an overview of the possible threats to be aware of, with regards to the system. It presents the team's threat assessment, in addition to a description of the implemented security measures, and the suggested security measures for future development.

8.1 Threat Assessment

To make sure your software is as invulnerable to potential threats as possible, it is crucial to identify the exposed assets. A threat assessment includes the definition of these and identification of threats associated with them.

Assets An asset can be defined as something of value and that should be protected. Table 19 presents the assets identified for this project and if the asset is defined as an attack surface and which threats it is exposed to. An asset is an attack surface if it can be exploited to carry out a security attack.

Table 19: Security Assets.

ID	**Asset**	**Attack surface**	**Threats exposed to**
A1	Uploaded data		Eavesdropping
A2	API	X	Insufficient Logging and Monitoring
A3	Server	X	DoS-attack
A4	Website UI	X	XXS

8.2 Potential Vulnerabilities and Threats

The system is a single page application with minimal functionality and features. As a consequence, there are some security considerations that do not apply to the system. With this in mind, an evaluation of the system's potential vulnerabilities and threats was made and the following paragraphs describe the threats that were considered relevant to this project. A risk assessment of the potential threats can be found in Table 20.

Eavesdropping An eavesdropping attack is a network layer attack that focuses on capturing small packets from the network transmitted by other computers and reading the data content in search of any type of information [16]. This threat could influence the confidentiality of the solution.

Insufficient Logging and Monitoring Insufficient Logging and Monitoring is on the OWASP API Security Top 10 list from 2019 which lists the top ten security threats APIs face [17]. OWASP is a foundation whose objective is to improve software security. When API logs are not monitored for suspicious activity, it can take a lot of time to notice that

the solution is under attack. This could impact the confidentiality and availability of the solution.

Denial-of-Service attack (DoS attack) DoS is an attack in which the perpetrator seeks to make a service unavailable to its intended user [18]. It is usually accomplished by flooding the target machine with requests in an attempt to overload the system and prevent legitimate user requests. Because the solution is deployed on a single server, it is easily exposed to overloading. This threat could influence the availability of the solution.

Cross-site scripting (XSS) XSS allows attackers to execute scripts in the user's browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites [19]. An attacker could use XSS to fool the user to think it is using the legitimate solution and perform a fake alignment to make the user download a file with malicious content. This threat could impact the reputation and integrity of the solution.

Table 20: Risk assessment of identified threats.

	CONSEQUENCES					
LIKELIHOOD		Negligible (1)	Minor (2)	Moderate (3)	Major (4)	Cata-strophic(5)
	Almost certain (5)	(5)	(10)	(15)	(20)	(25)
	Likely (4)	(4)	(8) **Eavesdropping**	(12) **Insufficient Logging and Monitoring**	(16)	(20)
	Possible (3)	(3)	(6)	(9)	(12)	(15)
	Unlikely (2)	(2)	(4) **XXS**	(6)**DoS**	(8)	(10)
	Rare (1)	(1)	(2)	(3)	(4)	(5)
Risk assessment		Low (1-3)	Moderate(4-7)	High(8-14)	Extreme(15-25)	

8.3 Implemented Security Measures

The data contained in the schema files uploaded on the web page is based on open information and is therefore not defined as sensitive data at this point. The team and the customer had a discussion that concluded to define this data as an asset nevertheless for further development matters. The conclusion was based on the fact that a potential attacker could retrieve this information with the intent to misuse it in a way that the team and customer are not able to foresee at this time in the process. Therefore, the team has implemented a security measure that ensures that the data is deleted from the server after the algorithms are done calculating the alignment.

No further security measures were implemented in the project. The team concluded in discussion with the customer, to spend time identifying and documenting potential threats and vulnerabilities rather than implementing them. The conclusion was based on the fact that the risks were considered too small for the team to prioritize them over functionality during the project. A risk assessment can be found in Figure 20.

8.4 Security Measures for the Future

This section includes a list of security measures for the customer to consider implementing in future development of the solution.

- User authentication and access control
- HTTPS instead of HTTP to avoid eavesdropping
- Captcha to mitigate the chance of DoS
- Functionality to ensure the validity of the schemas uploaded
- Logging and monitoring

9 Testing

This section presents the test plan and gives insight into the tests performed during the project.

9.1 Test plan

During the requirement specification phase, an overall test plan was created. The team decided to mainly use unit testing, user testing, and acceptance testing, as these types of testing were deemed the most appropriate for the system.

Unit testing The purpose of the unit tests is to test specific methods and classes in order to verify that they are behaving as intended. These tests will exclusively target code written in the back-end. The team plans to write the tests using a framework called JUnit, mainly because of its popularity as a testing framework in the Java community. These tests should be carried out during the development phase of the project.

User tests The purpose of the user tests is to assess if the interactions between users and the system are as intended, as well as to point out potential usability issues [20]. When choosing test subjects for a user test it is important to think about who will be the end-users of the system. Students would be the easiest focus group for the team to test on, but the team's ideal focus group requires people with significantly more knowledge about ReiseNavet and its project. Therefore the team had the customer recommend a few people the team could perform user tests on. As a result of a limited focus group, the team will only be able to carry out two user tests.

Acceptance testing The purpose of the acceptance testing is to test if the system behaves as expected and to ensure that the user stories and the requirements are properly addressed by the system. These tests will be carried out during the customer meetings at the end of each sprint.

9.2 Unit Tests

The backend was constructed in a modular way, as shown in Section 6, with classes and methods that could be tested independently of each other. Hence, unit testing was considered a reasonable way of testing the various backend components. Additionally, unit tests provide a way of quickly spotting implementation errors during development, as well as making it easier to change implementation details of specific components later. We choose to use the framework JUnit for creating the tests, and in Appendix A.4 we have included an example of a unit test made in with that framework.

The team did not create any unit tests for the existing code provided by the customer, as this was considered not to be within the scope of the project. Instead, we focused on the implementations the team had made and prioritized testing components related to parsing, selection of algorithms, and running of algorithms.

9.3 User Tests

The team decided that it would be best to perform the user tests during Sprint 2. The reason for performing the user tests so early in the process is that the team would have the time to further improve the design if deemed necessary in upcoming sprints. This approach allowed the team to make use of an iterative design process.

9.3.1 Preparation for User Test

In advance of the user test, the team had to make some preparations. A prototype was made using Figma [21] and the team formulated some tasks the users were supposed to try to complete during the test. The tasks were based on the use-cases formulated by the customer from Section 5. In addition, the team had prepared some questions for the user to answer before and after the test.

9.3.2 Completion of the User Tests

Due to the Covid-19 situation, it was not possible to conduct the user tests optimally. The process had to take place over the internet with help from Microsoft's Teams. The test subjects first got some questions to answer and then some information on why and how the user test would be conducted. The team made it clear to the test subjects that it was important to think aloud. The test subjects received a link to the Figma-prototype and had to share their screen so the team could monitor their interaction with the prototype. Thereafter the test subject was presented with some predefined tasks they had to perform. The actions made and thoughts expressed were copiously made notes of and included in the final evaluation. After the test subjects had performed all of the tasks, they got more questions regarding the prototype and the opportunity to ask questions. When all questions were answered and the test was done, the test subjects had to fill out the System Usability Scale (SUS) questionnaire and send the results back to the team. Questions from the SUS questionnaire and the answers given by the test subjects can be found in Appendix C.1.

9.3.3 Results and Reflections of the User Tests

After finishing the user tests the results were evaluated. The team presented all feedback to the customer and used it to make a new and improved design suggestion.

The most important feedback given was the following:

- It can be unclear that the Confidence result is between 0 - 1 (0 - 100%).
- It can be unclear for the user what the results should be used for.
- It can be unclear that the relations mean/represent.
- It can be unclear what the "Close"-button does. Does it close the pop-up or take the user back to the front page. Suggestion to change it to "Cancel".
- Suggestion to change header "Get Alignment" to "Alignment tool".
- Suggestion to remove header over buttons. The text can be presented on the buttons instead.

- The users found the design simple and easy to use.

A requirement from the customer was a clean and simple design, making the solution easy to use. This was something the test subjects were happy with after the first user test. As the system does not have much complex front-end functionality, and only minor changes were made, we decided not to make time for another round of user testing. Another reason for not doing another round of user testing was the difficulties of finding test subjects in the target audience.

The team experienced the advantage of performing user tests on unbiased people. Even though the solution has limited front-end functionality, the team had not been able to foresee how a person with no knowledge of the solution would perceive it. As a consequence, the team got invaluable feedback from the user tests and improved the solution accordingly. The answers given on the SUS questionnaires from the user tests were calculated into two SUS scores of 100 and 72.5. A score over 80.3 is considered a grade A, and a score over 68 is grade B [22]. Based on the results and further discussion with the customer, the team feels confident having fulfilled requirement BR1 presented in Section 4.

9.4 Acceptance Tests

In this section, the team explains the process behind passing the final acceptance test.

After Sprint 1 the team had made a design suggestion for the customer to review. The customer was presented with a low-fidelity paper prototype and was asked to give feedback on thoughts about the temporary design. The team and customer decided to use the design suggestion as a starting point, but that it needed further improvement to pass an acceptance test.

The team decided along with the customer to have a minimal viable product (MVP) ready at the end of Sprint 2. The MVP included minimal functionality and a simple graphical user interface to be able to execute an alignment. The team therefore made another design prototype in Figma that included all the functionality and views that the final solution would have. The customer gave some feedback, but agreed that the design looked satisfactory. The team and the customer decided that if the final solution had the same look as the Figma-prototype and worked as expected, the solution was ready to be accepted.

The customer was presented with the final solution at the demo at end of the Sprint 3. The team demonstrated all functionalities of the new system as explained in Section 3 and showed that they were implemented according to the requirements explained in Section 5: Requirements. The customer also invited a person from the ReiseNavet-project to look at the final solution. Together they agreed that the final solution behaved as expected and that it met all the specified requirements. The final solution therefore passed the final acceptance test.

10 Evaluation

In this section, the team reflects upon the internal process and result of the project, the communication with the customer and the supervisor, and on the project task. Further work necessary to finish the project is presented, as well as general suggestions for improvement of the course.

10.1 The Internal Process and Results

The team had a common goal to develop a good team dynamic, which we all agreed on was achieved. A big contribution to how this was done, was the team's effort to get to know each other on a personal level. The idea behind becoming better acquainted, both as colleagues and as friends, was that the collaboration would go even smoother. From experience, it is easier to have tough discussions and get through stressful situations if the team members are able to communicate efficiently. Due to the restrictions related to Covid-19, being able to communicate was more important than ever. This lead the team to add an activity on the meeting agenda where the team members talked about what they had done in their spare time since the last meeting. The team also made the efforts to meet outside of working hours to e.g eat dinner together.

Another important contribution to achieving a good team dynamic was a well-defined workshop framework for the retrospective meetings. This workshop facilitated a constructive discussion of achievements and possible improvements, and the definition of tangible action points in order to improve the team dynamic, the project processes, and the product quality. The action points resulted in the challenges being handled along the way so that they never piled up and accumulated into conflicts or bigger problems. The documentation of the sprint retrospective meetings is presented in Section 7, and discusses the challenges that occurred throughout the project. The retrospectives show that the number of improvements required decreases throughout the project. The further out in the project timeline, the more detailed does the action points get, which we interpret as if there were fewer and fewer problems to address. The team both hope and feel, we have been able to show an improvement from a completely new team of students to a well working agile development team through this report.

The assignment of roles made it easy to distribute work without the team leader having to oversee every part of the project. It took some time for the team members to get comfortable in their roles, but it worked great after a while. A frequently discussed dilemma was *how much* we wanted to take advantage of the roles. The trade-off between allowing team members with less experience do a task by spending time learning by doing, opposed to letting an experienced team member solve the task quickly, was discussed in depth during the second retrospective meeting. After conducting sprints as both a generalized and a specialized team, the team gained insight into all aspects required to drive a technical project. With all team members holding a defined role, we were all left with different experiences, whether it was related to leadership, planning and management, technical writing, or development.

A problem the team encountered was how much time was spent filling out the time sheets and similar process documentation. The team devised several documents in Google Sheets to facilitate project management. During the first weeks of the project, these documents were not optimally designed, and the process of filling them out was therefore somewhat tedious. Later, we realized how to properly utilize Google Sheet's features, making it easier

and less labor-intensive to fill out these documents.

In addition, to develop a good work environment, the number one goal for the project was to deliver what the customer wanted. As seen in Appendix D.2 the customer confirmed that the team accomplished this. The team finished all functionality initially requested and presented in Section 4.

Throughout this project, the team has learned a lot about collaboration and the advantages of a good process framework. We have experienced the disadvantages of sloppiness in prioritizing a good information flow within the team, and the importance of spending time on evaluating and improving the collaboration continuously. The team also faced the challenge of working remotely, but found that the process framework was easily transferable to a digital platform. We gained insight into an acknowledged development methodology and learned how the different resources in Scrum could be taken advantage of to optimize the workflow and the collaboration with a customer. Last, but not least, the team has learned about the field of ontology and semantic matching, about new technical frameworks, and the planning and documentation of a system architecture.

10.2 The Customer and the Project Task

The project task was assigned to the team in the first stage of the project, and properly understanding it was necessary to start planning the process. The task was hard to understand at first, containing unknown terms and concepts, in addition to requiring the team to understand and use code developed by the customer. An example is that schema matching is in the academic community considered a complex problem [23], and the development of algorithms for the matching process is a niche field not that well documented online. This lead to a long process of understanding the provided code and decoding the project task. Since the project also involved creating a user interface, some team members could start early on development, while the remaining members researched and got to know the concepts and problems. During the development sprints, the team got a much better understanding of the task and was able to learn a lot, even though much time was spent researching and not producing code.

To better understand the task, and to know what SINTEF Digital needed during the project, good communication with the customer was crucial. From the beginning of the project, a weekly digital meeting was established, giving the team valuable time to ask questions, get feedback, discuss important choices, and update the customer on the progress. As seen in Appendix D.2, the customer thought the meetings were efficient and useful, and was glad we had the meetings each week. SINTEF Digital also expressed that we listened to their opinions. Even though we faced a few technical challenges having only digital meetings, the team felt it worked very well considering the situation.

The meetings were also used for sprint planning and sprint demos, which gave the team valuable feedback and an understanding of whether or not we were on the right track. From Appendix D.2 it is also clear the customer was pleased with the sprint demos and planning. If additional questions came up during the week, the team communicated with the customer via email. This made it easy to ask more technical questions and gave the customer time to give a good answer. The communication was also important to give the team reasonable grounds for decision making based on what the customer wanted. The time of response established at the beginning of the project was followed. During the entire project, the customer was easy to work with and was an important part of the project process by providing feedback, advice, and help when needed. It was interesting to work

with a real customer, and the team agrees the communication with SINTEF Digital made the final system better.

10.3 Supervisors

The supervisor monitored the work progress and helped the team out with questions related to the report. The supervisor and the team had a weekly meeting, for which the team devised a weekly report. The accessibility of guidance ensured that the team worked efficiently throughout the project as the supervisor gave fast replies, and was very easy to work with.

The compendium was clear about how the team should work together with the supervisor, but there was uncertainty around what the team could demand from the supervisor in terms of help. During the weekly team leader meetings, it became apparent that different supervisors had taken different roles. Not being sure on what to expect from the supervisor made it uncertain whether the team could have asked for more help, or the supervisor should have offered it. Instead of referring to reports from previous years, the team would have appreciated more explicit feedback. Additionally, it could be helpful if the supervisor explained why we should do the changes suggested, preferably with respect to the learning goals of the course. The supervisor described report sections as good or acceptable, but it would have been beneficial to receive more detailed feedback. A suggested improvement of this course could therefore be to clearly state what to expect from the supervisor.

10.4 Further Work

While the team was able to create a satisfactory solution that fulfilled the functional requirements made by the customer, there are still improvements that could be made to it. These improvements were not implemented due to time constraints, complexity, or simply not being prioritized by the customer. Table 21 shows each improvement along with a rough estimate of how long the time it would take to implement it.

Table 21: Possible improvements to the system.

Improvement	**Description**	**Time estimate**
Matching	Improving the existing matching algorithms or creating new ones would improve the system overall. For instance, by using BabelNet to create semantic networks of the knowledge base, or by locating the inputs' application area and use external ontologies from the application area to help with the matching.	2-12 weeks
More relations	While the solution is able to find relations "<", ">" and "=", other kinds of relations, like disjunctions or meronymies [24] could also be useful to find.	2-4 weeks
Parallel running	Running each matching algorithm in their own threads would improve the runtime if the system is hosted on a multi-core server.	2-4 days

Parsing	File formats such as NeTEx and IXSI are very complex, therefore some of the data is lost during the parsing. Being able to completely parse all the available data would be beneficial for the matching algorithms.	1-3 weeks
Security	Preventive measures for each of the system's potential vulnerabilities and threats (See Section 8). For instance, implementing HTTPS would prevent eavesdropping attacks.	1-3 weeks
Runtime	Requests with big files can take several minutes to execute. The runtime could be improved by reducing unnecessary function calls or by pruning unlikely matching candidates.	1-2 weeks
Testing	More test coverage and more advanced test classes. The test classes were made simple as they only tested whether the code was able to successfully execute.	1-2 weeks
Loading bar	A loading bar that continuously displays the current progress would improve user satisfaction.	1 weeks

10.5 Suggestions for Improvement

Regarding the learning outcome, this course scores high from the team's perspective. We were all satisfied with the team members and the work environment we were able to create, but we would have preferred that the groups were distributed in a way that students received projects that were relevant for their specialization, for instance, AI-related tasks for AI-students.

As this course requires more than one workday each week it would have been helpful to have more rooms available. It was hard to get hold of a room to work, especially with the rights supplies like a TV. The extra pressure on room reservations due to the pandemic did not help. We think this would be helpful for the supervisor and customer as well because it would increase the quality of meetings for them compared to being forced to only use video-meetings.

Even though some of the lectures were very helpful, some of them came at a late point in the semester, and so the team was already past the discussed topic. We also believe it would be better to make the lectures mandatory only for some of the team members, e.g. the Scrum master for the Scrum lectures, who can bring the relevant information back to the team.

References

[1] Jérôme Euzenat and Pavel Shvaiko. *Ontology Matching*. Springer Science & Business Media, 2013. ISBN 9783662500422.

[2] Alon Halevy. Why your data won't mix: New tools and techniques can help ease the pain of reconciling schemas. *Queue*, 3(8):50–58, October 2005. ISSN 1542-7730. doi: 10.1145/1103822.1103836. URL https://doi.org/10.1145/1103822.1103836.

[3] ST4RT Project. Deliverable d3.3: Mapping between standard and reference ontology. 2017. URL http://www.st4rt.eu/. [Accessed: 14.09.2020].

[4] Pierfrancesco Bellini, Monica Benigni, Riccardo Billero, Paolo Nesi, and Nadia Rauch. Km4city ontology building vs data harvesting and cleaning for smart-city services. *Journal of Visual Languages & Computing*, 25(6):827 – 839, 2014. ISSN 1045-926X. doi: https://doi.org/10.1016/j.jvlc.2014.10.023. URL http://www.sciencedirect.com/science/article/pii/S1045926X14001165. Distributed Multimedia Systems DMS2014 Part I.

[5] IT2Rail Project. Deliverable d1.1: It2rail domain ontology specificiation and repository. 2017. URL http://www.it2rail.eu/Page.aspx?CAT=DELIVERABLES&IdPage=ea7a7af8-9a2b-40d6-bffc-a6effc30a9b9.

[6] I. Sommerville. *Software Engineering*. Pearson Education, 2015. ISBN 9780133943276.

[7] Jodi Lebow. What is agile methodology? n.d. URL https://digital.ai/resources/agile-101/agile-methodologies. [Accessed: 01.09.2020].

[8] Letizia Jaccheri, Anniken Holst, Sofia Papavlasopoulou, Yujie Xing, Jon Atle Gulla, and Orges Cico. *Compendium TDT4290 Customer Driven Project*. The Norwegian University of Science and Technology (NTNU) Faculty of Information Technology and Electrical Engineering (IE) Department of Computer Science (IDI), 2020.

[9] Knut M. Nygaard. What is universal design. 2018. URL http://library.ifla.org/2250/1/094-nygaard-en.pdf. [Accessed 08.11.2020].

[10] Eder Negrete. The vue architecture that worked for me. (in small and large apps), 2019. URL https://medium.com/@edernng/the-vue-architecture-that-worked-for-me-in-small-and-large-apps-9b253cf92951. [Accessed 05.11.2020].

[11] Philippe Kruchten. Architectural blueprints — the "4+1" view model of software architecture, n.d. URL https://www.cs.ubc.ca/~gregor/teaching/papers/4+1view-architecture.pdf. [Accessed 05.11.2020].

[12] Differences between procedural and object oriented programming, n.d. URL https://www.geeksforgeeks.org/differences-between-procedural-and-object-oriented-programming/. [Accessed 12.11.2020].

[13] P. Van-Roy and S. Haridi. *Concepts, Techniques, and Models of Computer Programming*. Mit Press. Prentice-Hall, 2004. ISBN 9780262220699.

[14] Smartcomposition: Bringing component-based software engineering to the web, n.d. URL https://www.researchgate.net/publication/299367855_

SmartComposition_Bringing_Component-Based_Software_Engineering_to_the_Web.

[15] Ivan Montiel. Low coupling, high cohesion, 2018. URL https://medium.com/clarityhub/low-coupling-high-cohesion-3610e35ac4a6. [Accessed 05.11.2020].

[16] J. Kurose and K. Ross. *Computer Networking: A Top-Down Approach, Global Edition.* Pearson Education Limited, 2017. ISBN 9781292153605.

[17] Project api security, owasp, n.d. URL https://owasp.org/www-project-api-security/. [Accessed 02.11.2020].

[18] CloudFlare. What is a denial-of-service (dos) attack?, n.d. URL https://www.cloudflare.com/learning/ddos/glossary/denial-of-service/. [Accessed 10.11.2020].

[19] Owasp project top ten, 2019. URL https://owasp.org/www-project-top-ten/. [Accessed 02.11.2020].

[20] Kate Moran. Usability testing, 2019. URL https://www.nngroup.com/articles/usability-testing-101/. [Accessed 20.10.2020].

[21] Figma. https://www.figma.com/prototyping, n.d. [Accessed 20.10.2020].

[22] Measuring and interpreting system usability scale (sus), n.d. URL https://uiuxtrend.com/measuring-system-usability-scale-sus/. [Accessed 09.11.2020].

[23] Z. Bellahsene, A. Bonifati, and E. Rahm. *Schema Matching and Mapping.* Data-Centric Systems and Applications. Springer Berlin Heidelberg, 2011. ISBN 9783642165184. URL https://link.springer.com/book/10.1007/978-3-642-16518-4.

[24] Claudia Leacock. Using corpus statistics and wordnet relations for sense identification. https://www.researchgate.net/publication/2473330_Using_Corpus_Statistics_and_WordNet_Relations_for_Sense_Identification, 2002. [Accessed 10.11.2020].

Appendices

Appendix A External documentation

A.1 Installation guide

README.md 11/5/2020

Reisenavet Integration-tool frontend

Frontend for the integration tool built for ReiseNavet/SINTEF.

Installation:

- Install Node to get npm (Node Package Manager).
- Install Vue CLI by running `npm install -g @vue/cli`, to run the Vue dev-server.
- Install Firebase CLI by writing `npm install -g firebase` (for hosting of frontend).
    - To publish, you will require access to the firebase project.
- Restart the terminal window.
- Clone the repository into your preferred folder, and `cd` into it.
- Install dependencies by running `npm install`.
- Run `npm run serve` to run a local development server, and you're there!

NOTE: For functionality, the backend must be running as well.

Deployment:

Our system is entirely (both front-end and back-end) hosted on SINTEF's server at ProISP. The server is available on this link (not HTTPS), or on IP `46.250.220.200`

Before deployment:

- Install an SFTP-client
    - Windows: WinSCP
    - Mac/Linux: FileZilla

Deployment:

1. In project-root, run `npm build`. This will build html, css and js-files in a dist-folder (distribution).
2. Connect to SFTP:

- Protocol: `SFTP`
- Hostname/IP: `46.250.220.200`
- Port: `22`
- Username: `student`
- Password: Ask Sebastian or Audun

3. Copy the content of your local dist-folder (the one that was built during step 1) into this the server-folder `/var/www/reisenavet/html`
4. Done! The updated front-end is now available from dataintegrasjon.reisenavet.no.

1 / 1

Reisenavet Integration-tool backend

Supported files

See wiki/Supported_files

Installing

For windows:

- Installer Visual Studio Code
- Installer Java Jdk 14.0.2
- Installer Maven (Binary zip) og sett "apache-maven-3.6.3" mappen i "C:\Program Files"
- Ordne environment variables:
  - Legg til `C:\Program Files\Java\jdk-14.0.2` til environment variable -> "JAVA_HOME"
  - Legg til `C:\Program Files\apache-maven-3.6.3\bin` til environment variable -> "Path"
- Gå inn på repo mappen i Visual Studio Code og velg "Start Debugging" (ikke Run Code)

For mac:

- Installer Visual Studio Code
- Installer Java Jdk 14.0.2
- Installer Maven (Binary zip)
  - Lag en mappe som heter "Maven" i systembiblioteket. Unzip mappen apache-maven-3.6.3 og legg den til i 'Bibliotek/Maven'.
- Skriv i terminal:
  - `~/.bash_profile`
- Lim inn følgende i bash_profile
  - `export JAVA_HOME='/Library/Java/JavaVirtualMachines/jdk-14.jdk/Contents/Home'`
  - `export PATH=/Library/Maven/apache-maven-3.6.3/bin:$PATH`
- Velg exit, lagre og åpne en ny terminal
- Sjekk om Java og Maven er riktige versjoner ved å skrive dette i terminalen:
  - `source ~/.bash_profile`
  - `java --version`
  - `mvn -v` Gå inn på repo mappen i Visual Studio Code og velg "Start Debugging" (ikke Run Code)

Deployment

Our system is entirely (both front-end and back-end) hosted on SINTEF's server at ProISP. The server is available on this link (not HTTPS), or on IP `46.250.220.200`

Before deployment:

- Install an SFTP-client
  - Windows: WinSCP
  - Mac/Linux: FileZilla
- Install an SSH-client

    - Windows: Putty
    - Mac/Linux: Already installed in terminal
- Install Eclipse for java-development

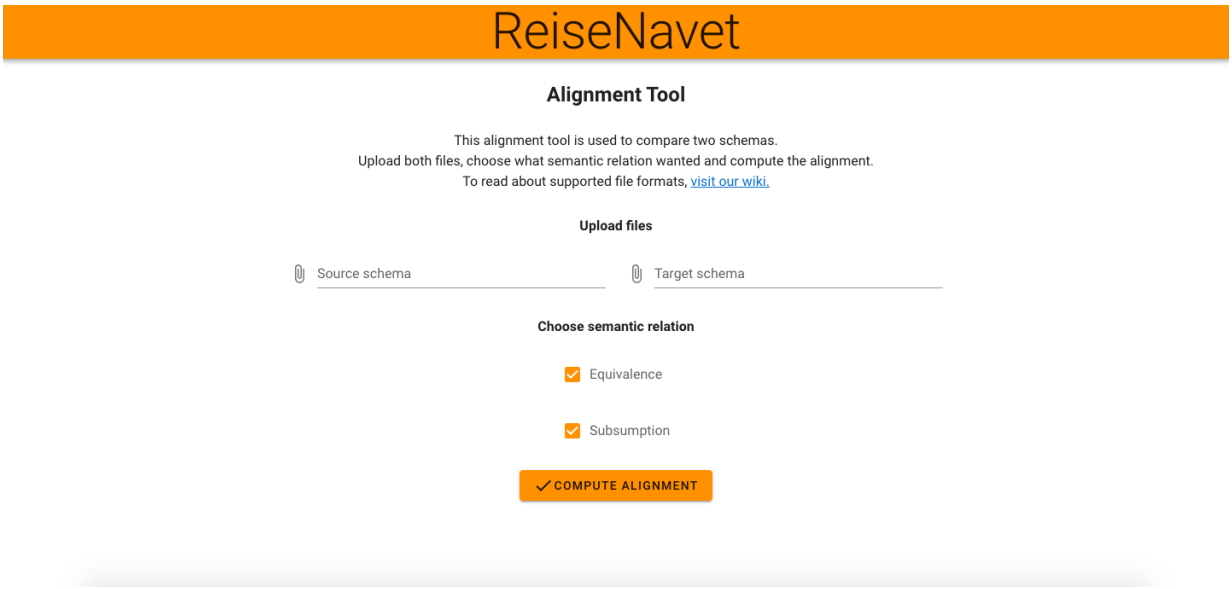
Deployment:

1. Open the project folder in Eclipse.
2. Navigate to `src/main/java/` > `(default package)` > `App.java`
3. Set up a run-configuration
    - Set main class to `App.java`
    - Click `Apply`
    - Close `Run Configurations`-window
4. Right click `App.java` and click `Export...`.
5. Select "Runnable Jar file" and use the following settings:
    - Launch configuration: The one you created before.
    - Export destination: `SemanticMatcher\App.jar` (where `SemanticMatcher` is the name of the project.)
    - Library handling: `Extract required packages into generated JAR`.
6. Click finish to build the App.jar file.
7. Connect to SSH:
    - Windows - Putty:
        - Hostname: `student@46.250.220.200`
        - Port: `22`
        - Connection type: `SSH`
        - Password prompt: Ask Audun
    - Mac/Linux - Terminal:
        - `ssh student@46.250.220.200`
        - Password prompt: Ask Audun
8. SSH: Stop the running java process with `killall java`
9. Connect to SFTP:
    - Protocol: `SFTP`
    - Hostname/IP: `46.250.220.200`
    - Port: `22`
    - Username: `student`
    - Password: Ask Audun
10. SFTP: Copy the compiled `App.jar` into this the server-folder `/home/student`
11. SSH: Restart the java server with `nohup java -jar App.jar &`

(If this doesn't work due to some missing dependency, fix this in eclipse before step 4.)

A.2 User guide

This guide will show how a user can use the alignment tool.



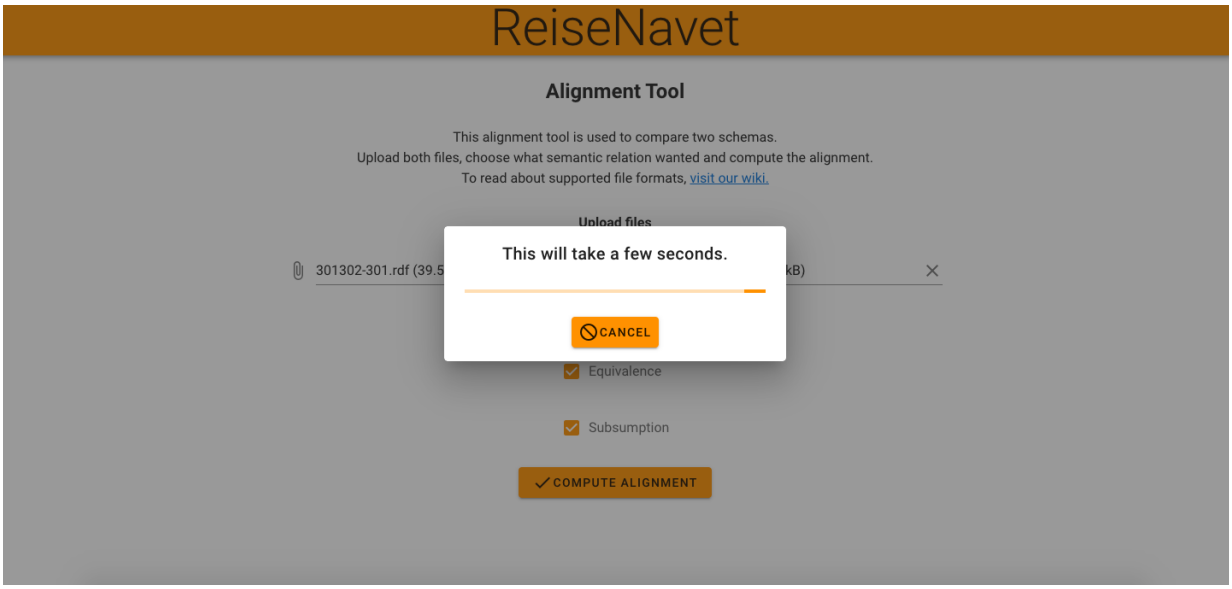
Figur 1: Front page

Upload target and source schemas (files) by pressing the paper clip-icons, or the text next to them. Only supported file formats are possible to choose when uploading. The supported file formats are presented in the wiki-page about supported files. Choose semantic relations (at least one has to be chosen) to decide what the results should include. When files are uploaded and semantic relation is chosen the button "Compute alignment" can be pressed.

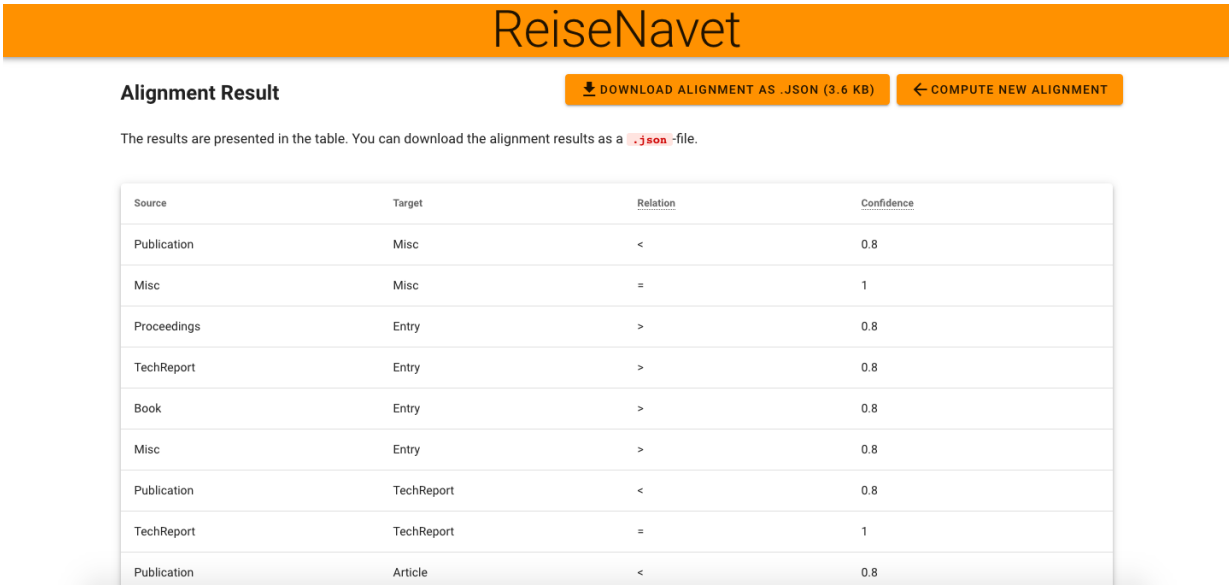
An error will occur either if something is filled out wrong, of if the back-end crashes.

Figur 2: Error message

When waiting for the result, the estimated time left is shown. This is calculated based on the the filesize of the files uploaded.



Figur 3: Loading page



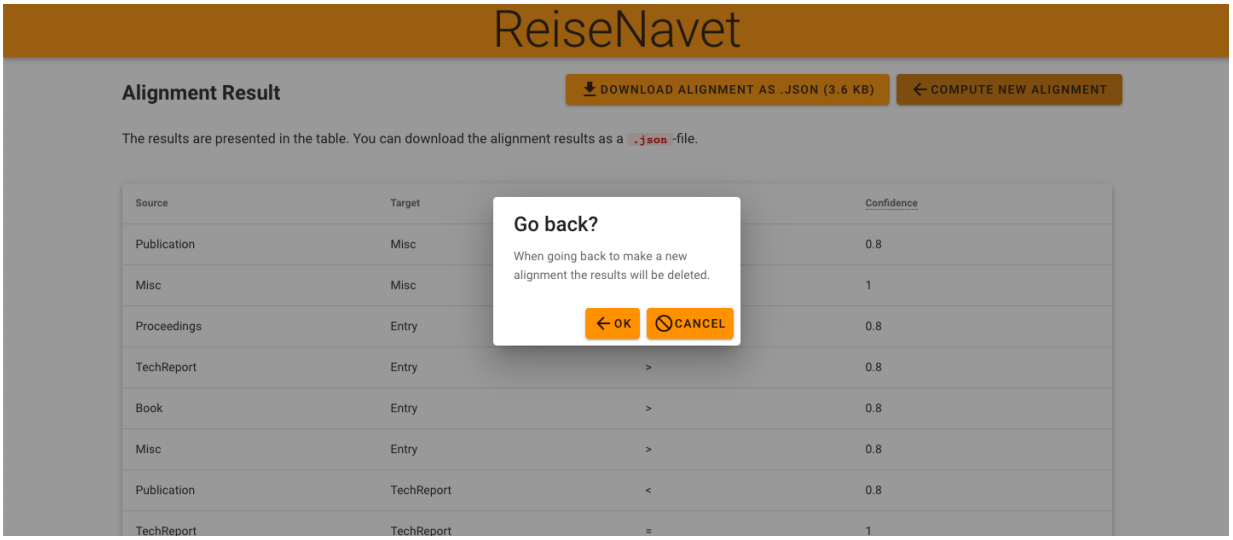
Figur 4: Result page

When the result is computed a table with the alignments is shown. The results can be downloaded as a .JSON file. A new allignment can be done by pressing the "Compute new alignment"-button. This will take the user back to the front page, and delete all results.

Source	Target	Relation	Confidence
Publication	Misc	<	0.8
Misc	Misc	=	1
Proceedings	Entry	>	0.8
TechReport	Entry	>	0.8
Book	Entry	>	0.8
Misc	Entry	>	0.8
Publication	TechReport	<	0.8
TechReport	TechReport	=	1
Publication	Article	<	0.8
Article	Article	=	1

Rows per page: 10 1-10 of 17

Figur 5: Result in table-format



Figur 6: Pop up shown if "Compute new alignment"-button is pressed

A.3 Supported files

The matcher supports .owl, .rdf, .xls/.xlsx, IXSI (.xsd), NeTEx (.zip), GBFS (.zip).
More information about each:

.owl

Any kind of .owl is supported.
See here for more information about .owl

.rdf:

Must be formatted as a RDF Schema

.xls/.xlsx:

The spreadsheet need to be formatet a specific way to be parsed. Look at the image below for an example.

	A	B	C	D
1	**element**	**attributes**	**definition**	**comment**
2	trip		Trips for each route. A trip is a sequence of two or more stops that occur	A comment that won't be parsed, good for documentation
3		route_id	Identifies a route.	
4		service_id	Identifies a set of dates when service is available for one or more routes.	
5		trip_id	Identifies a trip.	
6	stop_time		Times that a vehicle arrives at and departs from stops for each trip.	
7		trip_id	Identifies a trip.	
8		arrival_time	Arrival time at a specific stop for a specific trip on a route. If there are	Another comment
9		departure_time	Departure time from a specific stop for a specific trip on a route. For	
10				

First row is not used in the parsing. It is instead used to describe the columns to the user.
Each other row describes the ontologies and should be formatted this way:

- **Element** The name of the ontology.
- **Attributes** The attributes to the current ontology.
- **Description** Description of the ontology/attribute.
- **Comment** Won't be used in the parsing, good for documentation.

IXSI (.xsd)

The file must be formatted as IXSI to be parsed.
See here for more information about IXSI.

GBFS (.zip)

Zip the GBFS folder (a folder containing all the .json files) and use the zipped file for parsing.
See here for more information about GBFS.

NeTEx (.zip)

Before parsing the NeTEx, you will need to do some fixes/changes to it.

- For each .xsd inside NeTEx/xsd/gml you must change the opening and closing braces from "schema" to "xsd:schema"
- Make sure NeTEx/xsd/siri_utility/siri_types-v2.0.xsd imports xml/xml.xsd from the correct folder. It should say "../wsdl/xml/xml.xsd" and not "../xml/xml.xsd" at the import.
- Inside NeTEx/xsd/netex_service/netex_dataObjectRequest_service.xsd, remove each substitutionGroup="XXX"

After that zip the xsd folder and use the zipped file for parsing.
See here for more information about NeTEx.

A.4 Unit Testing

The code below shows an example of a unit test in the backend. It imports the JUnit pacakge with "org.junit.Test", and the test developer must manually import relevant testing methods, such as assertNull which is used below. For this particular test, the test will succeed if the method *run* does not return null, otherwise it will fail. Also note the annotation "@test", that must precede the test method body.

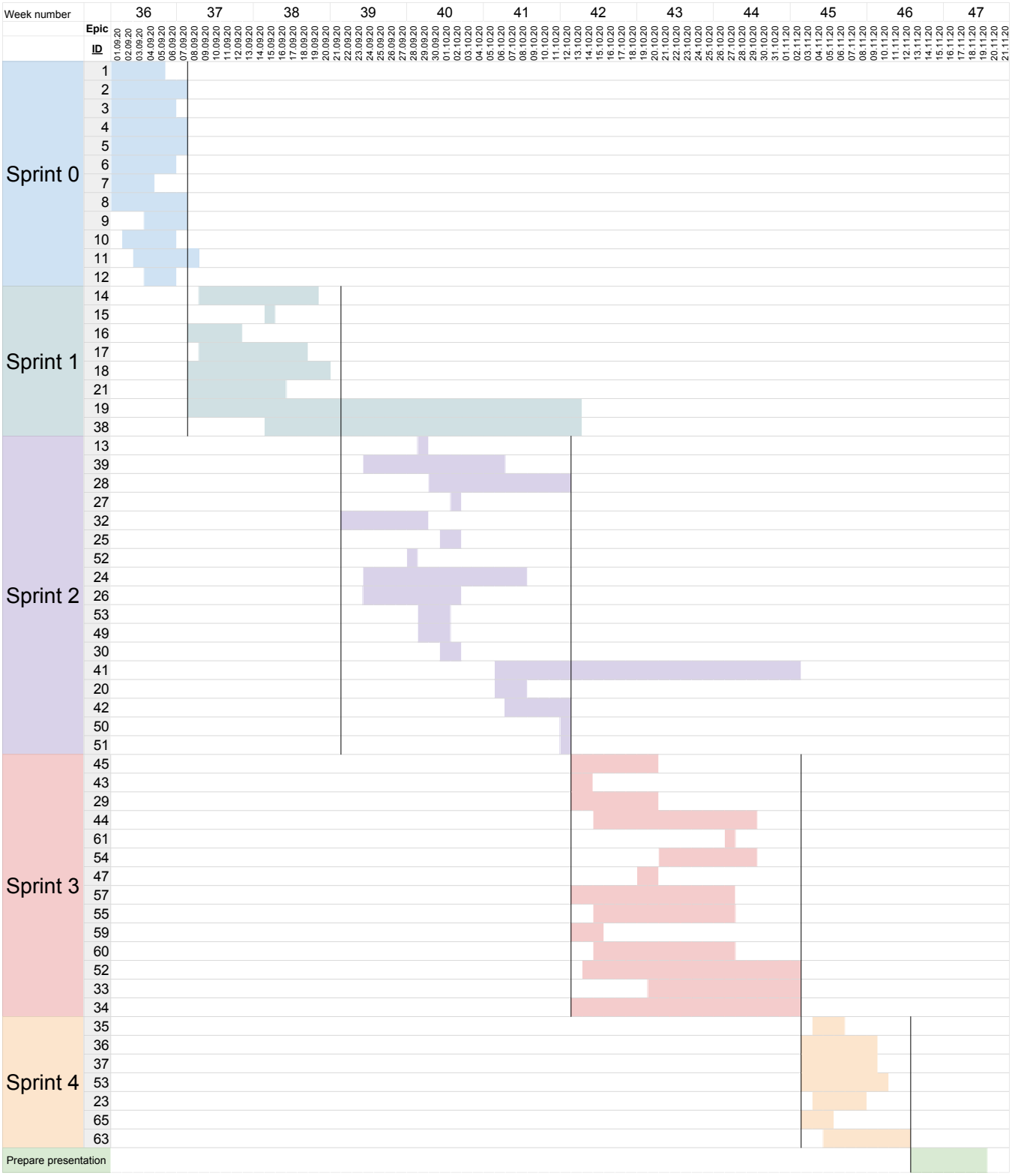
```
import java.io.File;

import org.junit.Test;

import static org.junit.Assert.assertNotNull;

import services.AlgorithmPicker;
import services.AlgorithmRunner;

public class AlignmentCombinerTest {
    private String sourceFileLocation = TestConfig.sourceFileLocation;
    private String targetFileLocation = TestConfig.targetFileLocation;
    private File sourceFile = new File(sourceFileLocation);
    private File targetFile = new File(targetFileLocation);

    @Test
    public void testRun() throws Exception {
        boolean useEquivalence = true;
        boolean useSubsumption = true;

        AlgorithmPicker picker = new AlgorithmPicker();
        AlgorithmRunner runner = new AlgorithmRunner();

        assertNotNull(runner.run(sourceFile, targetFile,
          picker.pickAlgorithms(sourceFile, targetFile,
          useEquivalence, useSubsumption)));
    }

}
```

Appendix B Project Management

B.1 Product Backlog

	Epic name	Category	Priority	Hours estimate	Planned sprint	Started date	Done	Done date	Done in sprint	Hours spent
1	Choose technologies	Architecture	1	8	Sprint 0	01.09.2020	☑	05.09.2020	Sprint 0	7
2	Set up project environments	Development	1	15	Sprint 0	01.09.2020	☑	07.09.2020	Sprint 0	8
3	Write readme for project	Development	1	8	Sprint 0	01.09.2020	☑	06.09.2020	Sprint 0	8
4	Define team contract, roles, meetings etc.	Project planning	1	40	Sprint 0	01.09.2020	☑	07.09.2020	Sprint 0	40
5	Choose development methology	Project planning	1	5	Sprint 0	01.09.2020	☑	07.09.2020	Sprint 0	5
6	Create templates for meetings, documentation, ...	Project planning	1	15	Sprint 0	01.09.2020	☑	06.09.2020	Sprint 0	15
7	Set up google drive and define use and structure	Project planning	1	10	Sprint 0	01.09.2020	☑	04.09.2020	Sprint 0	10
8	Project - and sprint planning (sprint overview)	Project planning	1	10	Sprint 0	01.09.2020	☑	07.09.2020	Sprint 0	10
9	Risk analysis	Project planning	1	5	Sprint 0	05.09.2020	☑	07.09.2020	Sprint 0	6
10	Create report in Overleaf and define structure	Report	1	4	Sprint 0	02.09.2020	☑	06.09.2020	Sprint 0	4
11	Document technology choices (section in planning)	Report	1	4	Sprint 0	03.09.2020	☑	07.09.2020	Sprint 1	6
12	Document Introduction	Report	1	10	Sprint 0	04.09.2020	☑	06.09.2020	Sprint 0	6
14	Define architecture	Architecture	1	50	Sprint 1	09.09.2020	☑	19.09.2020	Sprint 1	38
15	Create design-suggestion	Architecture	2	8	Sprint 1	15.09.2020	☑	15.09.2020	Sprint 1	2
16	Guidelines for coding	Project planning	2	5	Sprint 1	08.09.2020	☑	12.09.2020	Sprint 1	5
17	Document Development Methodology	Report	1	8	Sprint 1	09.09.2020	☑	18.09.2020	Sprint 1	20
18	Document Planning (minus technology choices)	Report	1	44	Sprint 1	08.09.2020	☑	20.09.2020	Sprint 1	22
19	Document Pre-study (minus alternative solutions)	Report	1	30	Sprint 1	08.09.2020	☑	11.10.2020	Sprint 2	19
21	Document Architecture: Create architectural views	Report	2	5	Sprint 1	08.09.2020	☑	16.09.2020	Sprint 1	2
38	Document Requirements	Report	1	30	Sprint 1	15.09.2020	☑	11.10.2020	Sprint 2	9
13	Decide and define result-format	Architecture	1	8	Sprint 2	29.09.2020	☑	29.09.2020	Sprint 2	0,5
39	Make design suggestion in Figma and user-test design	Architecture	1	20	Sprint 2	24.09.2020	☑	06.10.2020	Sprint 2	24
28	Implement design suggestion in Vue	Development	1	15	Sprint 2	30.09.2020	☑	11.10.2020	Sprint 2	11,5
27	Send and display result using defined result-format	Development	1	5	Sprint 2	02.10.2020	☑	02.10.2020	Sprint 2	7,5
32	Merge semantic matcher to backend	Development	1	5	Sprint 2	22.09.2020	☑	29.09.2020	Sprint 2	8,5
25	Allow uploading of source- and target-schemas	Development	1	10	Sprint 2	01.10.2020	☑	02.10.2020	Sprint 2	8
52	Implement all classes in the class diagram	Development	1	5	Sprint 2	28.09.2020	☑	28.09.2020	Sprint 2	4
24	Set up subsumption algorithm	Development	2	10	Sprint 2	24.09.2020	☑	08.10.2020	Sprint 2	15
26	Selection of semantic relations	Development	2	5	Sprint 2	24.09.2020	☑	02.11.2020	Sprint 2	10
53	Clean and refactor semantic-matcher	Development	3	5	Sprint 2	29.09.2020	☑	01.10.2020	Sprint 2	12,5
49	Fix Java-warnings in project	Development	3	5	Sprint 2	29.09.2020	☑	01.10.2020	Sprint 2	4
30	Downloading of results	Development	1	5	Sprint 2	01.10.2020	☑	02.10.2020	Sprint 2	0,5
41	Testing of system	Development	2	8	Sprint 2	06.10.2020	☑	02.11.2020	Sprint 2	8
20	Document Implementation/sprints (except figures)	Report	1	20	Sprint 2	06.10.2020	☑	08.10.2020	Sprint 2	12,5
42	Document Appendix for sprint 0 to section 8	Report	2	5	Sprint 2	08.10.2020	☑	10.10.2020	Sprint 2	3
50	Document Appendix for sprint 1 to section 8	Report	2	5	Sprint 2	13.10.2020	☑	10.10.2020	Sprint 2	3
51	Document Appendix for sprint 2 to section 8	Report	3	5	Sprint 2	13.10.2020	☑	12.10.2020	Sprint 2	3
45	Find out what more algorithms (file formats) we need	Architecture	2	5	Sprint 3	13.10.2020	☑	20.10.2020	Sprint 3	2
43	Make final design suggestion in Figma based on user tests	Architecture	2	10	Sprint 3	13.10.2020	☑	14.10.2020	Sprint 3	3
29	Set up a server at SINTEF	Architecture	2	30	Sprint 3	13.10.2020	☑	20.10.2020	Sprint 3	16,5
44	Finalize product, both frontend and backend	Development	1	20	Sprint 3	15.10.2020	☑	29.10.2020	Sprint 3	20
61	Implement universal design	Development	3	16	Sprint 3	27.10.2020	☑	27.10.2020	Sprint 3	3,5
54	Implement support for more file formats	Development	1	50	Sprint 3	21.10.2020	☑	29.10.2020	Sprint 3	54
47	Implement error-handling (e.g. with broken input files)	Development	2	15	Sprint 3	19.10.2020	☑	20.10.2020	Sprint 3	11
57	Estimate algorithm run-time	Development	2	20	Sprint 3	13.10.2020	☑	27.10.2020	Sprint 3	4
55	Minimize running times	Development	3	11	Sprint 3	15.10.2020	☑	27.10.2020	Sprint 3	6
59	Setup in overleaf and issue board for all remaining report sections	Project planning	1	10	Sprint 3	13.10.2020	☑	15.10.2020	Sprint 3	5
60	Create Figures for section 8	Report	1	14	Sprint 3	15.10.2020	☑	27.10.2020	Sprint 3	25
52	Document Appendix for sprint 3 to section 8	Report	1	2	Sprint 3	14.10.2020	☑	02.11.2020	Sprint 3	4
33	Document Security	Report	2	25	Sprint 3	20.10.2020	☑	02.11.2020	Sprint 3	21,5
34	Document Testing	Report	1	35	Sprint 3	12.10.2020	☑	02.11.2020	Sprint 3	16
35	Document Documentation	Report	2	5	Sprint 4	04.11.2020	☑	06.11.2020	Sprint 4	1,5
36	Document Evaluation	Report	2	25	Sprint 4	03.11.2020	☑	09.11.2020	Sprint 4	31
37	Document Summary and finish introduction	Report	2	10	Sprint 4	03.11.2020	☑	09.11.2020	Sprint 4	4
53	Document Appendix for sprint 4 to section 8	Report	1	4	Sprint 4	03.11.2020	☑	10.11.2020	Sprint 4	12,5
23	Document Architecture: Tweak architectural views and document	Report	1	20	Sprint 4	04.11.2020	☑	08.11.2020	Sprint 4	15
65	Document unit testing	Report	2	5	Sprint 4	03.11.2020	☑	05.11.2020	Sprint 4	5
63	Review report	Report	3	70	Sprint 4	05.11.2020	☑	12.11.2020	Sprint 4	107,9

B.2 Gantt Chart of Epics Implementation Time



B.3 Time sheets

Date	Hours							Total hours / day
	Regine	Åsne	Sebastian	Elise	Eivind R	Eivind N	Markus	
01.09.2020	7	8	6	4,5	8,5	10,5	8	52,5
02.09.2020	2	2	2,5	2,5	4	3	3	19
03.09.2020	1	3	3,5	3,5	3	2,5	3	19,5
04.09.2020	0	2	0	0	0	0	0	2
05.09.2020	0	0	0	0	0	0	3	3
06.09.2020	1,5	0	0	0	0	0	0	1,5
07.09.2020	5	0	0	0	0	0	0	5
08.09.2020	8	8	8,5	5	8	6	6	49,5
09.09.2020	0	2	0	2	5,5	0	2	11,5
10.09.2020	5	7	7	3	3	3	3	31
11.09.2020	0	0	0	0	0	0	0	0
12.09.2020	0	0	0	0	0	0	3	3
13.09.2020	0,5	0	0	0	0	0	0	0,5
14.09.2020	4	1,3	2	0	0	0	0	7,3
15.09.2020	3	5,5	7	4,5	8	6,5	3	37,5
16.09.2020	5	2	2	2	2	0	2	15
17.09.2020	4	0	0	3	3	2,5	1	13,5
18.09.2020	0	0	0	2	0	0	0	2
19.09.2020	0	4	0	0	0	0	0	4
20.09.2020	2	0	0	0	0	6	3	11
21.09.2020	3	2,5	0	2	4,5	2,5	2	16,5
22.09.2020	8	8	9	4,5	9,5	4	8	51
23.09.2020	0	2,5	0	3	6	1	0	12,5
24.09.2020	3	3	5,5	3	5	5	2	26,5
25.09.2020	0	0	0	0	0	0	5	5
26.09.2020	0	0	0	0	0	0	0	0
27.09.2020	1	0	0	0	0	0	0	1
28.09.2020	4	0,5	0	0	0	4	0	8,5
29.09.2020	6	7,5	9	4	8	8	4	46,5
30.09.2020	0	4	1,5	2	0	0	2	9,5
01.10.2020	0	2	3	3	5	3,5	4	20,5
02.10.2020	0	1,5	3,5	0,5	0	1	0	6,5
03.10.2020	0	0	0	0	0	0	0	0
04.10.2020	0	0	0	0	0	0	0	0
05.10.2020	0	0	0	2	1	0	0	3
06.10.2020	7,5	6,5	7	5	7	9	4	46
07.10.2020	3	1	0	0	1	0	1	6
08.10.2020	3	4	3	3	3	3	2	21
09.10.2020	0	0	0	0	0	0	0	0
10.10.2020	0	0	0	0	0	0	0	0
11.10.2020	0	0	0	0	0	0	0	0
12.10.2020	0	0	1,5	3	1,5	0	0	6
13.10.2020	9	8	7	5	5	7	6	47
14.10.2020	0	0	0	0	0	0	2	2
15.10.2020	3	3	7	3	3	3,5	2	24,5
16.10.2020	0	0	0	0	0	0	0	0
17.10.2020	0	0	0	0	0	0	0	0
18.10.2020	0	0	0	0	1	2	0	3
19.10.2020	0	0	0	0	0	0	0	0
20.10.2020	8	7	7	5	6	5	5	43

Date	Hours							Total hours / day
	Regine	Åsne	Sebastian	Elise	Eivind R	Eivind N	Markus	
21.10.2020	0	0	0	0	0	0	0	0
22.10.2020	4,5	3	3	3	6	5	2	26,5
23.10.2020	0	0	0	2	0	0	0	2
24.10.2020	0	3,5	0	0	0	0	0	3,5
25.10.2020	0	0	0	0	1	4,5	4	9,5
26.10.2020	1	0	0	3	0	3	0	7
27.10.2020	8	10	8,5	9	4,5	12	5	57
28.10.2020	1	0	3,5	3	1	6	0	14,5
29.10.2020	8	4	7	4	3	6,5	6	38,5
30.10.2020	0	0	0	0	0	0	0	0
31.10.2020	0	0	0	0	0	0	0	0
01.11.2020	0	0	2	0	1	4	0	7
02.11.2020	6	1	3,5	2	1,5	3,5	2	19,5
03.11.2020	10	6	7	7	7	6	6	49
04.11.2020	1,5	3,5	1	0	0	3	0	9
05.11.2020	2	4	7,5	4	3	3	3	26,5
06.11.2020	5	0	0	0	0	0	0	5
07.11.2020	0	0	0	2	0	0	0	2
08.11.2020	0,5	1	2	6	0,5	2	3	15
09.11.2020	0	1	0	1	0	1	0	3
10.11.2020	7,5	6	7	8	8	8	3	47,5
11.11.2020	0	2	3	2	0	1,5	0	8,5
12.11.2020	8	6	8	7	1,4	5	8	43,4
13.11.2020	3	2	4	4	0	0	8	21
14.11.2020	0	0	0	0	0	0	8	8
15.11.2020	0	0	0	0	0	0	0	0
16.11.2020	0	0	0	0	0	0	0	0
17.11.2020	0	0	0	0	0	0	0	0
18.11.2020	0	0	0	0	0	0	0	0
19.11.2020	0	0	0	0	0	0	0	0
20.11.2020	0	0	0	0	0	0	0	0
21.11.2020	0	0	0	0	0	0	0	0
Totalt	172,5	158,8	169,5	147	149,4	172,5	147	1116,7

B.4 Detailed Project Schedule

Practical			Mandatory activities		Planning	
Week #	Sprint #	Date	Meetings	Deadlines	Sprint name	Sprint goal
36	0	01.09.20	Sprint planning		Project intro	First draft of backlog, document choice of technologies and plan the pre-study
		02.09.20				
		03.09.20	Stand-up			
		04.09.20				
		05.09.20				
		06.09.20				
		07.09.20				
37	1	08.09.20	Sprint review, Stand-up and Sprint planning	Sprint retrospective	Pre-study	Define architecture, define result format and create a design suggestion
		09.09.20				
		10.09.20	Stand-up			
		11.09.20				
		12.09.20				
		13.09.20				
		14.09.20				
38		15.09.20	Stand-up			
		16.09.20				
		17.09.20	Stand-up			
		18.09.20				
		19.09.20				
		20.09.20		Report freeze		
		21.09.20		Report review		
39	2	22.09.20	Sprint review, Stand-up and Sprint planning	Sprint retrospective	MVP 1	General design i Vue, product that works for matching minimum two existing algorithms. User-test design
		23.09.20				
		24.09.20	Stand-up			
		25.09.20				
		26.09.20				
		27.09.20				
		28.09.20				
40		29.09.20	Stand-up			
		30.09.20				
		01.10.20	Stand-up			
		02.10.20				
		03.10.20				
		04.10.20				
		05.10.20				
41		06.10.20	Stand-up			
		07.10.20				
		08.10.20	Stand-up	Code freeze		
		09.10.20				
		10.10.20				
		11.10.20				
		12.10.20				
42	3	13.10.20	Sprint review, Stand-up and Sprint planning	Sprint retrospective	MVP 2	Extend solution to work with support for more file formats. New design implemented from user-test results
		14.10.20				
		15.10.20	Stand-up			
		16.10.20				
		17.10.20				
		18.10.20				
		19.10.20				
43		20.10.20	Stand-up			
		21.10.20				
		22.10.20	Stand-up			
		23.10.20				
		24.10.20				
		25.10.20				
		26.10.20				
44		27.10.20	Stand-up			
		28.10.20		Report freeze		
		29.10.20	Stand-up	Report review		
		30.10.20				
		31.10.20		Final frature freeze		
		01.11.20				
		02.11.20		Final code freeze		
45	4	03.11.20	Sprint review, Stand-up and Sprint planning	Sprint retrospective	Finishing up	Finish the report in time
		04.11.20				
		05.11.20	Stand-up			
		06.11.20				
		07.11.20				
		08.11.20		Final report freeze		
		09.11.20				
46		10.11.20	Stand-up	Report review whole report		
		11.11.20				
		12.11.20	Stand-up	Report textural fix freeze		
		13.11.20		Report design fix freeze		
		14.11.20		Report delivery		
		15.11.20				
		16.11.20		Presentation content freeze		
47		17.11.20	Stand-up			
		18.11.20				
		19.11.20	Stand-up	Project presentation and code delivery		
		20.11.20				
		21.11.20				

Appendix C Internal documents

C.1 SUS Questionnaire

PARTICIPANT NAME: _____ DATE: _____

System Usability Scale

For each of the following statements, please mark one box that best describes your reactions to the solution today.

		Strongly disagree				Strongly agree
1.	I think that I would like to use the solution frequently.	☐ 1	☐ 2	☐ 3	☐ 4	☐ 5
2.	I found the solution unnecessarily complex.	☐ 1	☐ 2	☐ 3	☐ 4	☐ 5
3.	I thought the solution was easy to use.	☐ 1	☐ 2	☐ 3	☐ 4	☐ 5
4.	I think that I would need the support of a technical person to be able to use the solution.	☐ 1	☐ 2	☐ 3	☐ 4	☐ 5
5.	I found the various functions in the solution were well integrated.	☐ 1	☐ 2	☐ 3	☐ 4	☐ 5
6.	I thought there was too much inconsistency in the solution.	☐ 1	☐ 2	☐ 3	☐ 4	☐ 5
7.	I would imagine that most people would learn to use the solution very quickly.	☐ 1	☐ 2	☐ 3	☐ 4	☐ 5
8.	I found the solution very cumbersome (awkward) to use.	☐ 1	☐ 2	☐ 3	☐ 4	☐ 5
9.	I felt very confident using the solution.	☐ 1	☐ 2	☐ 3	☐ 4	☐ 5
10.	I needed to learn a lot of things before I could get going with the solution.	☐ 1	☐ 2	☐ 3	☐ 4	☐ 5

Comments (optional):

Question ID	User test 1	User test 2	Average Score
1	5	3	4
2	1	2	1,5
3	5	4	4,5
4	1	1	1
5	5	3	4
6	1	3	2
7	5	4	4,5
8	1	1	1
9	5	3	4
10	1	1	1

Figure 22: Results from the SUS questionnaires.

C.2 Team contract

Gruppekontrakt i Kundestyrt

Gruppe

Gruppe 5

Medlemmer

Regine Ruud
Elise Nordal
Eivind Rebnord
Eivind Nilsen
Markus Opdahl
Sebastian Vittersø
Åsne Stige

Roller

Leder - Åsne
Nestleder - Eivind R
ScrumMaster - Regine
Ansvarlig for rapport - Markus
Sekretær - Elise
Tech ansvarlig - Sebastian og Eivind N

Full oversikt over roller finnes her

Møtetider

Tirsdag kl. 08:15 - 16:00 rom A4-144

- Møte med veileder kl. 10:00 - 10:30
- Weekly standup kl. 10:30 - 11:00
- Møte med kunde kl. 15:15 - 16:00

Torsdag kl. 12:15 - 18:00 (14.15 - 18:00 rom GK1)

Kommunikasjon

Hvor skal kommunikasjon foregå (Slack, Messenger, Mail)?

Messenger dersom man er for sent, eller casual meldinger.
Slack for faglig kommunikasjon.

Hva er svartid på Slack og Mail?

Mail - 48 timer
Slack - Ha på varsler i arbeidstiden (8 - 16). Svare minst en gang om dagen. Tagg personer dersom man trenger svar med en gang.

Erfaring

Hva kan du bidra med (Tidligere erfaringer / teknologier) og hvordan ønsker du å jobbe?

Regine Ruud

- Verv i webkomitéen på kjemi i 1 år før jeg byttet linje. Nettside i django hvor jeg har bidratt i det lille og lært litt python, django og html for det meste.
- Sommerjobb 2020: full stack webapplikasjon med Enonic XP, javascript, React TS og redux, litt SQL.
- Administrativt verv så kanskje litt rundt prosjekt, samarbeid osv. i tillegg til teknisk?

Liker å jobbe i klare rammer og med regler og retningslinjer man kan forholde seg til. Forutsigbarhet og planlegging over spontanitet og frihet for meg personlig. Heller bruke tid på å planlegge godt enn å kaste seg ut i alt. Sitter gjerne sammen, men godt å ha på headset og konsentrere seg alene.

Elise Nordal

- Django fra PU
- React og React Native fra webutvikling
- Sommerjobb 2019: C#, Azure Functions
- Erfaring fra linjeforening og verv

Er avhengig av litt struktur for å få ting gjort. Foretrekker at man gjør avtaler om når man skal møtes og når man skal ha gjort hva. Syns det er mer effektivt å parprogrammere, da jeg ikke er så god på å progge.

Eivind Rebnord

- Jobbet som data manager i et forskningsprosjekt, både deltid og sommer. Har brukt verktøy som VBA i Excel og Stata. Deler av arbeidet dreide seg om å aligne datasett, men var en mer manuell jobb enn det vi står overfor nå.
- Kjenner jQuery og NoSQL fra PU

Sitter gjerne sammen og jobber, men klarer meg også fint alene.

Eivind Nilsen

- Jobbet med C# (.NET) gjennom verv
- Jobbet med C++ gjennom verv

- Har hatt webutvikling (Javascript og React)
- Også brukt Java, Python, SQL, HTML, CSS.
- Litt kjent med tegneverktøy

Markus Opdahl

- Jobbet med django gjennom verv.
- Jobbet med react gjennom sommerjobb.

Sebastian Vittersø

- Hatt egne prosjekter med hosting av SPA på **Firebase** (gratis nettsidehosting, veldig enkelt og greit).
- Jobbet sommer 2019 + deltid 2019/2020 med **Vue** (frontend **js-framework**) og litt **django**.
- Jobbet sommer 2020 + deltid nå med **Laravel** (full-stack **php-framework**).

Åsne Stige

- Sommerjobb der jeg jobbet med frontend. Brukte Vue.
- Lederverv, lært med om administrering, osv.
- Jobbet med React Native i PU, ikke veldig mye erfaring men helt ok.
- Brukt Java Script, Java, HTML og CSS.

Glad i forutsigbarhet, og klare rammer når jeg jobber. Trives best med tydelige mål, en felles forståelse, og fokus på god kommunikasjon. Blir lett distrahert, og veldig glad i å prate.

Forventninger

Hva er målet for faget?

Godt kundeforhold. Lære mye av faget. Hjelpe hverandre.

Arbeidsmengde. Hvor mye vil medlemmene jobbe med faget?

Bruke tirsdager og torsdager på kundestyrt. Jobber mer eller mindre etter behov.

Ambisjonsnivå for gruppen. Hva ønsker gruppen å lære? Hva slags kvalitet ser gruppen for seg å levere i faget?

Bryr oss mer om hva man lærer enn karakter. Ambisjon om iallfall en B.

Oppmøte

Hva skjer om noen ikke møter opp eller ikke gjør det de skal?

Kakestraff og saft om man ikke kommer. Åpne om problemer, vi må hjelpe hverandre.

C.3 Meeting minutes templates

Customer meeting

xx/xx/xx, xx:xx - xx:xx

Not attending:

Questions to customer

-

Agenda

To-Do's til Audun

Group meetings

XX/XX/20, 10:30 - 11:30

Ikke tilstede:

Innsjekk

Kort om hvordan det går. (Refereres ikke)

Daily standup / Runde rundt bordet

(Refereres ikke)

Informasjon

-

Ris og Ros

(Refereres ikke)

TODOs fra forrige møte

- *Lim inn fra forrige møtereferat [ ]*

TODOs til neste møte

-

Dagsorden

-

Appendix D Customer interaction

D.1 User scenarios from the customer

Steg	Beskrivelse	Kommentarer
1	Bruker trykker på knappen "Source schema" og får opp en dialogboks som gjør det mulig å velge en fil fra disk.	Ved å trykke på knappen "Source schema" på nytt kan bruker overskrive forrige valgt fil og velge ny fil fra disk. I et slikt scenario kan "source schema" f.eks. være et egen-utviklet XML schema transporttilbyder X har utviklet og de ønsker å identifisere alle relasjoner mellom sitt format og NeTEx (som representerer "Target schema").
2	Bruker trykker på knappen "Target schema" og får opp en dialogboks som gjør det mulig å velge en fil fra disk.	Ved å trykke på knappen "Target schema" på nytt kan bruker overskrive forrige valgt fil og velge ny fil fra disk.
3	Bruker velger hvilke semantiske relasjoner hun ønsker at dataintegrasjonsverktøyet skal identifisere automatisk. Kan velge en eller begge.	Velges fra hakebokser.
4	Bruker trykker "Compute Alignment" for å aktivisere dataintegrasjonsverktøyet.	Alternativt: Ved å trykke boksen "Clear" nullstilles all input og bruker starter prosessen på nytt (iht steg 1)
5	Når dataintegrasjonsverktøyet har kjørt ferdig, presenteres de identifiserte semantiske relasjonene på skjerm.	Her bør hver identifiserte relasjon presenteres i form av: ***[Source schema concept : Target schema concept : Type of semantic relation : Confidence value]*** Type of semantic relation er enten '=,' '>' eller '<' Confidence value er kalkulert av dataintegrasjonsverktøyet og representerer hvor sikker systemet er på at den identifiserte relasjonen er korrekt.
6	Bruker kan velge å laste ned den produserte alignmenten til disk fra dialogboks som dukker opp når "Download Alignment" er trykket.	Alignment som er nedlastbar bør formateres iht til Alignment Format. Se http://alignapi.gforge.inria.fr/format.html

Figure 23: User scenarios formulated by the customer

D.2 Email correspondence with customer

Gmail **Åsne Stige <asnestige@gmail.com>**

Spørsmål

2 e-poster

Åsne Stige <asnestige@gmail.com> 3. november 2020 kl. 16:50
Til: Audun Vennesland <audun.vennesland@sintef.no>

Som nevnt forrige møte har vi noen spørsmål vi hadde satt pris på om du ville ta deg tiden til å svare på.

1. Hva synes du om kommunikasjonen med oss?
2. Føler du at dine meninger og ønsker til produktet har blitt hørt?
3. Føler du at møtene med oss har vært nyttig?
4. Har det vært nyttig å gjennomføre Sprint demo og Sprint planning for å gi deg oversikt og holde deg informert om fremgangen til prosjektet?
3. Er du fornøyd med produktet, hvis ikke er det noe du synes vi skulle gjort eller prioritert annerledes?
4. Har du noen andre tanker eller kommentarer?

Audun Vennesland <Audun.Vennesland@sintef.no> 4. november 2020 kl. 21:48
Til: Åsne Stige <asnestige@gmail.com>

Hei, se svar under!

Mvh
Audun

> 3. nov. 2020 kl. 17:02 skrev Åsne Stige <asnestige@gmail.com>:
>
>
> Som nevnt forrige møte har vi noen spørsmål vi hadde satt pris på om du ville ta deg tiden til å svare på.
>
> 1. Hva synes du om kommunikasjonen med oss?
Veldig ryddig og bra.
> 2. Føler du at dine meninger og ønsker til produktet har blitt hørt?
Ja, absolutt.
> 3. Føler du at møtene med oss har vært nyttig?
Ja, møtene har vært styrt bra og effektivt. Også veldig fornøyd med at vi avtalte å gjennomføre regelmessige møter.
> 4. Har det vært nyttig å gjennomføre Sprint demo og Sprint planning for å gi deg oversikt og holde deg informert om fremgangen til prosjektet?
Helt klart.
> 3. Er du fornøyd med produktet, hvis ikke er det noe du synes vi skulle gjort eller prioritert annerledes?
Veldig fornøyd med hvordan dere har tatt tak og løst i oppgaven. Føler at alle kravene vi stilte i prosjektbeskrivelsen er innfridd og på en god måte.
> 4. Har du noen andre tanker eller kommentarer?
Nei, tenker svarene over sier sitt.

Appendix E Standards

E.1 Report guidelines

General

- Use present tense except when referring to decisions made in the past
- Don't use the personal pronoun *I*.
- Beware of the use of "this" at the start of the sentences

Sources We will use the natbib package for managing sources. The project contains a separate file named *references.bib*, where you will register your sources. Your sources should conform to the following format:

```
@book{bellahsene2011schema,
  title={Schema Matching and Mapping},
  author={Bellahsene, Z. and Bonifati, A. and Rahm, E.},
  isbn={9783642165184},
  lccn={2011922131},
  series={Data-Centric Systems and Applications},
  url={https://link.springer.com/book/10.1007/978-3-642-16518-4},
  year={2011},
  publisher={Springer Berlin Heidelberg}
}
```

In some cases, not all details need to be specified. For instance, a book might not have a URL. Depending on the source' medium, other formats may be more appropriate. There are some tools that can be used to autogenerate sources, for instance Google Books.

Referencing sources

```
\citep{bellahsene2011schema}
```

This will generate references like this: [23]

E.2 How to code

How to code Java

Follow [Google Java Style Guide](#).

Example class

```java
public class Format {
  static final int NUMBER = 5; // Constant numbers should be static and uppercase
  private boolean condition;
  /*
   * Longer comments
   * are done this way
   */
  public void howToCode(){
    if (condition) {
      // ...
    } else {
      // ...
    }
    MyLambda<String> lambda =
        (String label) -> {
          // ...{
        };
    Predicate<String> predicate = str ->
        longExpressionInvolvingThisIsASuperLongMethodNameTherefore(str);
    switch (input) {
      case 1:
      case 2:
        prepareOneOrTwo();
      case 3:
        handleOneTwoOrThree();
        break;
      default:
        handleLargeNumber(input);
    }
    // Legal way to instantiate arrays
    int[] arrayExample = new int[] {
      0, 1, 2, 3
    };
    // Also legal way to instantiate arrays
    int[] arrayExample2 = new int[] {
      0,
      1,
      2,
      3
    }
  }
}
```

Rules

Indentation

Indent java files with 2 spaces, except 4 spaces when line-wrapping .

Column limit

100 characters

Braces

- No line break before the opening brace.
- Line break after the opening brace.
- Line break before the closing brace.
- Line break after the closing brace, *only if* that brace terminates a statement or terminates the body of a method, constructor, or *named* class. For example, there is *no* line break after the brace if it is followed by else or a comma.

Comments

Single line comments “//” must be placed at the end of a line.
Multi-line /* ... */ comments must be placed at the start of a line and outside of methods, subsequent lines must start with * aligned with the * on the previous line.

Cases in naming

File names and classes are written in `PascalCase`, while functions and variables are written in `camelCase`. Constant variables should be in uppercase.

Horizontal white space

1. Separating any reserved word, such as if, for or catch, from an open parenthesis (() that follows it on that line
2. Before any open curly brace ({), with two exceptions:
    - @SomeAnnotation({a, b}) (no space is used)
    - String[][] x = {{"foo"}}; (no space is required between {{, by item 8 below)
3. On both sides of the double slash (//) that begins an end-of-line comment. Here, multiple spaces are allowed, but not required.

Array instantiation

“Block-like”, look at example class for valid options.

File structure:

The file structure should look something like this:

- src/
  - assets/
    - any_asset_file.jpg
  - main/java/
    - algorithms/
      - [Audun’s semantic matcher algorithms]
    - services/
      - Manager.java
      - AlgorithmRunner.java
    - App.java
  - pom.xml (nesten som package.json)

How to code JavaScript/Vue

Example component

```vue
<template>
  <div class="example-container">
    <SchemaForm :dynamicValue="stateVariable" @click="incrementStateVariable"/>
    <p class="text-muted">{{propNumberOne}}</p>
  </div>
</template>

<script>
import SchemaForm from './SchemaForm'
export default {
  name: 'ExampleComponent',
  components: { SchemaForm },
  props: {
    propNumberOne: {
      type: String,
      required: false,
      default: "My default prop value",
    }
  },
  data: () => ({
    stateVariable: 42,
    externalData: null,
  }),
  computed: {
    stateVariableGreaterThan40() {
      return this.stateVariable > 40
    },
  },
  methods: {
    incrementStateVariable(value) {
      this.stateVariable += value
      this.$emit('customEvent', value)
    },
    async fetchData() {
      const response = await fetch('http://example.com/api')
      this.externalData = await response.json()
    },
  },
}
</script>

<style lang="scss">
 .example-container {
   color: red;
 }
</style>
```

Rules

Indentation

Indent vue files with 2 spaces.

Keyword structure

The keywords (on both the top level [*template, script, style*], and the sub-levels in script [*name, props...*]) should be in the following order:

- `template`
- `script`
    - `name`
    - `props`
    - `data`
    - `computed`
    - `methods`
    - `mounted`
    - `watch`
- `style`

This list might be expanded throughout the project, as there are multiple more keywords (different life-cycle hooks for example)

Semicolons:

Don't use semicolons on the end of code lines. They're not required in JavaScript.

Cases in naming

File names are written in `PascalCase`, while function names are written in `camelCase` except in CSS where it's `kebab-case`.

File structure:

In our little project, *App.vue* is the only view-file, while we will have multiple components referenced from *App.vue*. The file structure should look something like this:

- `src/`
    - `assets/`
        - `any_asset_file.jpg`
    - `components/`
        - `ButtonComponent.vue`
        - `ExampleComponent.vue`
    - `views/`
        - `ViewFile1.vue`
    - `App.vue`

- `main.js`

… and naturally all component files should go in the component folder, and any graphical assets go in the asset folder.
If we end up needing multiple views, the “main view” should be extracted into its own file, into a view folder, and *App.vue* should be handling the navigation.

Tips and tricks

Vue is both quite simple to begin using, and quite advanced to learn completely. While working with it (and googling quite a bit) you might come over some weird syntax or unusual ways to do things. Here are some tips and tricks when using Vue.

Read the docs

Obviously don’t read the entire documentation, but it’s quite nice to have skimmed through some of the top level guides in the *Essentials*-category in the Vue docs (everything before *Components in-depth*).

Some special Vue-syntax

In Vue, much of the syntax is either way too verbose, or a bit cryptic. This might be because it’s doing things differently from other frameworks.

A few examples, also found in the docs.

`<p>{{text}}</p>`
This syntax, often called mustaches, allows us to inject dynamic content in our templates. The content of this paragraph-tag is simply the value of the variable **`text`**.

`<hr :style="styleVariable" />`
In this horizontal ruler-tag, the colon is shorthand for **`v-bind:`**, which means “bind this value to the variable **`styleVariable`**”. Basically: The contents of the style-property should follow the value of the styleVariable.

`<button @click="clickHandler"> Click me! </button>`
The **`@`**-symbol in this button-tag is shorthand for the vue-keyword **`v-on:`**, which replaces the classic javascript **`onclick`**-syntax. The **`clickHandler`** must be defined under methods in the scripts-section of the file.

`<input type="text" v-model="textVariable" />`
The **`v-model`**-keyword has no fancy shorthand, but basically tells Vue to handle all user-input from this element and store it in a variable. Whenever the value of the input-element changes, the variable does too, and whenever the variable value changes, so does the input field.

```
<ExampleComponent
  :numberOfBananas="bananas"
  @crazyEvent="eventHandler"
/>
```
This entire tag contains a whole bunch of new vue-functionality, related to defining our own components.

Firstly, the **`ExampleComponent`** name is not a standard HTML-tag name, so we'll have to import it, and then register it as an external component (see line 9 and 12 in the example component). This allows us to use it in our template as a component. Because of vue's way of registering components, we could actually use either the PascalCase name **`<ExampleComponent/>`** and the kebab-case name **`<example-component/>`** if we'd like.

Secondly, **`numberOfBananas`** is not a standard html tag-attribute, but has to be defined on the **`ExampleComponent`** component as a prop (see line 13-18 in the example component). The **`bananas`**-variable references whatever is defined as bananas on the instance, preferably a data-variable or a computed-function.

In the same way as the last point, **`crazyEvent`** is an event defined in a method in the component. On line 13-18 in the example component we see how to define the event **`crazyEvent`** is emitted every time a method is called. This is emitted to the parent-component, and handled just like **`crazyEvent`**. If **`crazyEvent`** emits a value (which is optional), **`eventHandler`** can accept these as arguments. As previously mentioned, **`eventHandler`** should be a method defined on the instance.

Lastly, we've split the entire tag into multiple lines. For some components we might need to register a whole bunch of event handlers and props, and then we split it up into different lines so the line doesn't end up being too long. We indent the contents of the tag one level inwards (two spaces), but make sure that the tag-ender is indented the same as the tag-opener.

Concrete examples:

Importing data from a .json-file
The file structure in this example is as follows:

```
src/
  data.json
  components/
    ExampleComponent.vue
```

Let's assume data.json and ExampleComponent looks like this:

```json
[
  {
    "name": "John",
    "age": 20,
    "image": "https://i.imgur.com/yJueFD8.jpg"
  },
  {
    "name": "Pete",
    "age": 28,
    "image": "https://i.imgur.com/gLFDdwf.jpg"
  }
]
```

```vue
<template>
  <div>
    <div
      v-for="person in personList"
      :key="person.name"
    >
      <h1>{{person.name}}</h1>
      <p>{{person.age}} years old.</p>
      <img
        :src="person.image"
        alt="Profile picture"
      >
    </div>
  </div>
</template>
<script>
import jsonData from '../data.json'
export default {
  data: () => ({
    personList: jsonData,
  })
}
</script>
```

Explanation: Before `export default`, we import the file from into a variable we call `jsonData`. This is then referenced in the initialization of the data-variable `personList`, making it available to our template.

We're doing something else cool here; The keywords `v-for="person in personList"` do to things:

1. Make the tag repeat as many times as there are objects in the array
2. Bind the current value of the iterator to the children of the tag (basically allowing us to write `person.name` in our template, giving us the value).

E.3 How to process

How to Prosess

Før du starter på en oppgave

1. Teamet definerer oppgavene og har en felles "Definition of Done".
2. Alle tasks er lagt inn i Integrasjonstest-siden på wiki, og har oppgaver har punkter i Tasken som skal integrasjonstestes.
3. Ta på seg en oppgave (en task)Snakk med gruppen før man tar på seg oppgaven, og bli enige om fordeligen. Dersom det er tomt for oppgaver må det settes opp et møte og planlegge hvilke oppgaver som skal prioriteres videre.

Når du starter på en oppgave

1. Assign deg selv til tasken, eller følge tasken dersom man er flere som jobber på den sammen.
2. Flytt tasken til "Development" i taskboardet og gjør det kjent med "Definition of Done" og Integrasjonstest for tasken.
3. Opprett en ny branch for tasken du jobber med utifra Master-branchen. git checkout -b <#2000-Listevisning>

Gjennomføringen av en oppgave

1. Push endringer til oppgavens branch, husk å inkludere navn på task i commit-meldingen slik at den linkes til tasken. git push origin #2000-Listevisning
2. Sjekk av alt du gjør på tasken i "Definition of Done".
3. Test all ny funksjonalitet lokalt ved en komponenttest. Testingen skal loggføres i testplanen.

Ferdig med en oppgave

1. Sjekk av alt er checket av på "Definition of Done".
2. Flytt tasken til "QA" i taskboardet og assign en person.
3. Send inn en pull-request for å merge branchen inn i "Master" når alt er ferdig testet lokalt. Assign en person til å gjennomføre QA.
   git add . git commit -m "Handle user input. #2000"
4. Slett branchen. git push origin --delete
5. Oppdater listen for regresjonstest dersom man legger til ny funksjonalitet git branch -d

Testing av en oppgave

1. Sett tasken til "Testing" i taskboardet og assign en person.
2. Gjennomfør regresjonstest
3. Fyll ut informasjonen i tabellen for testplan etter gjennomført testing.
4. Hvis alt går etter planen, flytt task til Closed. Hvis noe ikke blir godkjent følg fremgangsmåte fra testplan

E.4 How to git

How to Git

Workflow

Vi tenker å følge en workflow som sikkert mange er kjent med som heter GitFlow.

Brancher

Når man jobber med en feature / task så lager man først en branch utifra 'master' med et navn som beskriver delen eller featuren man jobber med. Husk å legge til en referanse i commit-meldingen til hvilken issue det bidrar til i form av en #.

Branch navn: “4-score-board”

`git checkout -b <navn>` der navn er navnet på den nye branchen

`git push --set-upstream origin dev` for å gjøre branchen tilgengelig for alle

Branch: scorelabel. Hvis man har gjort noe som bidrar til en issue nevner man taskens nummer i commit meldingen.

Commit melding: `"add logic to player score. #4"`

Hvis man har lagt til noe som gjør at issuen skal closes kan man skrive i meldingen: `"closes #4"`

Hvordan bruke git

Liten oppfriskning på litt git commands å huske på:

Add & Commit

`git add .` - legger til alle filer som er endret lokalt inn i index. Kan også være lurt å kun adde de filene som faktisk er endret ved å bruke

`git commit -m <melding>` - committer alle filene og endringene som ligger inne i index. (Husk å referere til issue i meldingen med '#Task nummer' som du jobber med)

`git push origin <branch>` - pusher det du har committet i branchen din lokalt til remote slik at andre kan hente det ut ved pull.

`git pull origin <branch>` - henter alle endringer fra remote i . Kan også bruke:

`git pull` - henter alt fra remote inkludert nye brancher osv.

Branching

`git checkout -b <feature_x>` - lager ny branch utifra den branchen du er aktiv i med navn <feature_x> og går inn i den.

`git checkout <branch>` - bytter aktiv branch til eksisterende <branch>.

`git branch -d <freature_x>` - sletter branchen med navn <feature_x>.

`git push origin <branch>` - pusher branchen <branch> til remote slik at den er tilgjengelig for andre.

`git branch -a` - se alle brancher som ligger lokalt og på remote.

Workflow i praksis

Eksempel på workflow:

Steg 1: Si vi har en task med id `#6` og med beskrivelse: `Set up and handle touch input from user`

Da tar man først å oppretter en egen branch for dette ved å først sørge for at man er i dev branchen slik at man får riktig versjon og så sjekker at du er up-to-date her ved å pulle fra denne.

Steg 2: Da er man klar til å branche ut å begynne på oppgaven. Da lager du først en ny branch enten med navn som beskrive issue eller feature id.

`git checkout -b handletouch` (evt. `feature_6`)

Steg 3: Man begynner så å jobbe med featuren og er klar til å committe noe som bidrar til issue-en men som ikke closer den. Da legger man først alt i index, så committer man det med referanse melding og så pusher man det til remote.

`git add .` - adder alle endringer til index

```
git commit -m "Initialized touch input. #6"
```

```
git push origin handletouch
```

Steg 4: Man jobber så videre og ferdigstiller featuren slik at tasken kan closes.

Steg 5: Når alt er på plass i branchen handletouch kan man så opprette en pull request med dev for å merge endringene inn her.

PS! Husk å slette branchen når man er ferdig med en oppgave. Dette kan du gjøre ved kommandoen over for å slette lokalt og for å slette fra remote kan du gjøre det ved ´git push origin --delete´ i terminalen.

Tips og triks:

Hente fra dev

Steg 1: Commit det du har gjort:

```
git add .
```

```
git commit -m "melding. #issue"
```

Steg 2: Gå til dev branchen:

```
git checkout dev
```

Steg 3: Oppdater dev branchen:

```
git pull
```

Steg 4: Gå tilbake til din branch

```
git checkout your-branch
```

Steg 5: Rebase på dev (Gjør at branchen starter på dev head)

```
git rebase dev
```