**ABSTRACT**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Title: |  | | | Date: |  |
|  |  | | |  |  |
|  |  | | | | |
|  |  | | | | |
| Participants: |  | | | | |
|  |  | | | | |
|  |  | | | | |
|  |  | | | | |
| Supervisor(s): |  | | | | |
|  |  | | | | |
| Employer: |  | | | | |
|  |  | | | | |
| Keywords: |  | | | | |
| (3-5) |  | | | | |
| Number of pages/words: | | Number of appendix: | Availability: | | |
| Short description of the bachelor thesis: | | | | | |

# Preface

Thank you to ...

Intro about Intility (MAX 0.5 PAGE)

# Glossary

Data jobs

DBT docs

Business system

Reports/models

Exposures

Mart

Staging

Source

WAN

WCAG

Bifrost

Intility standard

SSMS 18 (SQL Server Management Studio 18)

[1 Preface 2](#_Toc133320187)

[2 Glossary 2](#_Toc133320188)

[3 Introduction 5](#_Toc133320189)

[3.1 Domain 6](#_Toc133320190)

[3.1.1 Data science 6](#_Toc133320191)

[3.1.2 The current solution – DBT and DBT docs 6](#_Toc133320192)

[3.1.3 Manifest.json 7](#_Toc133320193)

[3.2 Objective 7](#_Toc133320194)

[3.3 Group background 8](#_Toc133320195)

[3.3.1 Competence 8](#_Toc133320196)

[3.3.2 Choosing the task 8](#_Toc133320197)

[3.4 Constraints 9](#_Toc133320198)

[3.4.1 Time 9](#_Toc133320199)

[3.4.2 Access to sensitive information 9](#_Toc133320200)

[3.4.3 Practical 10](#_Toc133320201)

[3.4.4 Physical 10](#_Toc133320202)

[3.5 Delimitations 11](#_Toc133320203)

[3.6 Project goals 11](#_Toc133320204)

[3.6.1 Result goals 11](#_Toc133320205)

[3.6.2 Effect goals 11](#_Toc133320206)

[3.6.3 Learning goals 12](#_Toc133320207)

[3.7 Target audience 12](#_Toc133320208)

[3.7.1 Application 12](#_Toc133320209)

[3.7.2 Report 12](#_Toc133320210)

[4 Requirements 13](#_Toc133320211)

[4.1 UML Class diagram 13](#_Toc133320212)

[4.2 Functional requirements 14](#_Toc133320213)

[4.2.1 Use-case diagram 14](#_Toc133320214)

[4.2.2 Actors 15](#_Toc133320215)

[4.2.3 High-level use-cases 16](#_Toc133320216)

[4.2.4 Descriptive use-case 17](#_Toc133320217)

[4.3 Non-functional requirements 18](#_Toc133320218)

[4.3.1 Availability 19](#_Toc133320219)

[4.3.2 Reliability 19](#_Toc133320220)

[4.3.3 Maintainability 19](#_Toc133320221)

[4.3.4 Scalability 20](#_Toc133320222)

[4.3.5 Security 20](#_Toc133320223)

[5 Design 20](#_Toc133320224)

[5.1 GUI 20](#_Toc133320225)

[5.2 System design 22](#_Toc133320226)

[5.2.1 Backend system design 23](#_Toc133320227)

[5.2.2 Frontend system design 24](#_Toc133320228)

[6 Technologies 25](#_Toc133320229)

[6.1 Backend 26](#_Toc133320230)

[6.1.1 Alternatives to .NET: 27](#_Toc133320231)

[6.1.2 Manually testing the endpoints 28](#_Toc133320232)

[6.1.3 Backend compilators 28](#_Toc133320233)

[6.1.4 Backend testing tool 28](#_Toc133320234)

[6.1.5 Database tool 28](#_Toc133320235)

[6.2 Frontend 29](#_Toc133320236)

[6.2.1 Typescript vs Javascript 29](#_Toc133320237)

[6.2.2 React vs Vuejs 29](#_Toc133320238)

[6.2.3 Frontend testing tool 30](#_Toc133320239)

[6.2.4 Bifrost 30](#_Toc133320240)

[6.3 Communication and helper tools 30](#_Toc133320241)

[6.3.1 Azure (alternativet til å ikke ha det i cloud) 30](#_Toc133320242)

[6.3.2 Gitlab 31](#_Toc133320243)

[6.3.3 Communication and documentation 31](#_Toc133320244)

[7 Implementation 32](#_Toc133320245)

[7.1 Deployment 32](#_Toc133320246)

[7.1.1 Continuous Integration and Continuous Deployment 32](#_Toc133320247)

[7.1.2 Deploying to Azure 32](#_Toc133320248)

[7.2 API 32](#_Toc133320249)

[7.3 Database 32](#_Toc133320250)

[7.4 Lineage graph 33](#_Toc133320251)

[7.4.1 Lineage graph frontend 38](#_Toc133320252)

[7.4.2 Lineage graph backend 38](#_Toc133320253)

[7.5 Changelog 40](#_Toc133320254)

[7.5.1 Backend Changelog 40](#_Toc133320255)

[7.5.2 Frontend Changelog 40](#_Toc133320256)

[8 Testing 40](#_Toc133320257)

[8.1 Backend Testing 40](#_Toc133320258)

[8.2 Frontend Testing 40](#_Toc133320259)

[8.3 User testing 45](#_Toc133320260)

[9 Methodology 46](#_Toc133320261)

[9.1 Scrum 46](#_Toc133320262)

[9.2 Jira 46](#_Toc133320263)

[9.3 Issue types 46](#_Toc133320264)

[9.4 Meetings with Intility 47](#_Toc133320265)

[9.5 Meetings with bachelor guidance consultant 47](#_Toc133320266)

[9.6 Internal meetings 48](#_Toc133320267)

[10 Discussion 48](#_Toc133320268)

[10.1 Work environment 48](#_Toc133320269)

[10.2 What we learned working for Intility 49](#_Toc133320270)

[11 Conclusion 50](#_Toc133320271)

[11.1 Results 50](#_Toc133320272)

[11.2 Process 50](#_Toc133320273)

[11.3 Future development 51](#_Toc133320274)

# Introduction

In today's data-driven world, organizations rely heavily on data to make informed decisions. However, managing and documenting data can be a complex and time-consuming process. In response to this challenge, our team has developed a data documentation tool aimed at simplifying the documentation process for data jobs.

This tool is designed to help data professionals track and document data transformation processes, provide an overview of data dependencies, and facilitate collaboration and knowledge sharing among team members leading to increased efficiency and productivity. In this report, we describe the design and implementation of our data documentation tool and provide an evaluation of its effectiveness in a real-world setting.

## Domain

To ensure a shared understanding for this thesis, it would be helpful to clarify terminology and concepts, which we will do in the following subchapter.

### Data science

Data science is a field which involves using statistical and computational methods to extract insights from large and complex datasets. It combines computer science, mathematics and statistics, machine learning and advanced analytics to identify patterns and relationships within the data (Kelleher & Tierney, 2018). The goal of data science is to use data-driven insights to make better predictions and decisions.

Data scientists are responsible for a range of tasks, such as data cleaning, preprocessing, analyzing, and visualizing, as well as building predictive models using various machine learning algorithms (Gardner, 2021). To be successful, data scientists need to have good problem-solving, critical thinking, and communication skills to develop data-driven strategies and explain their findings to stakeholders.

The importance of data science is increasing in many industries, including healthcare, finance, and marketing, due to the exponential growth of data collection. This has led to a rapid expansion in the field of data science, which again calls for useful and user-friendly tools for documentation.

### The current solution – DBT and DBT docs

The employees in Intility’s data science department use DBT (Data Build Tool) extensively in their daily tasks. DBT is a tool that helps build, test and maintain data pipelines. It provides features for testing, automatic documentation generation, modularity, and version control, making it easy to collaborate and manage changes. (Barcheski & Collis, 2023)

In addition to building and maintaining data pipelines, DBT also provides a feature called DBT docs. This feature automatically generates a website containing documentation for your DBT project, including information on models, sources, and tests (See example in Figure X.X). With DBT docs, you can easily keep your project's documentation up to date, collaborate with team members, and provide a source of truth for your data pipeline. Another feature within the website is a lineage graph, visualizing the data pipeline inside of DBT (Figure X.X).  
By using DBT and DBT docs together, you can create a robust, maintainable, and well-documented data pipeline that is ready for analytics and reporting.

### Manifest.json

The ‘Manifest.json’ file can be exported from DBT docs and contains all the information about how the data is organized and transformed in the DBT project. This includes information about models, sources, and tests, as well as their relationships and dependencies. An example of such metadata can be the name, id, file path, type, description and dependencies of an exposure. This information can be used to get a better understanding of what data the exposure contains and how it is affected by other data.  
In our project we will use the information in the manifest file to build a custom documentation tool which displays the data pipeline, where the data comes from, how it's transformed, and what the final output looks like.

## Objective

The main objective of this project is to create an easy-to-use application for documenting data jobs. The application should function as a substitute for the existing documentation solution (DBT docs) and have extended functionality to fit the needs of Intility’s employees in the data science department. Our application should also include the functionality to display information about business systems, and the relationship between business systems and sources.

A key issue Intility has with the existing solution is that there are no opportunities to develop further functionality. Since the applications used today are not made by Intility, they do not have the option to change layout, change existing functionality or implement new features. Opening for such an opportunity would be very helpful to further increase automation in Intility’s data science department.

## Group background

### Competence

This report introduces a group of four bachelor students who are all enrolled in the ‘Programming’ study program at NTNU in Gjøvik. Throughout the program, the students have acquired skills and knowledge about a variety of programming subjects. We have experience working in a group from previous projects throughout their education, which was a major reason why we chose to come together to write our bachelor thesis.

In the figure below is a table containing the relevant courses from the program, with a short explanation of their relevant learning outcomes.

|  |  |
| --- | --- |
| Fundamental programming | Basic programming skills |
| User centered design | Sketching, prototyping, user testing |
| Software development | Scrum, project planning and execution |
| WWW-technologies | JavaScript, React |
| Advanced programming | Testing |
| Cloud technologies | APIs and cloud-based programming |
| Data modelling and database systems | Databases and SQL |

### Choosing the task

In the initial phase of the project, our main goal was to find a suitable task that would allow us to apply the knowledge and skills we have acquired throughout our studies. Among the suggested tasks from the university, we saw three tasks from Intility, all of which were interesting. To apply for these tasks, we sent an application letter that included our CVs and grades from all previous courses. After two interviews, we were offered the opportunity to work with Intility.

We accepted the offer, as we found the task to be remarkably interesting and helpful to the company. We especially liked that it was an application which provided real value to the data science department, would be frequently used, and that would be developed further after our time at Intility was over.  
The agreement included that the group would develop the data platform manager this report is based on. Intility would provide the necessary equipment, train tickets and commute housing every two to three weeks. We have described the work environment and collaboration with the company further in chapter X.X.

## Constraints

The constraints of a project can play a critical role in shaping the direction of the work and its outcome. In the case of our bachelor thesis, we faced a range of constraints that influenced the development of our data documentation tool. These constraints were both internal and external and affected various aspects of the project. In this chapter, we will explore the constraints we faced and how we managed them to achieve our project goals.

### Time

This assignment was large, and Intility did not expect us to finish the application with the given time. The application had limitless potential for further development, which meant that we had to set clear delimitations of what should be prioritized. The whole project, including writing the project report should take between 560 and 660 hours per student.

To manage this constraint, we decided not to create tests for every part of the code, but instead create good tests for the main functionality. This was to show that we have the knowledge and skills to create good tests and could have created tests for all functionality if time was no constraint.

### Access to sensitive information

Access to Intility’s internal information about customers and employees was another constraint. Firstly, we did not have access to a database to retrieve employee names to have options when selecting owner and contacts for new business systems. After dialogue with Intility, we produced a solution to store the previous used name which had to be added manually. This will be fixed when Intility take over the application development.

Secondly, we did not have access to the endpoint which let us access an updated manifest.json file. This introduced a problem in the early stage of the development due to information from the updated manifest file which we did not have access to. The final product also did not have automatically updated information due to this constraint. The functionality to update the database based on a new manifest file is implemented, but for now the user has to manually change the existing file.

### Technical expertise

Technical expertise can pose a significant constraint in the development of the data documentation tool. Although we have obtained a strong foundation in computer science throughout our studies, the development of a complex tool like this requires advanced technical knowledge and expertise. The tool requires the integration of various data sources, such as business system information, reports, and automatically generated documentation from DBT. This will again require knowledge about DBT, DBT docs and the entire data science field, which we have little to no experience with.

Furthermore, the tool should be coded using Intility’s preferred languages, namely TypeScript and C#, which are also relatively new to everyone in the group. To achieve the objectives of this project, we need to first acquire the necessary knowledge and skills within these technologies.

### Practical and physical

Effective communication within the team is essential for a successful software development process. The physical distance between the team members and Intility was another hurdle we had to manage. With the use of several digital communication tools, we were able to keep the close collaboration with the company during the entire development process. We as a group are well familiar with this method of communication due to the Covid-19 lockdown. More information about the digital communication tools we used can be found in chapter [‘7.3.4 Communication and Documentation’](#_Communication_and_documentation). It was important that we had descriptive issues in Jira, and that everyone on the team updated the issue board so there were no one working on the same issues.

Other than the physical distance, there have not been any other practical constraints related to our work. Everyone on the group got a laptop and the necessary equipment to work as efficient as possible both at the office and at home. Intility provided us with all the needed software, hardware, our very own office spaces, and the access to meeting rooms.

Intility also covered lunch at the office and any travel costs related to the commute. The employees at Intility have done everything they can to ensure that we have had good conditions to work on this project.

## Delimitations

As mentioned, the main objective of the project is to make a data job documentation tool, however there are delimitations regarding what should be implemented in the final product.

Only documenting functionality will be within the scope of this project. This means we will not implement functionality which has a direct impact on the data jobs, but rather documents information about each stage of the data pipeline. The application will not have functionality to trigger or re-run data jobs, however, we should leave this as an opportunity to expand for the company at a later stage.

The finished product will feature user authentication, however individual user roles like admins, readers and developers will not be part of this version of the application. This feature can be added at a later stage. We are not provided with the privileges necessary to obtain information about individual employees whenever they are logged in to our application.

## Project goals

In this project, our primary objective is to achieve a set of specific goals that will help us accomplish our overall task. Our goals are divided into three different categories: result goals, effect goals and learning goals. Result goals focus on the results we wish to achieve, while effect goals concentrate on the impact our project will have on the intended audience. Learning goals center on the new knowledge and skills we hope to acquire during the project. With a clear understanding of these goals, we can effectively plan, execute and evaluate our project to ensure its success.

### Result goals

The result goal of our project is to create an application capable of displaying all relevant information about a data job and its pipeline. This will be achieved by tying together automatically generated documentation from DBT with business system information and reports. In addition, we aim to include a feature displaying a visual representation of the relationships between various parts of a data job, while ensuring that the loading time of the visual representation is no more than 50% of DBT docs’ alternative. Lastly, we intend for the application to automatically update its saved information once every 24 hours.

### Effect goals

* Increased adoption and usage of the data documentation tools outside of the *Data Science* department, resulting in increased overall understanding and collaboration across teams.
* Increase accessibility of information and ease of information gathering in terms of more forgiving search parameters.
* Ensure that data in the system is accurate and reliable by tracking changes made.
* Reduce the loading time for large lineage graphs.
* Design which complies with WCAG and Intility’s brand.

### Learning goals

Another important aspect of our bachelor thesis is learning goals. The goals below are what we expect to learn through conducting and concluding this bachelor thesis. The group would like to acquire more experience and knowledge of the following:

* Agile development and Scrum in order to have an effectively workflow on the project.
* Git best practices, version control and CI/CD (continuous integration & continuous deployment).
* Azure and cloud platforms to host an application.
* Managing a larger project with deadlines for delivery.
* Become confident in .NET and Typescript and SQL for data management and documentation tasks.
* To learn more about how to collaborate with a company to develop a useful application that meets their needs.
* Making an application within an unfamiliar domain like Data Science.

## Target audience

### Application

The primary target audience for our application is individuals who work in the data science department of Intility. This includes data scientists, data analysts, and data engineers who require an intuitive and user-friendly tool to analyze large and complex datasets efficiently. However, the application may also be used by other employees at the company who work with data but do not have a specialized data science knowledge. These include departments such as finance or marketing, that can also benefit from using the application to analyze data relevant to their areas. The tool is designed to simplify the data analysis process enabling users to gain a better understanding of it. To summarize, the application is valuable for anyone working with data at Intility, regardless of their level of expertise.

### Report

The target audience for our report is primarily educators and evaluators. The report provides an in-depth analysis of the development process and implementation of our data platform manager. It covers the relevant information needed for someone reading to understand how the application is implemented, our development process and other relevant information. In addition to educators and evaluators, our report may also be of interest to students. The report can be valuable for students wanting to learn more about creating or learning more about data analysis tools. Overall, our report is a comprehensive and informative resource that offers insights into the development and implementation of our data platform manager, making it a valuable resource for a wide range of individuals.

# Requirements

The requirements chapter outlines the specific needs and expectations for the data documentation tool developed as part of our bachelor thesis project. By defining these requirements, we aim to ensure that the tool effectively meets the needs of its target users, mentioned in chapter X.X.

This chapter covers the functional and non-functional requirements of the tool, including its desired features, performance specifications, and usability guidelines. We have identified these requirements through extensive research, meeting with the target users, analysis of existing data management tools, and research around the industry best practices. The resulting set of requirements provides a clear roadmap for the development and evaluation of the tool, ensuring that it meets the needs and expectations of its users.

This chapter covers the functional and non-functional requirements of the tool, including its desired features, performance specifications, and usability guidelines. We have identified these requirements through extensive research, including interviews with potential users, analysis of existing data management tools, and consideration of industry best practices. The resulting set of requirements provides a clear roadmap for the development and evaluation of the tool, ensuring that it meets the needs and expectations of its users

## UML Class diagram

The class diagram shows the structure of the domain and the software requirements needed.[[1]](#footnote-2) It describes the systems entities and their relations which helped gain understanding of the problem area. It was also helpful when creating the logical model and data classes in the code used in the API.

Diagram

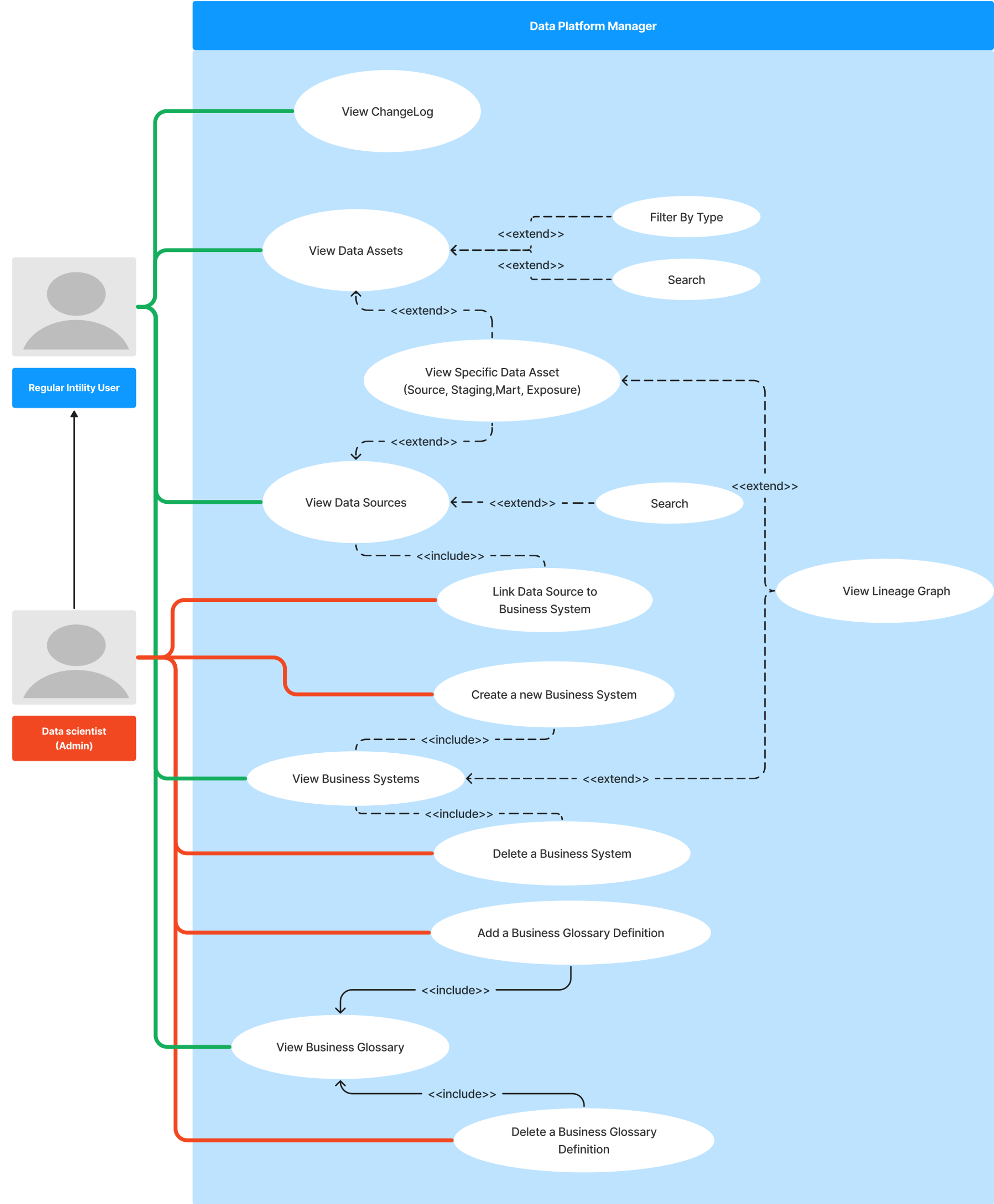
Description automatically generated

In the image above one can see that the manifest contains many “data models” which is a xxxx. This means a specific data model can be either be one the four classes which inherit it. This was not in the original domain model which only contained the subclasses. Figuring this out early meant that normalization was easier when creating the logical model.

## Functional requirements

### Use-case diagram

The use case diagram shows how the different actors can interact with our system and the functionality it provides. The diagram doesn’t give much detail but gives us a list of requirements that the system needs. This helps in creating the system and getting a common understanding of what the system needs, which was difficult at the beginning of the project.



### Actors

**Regular user:** A regular user will have the possibility to inspect all data assets and metadata. The lineage graph will provide an overview and relations between business systems and data. The user will not be able to make any changes that can impact other users or the system outside of their own view.

**Data Scientist:** A data scientist will have all the functionality of a regular user. As well as the possibility of adding or deleting business systems and linking them to data sources. The admin user will also be able to add or delete business glossary definitions.

### High-level use-cases

|  |
| --- |
| **Use case:** View Data Assets and search.  **Actor:** Regular User  **Objective:** Find all data assets relevant to a specific keyword.  **Description:** The user navigates to the data assets page showing a list of data assets. The user searches for the keyword in the search field. Capitalization should not impact the results which will be displayed in alphabetical order. However, it will filter based on ASCII symbols used such as underscore or forward slash etc. |
|  |
| **Use case:** View specific data source.  **Actor:** Regular User  **Objective:** Find a specific data source and view its information.  **Description:**  **(main)** The user navigates to the data source page.  **(alternative)** The user navigates to the data assets page and filters based on “*source”.*  **(continuation)** The user clicks on the source he wishes to see and enters the specific data source page which shows relevant information to a data source. |
|  |
| **Use case:** View changelog.  **Actor:** Regular user  **Objective:** View changelog for the last four days  **Description:** The user navigates to the application home page which displays recent changes to data formats such as deletions, additions, or changes to data assets. The user then selects a date four days prior, which filters to show the latest changes. |
|  |
| **Use case:** View the lineage graph for a business system.  **Actor:** Regular User  **Objective:** Navigate to a business system and display its lineage view.  **Description:** The user navigates to the business system page and clicks on the business system they wish to view. Further the user clicks an icon or button that specifies the lineage graph and opens the lineage graph page. The page shows the business system’s data sources and their lineage through data transformations into final data products. |
|  |
| **Use case:** Add business glossary.  **Actor:** Data Scientist (Admin)  **Objective:** Add a new business glossary  **Description:** The user finds the help section of the navigation bar and selects “business glossary”. The user searches the glossary and doesn’t find the definition they are interested in. The user clicks the add button or icon which allows the user to write a new word and description. If the word already exists, an error is issued, elsewise the word is added to the glossary. |

### Descriptive use-case

An example of a detailed use case to show ideal flow of execution.

|  |
| --- |
| **Use case:** Create a new business system and link data sources then view the updated lineage graph  **Actor:** Data Scientist (Admin)  **Objective:** Add a new business system and link data sources to the given business system  **Description:** The user navigates to the business system page and adds a business system. Then the user adds data sources which should be linked.  **Program flow:**   1. The user clicks the business system from the application menu. 2. In the business systems page the user clicks the *“add business system”* button or icon. 3. A form to create a new business system opens and the user fills fields such as name, system type, description, owner and contact person. 4. The user then clicks submit, prompting any fields which doesn’t pass validation for change. 5. The page navigates to the newly create business system and the user clicks and clicks *“add new sources”.* 6. A list of eligible sources is displayed and the user searches and checks off any source they wish to link. Then the user clicks add. 7. The selected sources are displayed under the business system. 8. The user clicks the lineage graph icon or button and is navigated to the lineage graph. 9. The lineage graph shows the lineage to the added sources and their transformation through the system. |

## Non-functional requirements

Non-functional requirements specify the characteristics and performance needs of a system.[[2]](#footnote-3) These requirements we have divided into five categories: availability, reliability, resilience, scalability, and security. By specifying these requirements, we know what we need to develop towards, but also what we don’t need to prioritize. If the system isn’t going to grow in users, functionality or data, there is no reason to prioritize testing and potentially refactoring for a much larger scale.

### Availability

After discussions with the product owner and the data science team it was clear that availability of the program would be especially important during business hours and preferably in the evenings. An agreement was made that nighttime would be the time to update our system with new metadata and changes made in the data catalogue. We therefore landed on an availability percentage of 99 which gives us roughly 15 minutes each day of downtime for updating and service downtime.

### Reliability

Since the system gives an overview of much of the company’s data, an important factor is the reliability to trust that the system’s information is correct and functional. Errors in data and dependencies between data can be an issue as it may give users wrong information resulting in bad business decisions and lost work hours. This is especially important as the system grows and expands functionality as described later in the further work chapter. This meant that extra focus had to be put on testing the functions and logic which read and update metadata to ensure reliability. Therefore, we aim to perform integration and unit testing on the most crucial functionality as well as user testing.

### Maintainability

“Maintainability refers to the ease with which you can repair, improve and understand software code”.[[3]](#footnote-4) The software built during this project is going to be iterated on and added features to. As stated earlier in xxx, the ambition of this project long term to also be used for scheduling and monitoring data jobs. Maintainability is not an easy thing to measure and can be quite subjective, but the objective is to write readable code without unnecessary complexity.

* Write about static code analyzer “SonarQube” and refer to this chapter.

### Scalability

* Write about how many users at launch
* How many users in the future

In terms of scalability a bigger concern than the number of users involved is the issue of the application to more data. Each week since the start of the project more metadata has been added and this can create problems which do not exist at this size. Therefore, we need to test for large data changes in our integration tests and monitor the speed of API closely. An API request with a response time of 2 seconds might be good enough at the current size, but not in the future.

### Security

This application’s purpose is as an internal business tool for employees and potentially customers. Users outside of these constraints should not be able to use the application. To solve this issue, the system will use user-authentication for both the front-end and the API.

The use-case diagram showed the two users: *regular user* and *admin user,* where the *regular user* only has READ privileges while the *admin user* also has POST, UPDATE and DELETE privileges. The authentication will need to differentiate between the two and limit the regular user’s options. You can read more about this in the API design and controller design chapter.

The report will cover specific security measures in the relevant chapters.

# Design

Bifrost

DBT docs vs DPM

How DPM design compares to other Intility applications

Refer to images within the implementation chapter to show the design

## GUI

Design:

The design of the Data Platform Manager utilizes Intility’s React library Bifrost, which offers a wide range of pre-built components and utilities that allow for the creation of responsive and scalable web applications. One key advantage of this library is its built-in coloring system, which adheres to the WCAG guidelines. This makes Bifrost the ideal choice for developers, as it drops the need to fully consider these guidelines when using other libraries, such as React Bootstrap. Moreover, since all applications and websites of Intility share the same styling foundation, it makes sense to use Bifrost in this case. By utilizing a familiar and widely used library, the development process of the Data Platform Manager becomes more streamlined and efficient. Furthermore, it ensures consistency with the existing applications and websites within the organization.

The frontend design for the data platform manager follows a standard layout that is optimized for a good user experience. The layout consists of a top-bar, navigation-bar on the left and a main body with content. They are set up as follows:

**Top bar:** The top bar includes an Intility logo, name of the application and on the right in the top bar there are two buttons. One button is for the user’s profile and the other to change between light and dark mode.

**Left side navigation-bar:** The left side navigation-bar has four main categories, overview, data catalog, data consumers and help. Overview links to the main page, the others have under-categories that link to the specific content.

**Main body:** The main body displays the content depending on which page that is navigated to.

For a good user experience the website is deigned to be as user friendly and intuitive as possible. First off it uses font size and icons following WCAG. Icons are made and used to be easily distinguishable. Another example is when the website is under a certain width the left side navigation bar will be hidden under a dropdown menu. This will make it more appealing for users with smaller screens so that they will be able to see the main content just as good as those with a larger monitor. Lastly, the sidebar contains dropdown menus for each of the main categories that can easily be opened. With this feature, users can find the information they need without having to spend too much time searching for it, making their browsing experience more enjoyable. Overall, the application uses a very minimalistic design that eliminates clutter and distraction making it very user friendly and simple to navigate.

As mentioned earlier the data platform manager adheres to several WCAG guidelines to ensure that it is accessible to a wide range of users, including those with disabilities, and offers a good user experience. WCAG 2.2 Guideline 1.4 is met using font sizes and icons that adhere to the guidelines, ensuring that users can easily read and comprehend the content of the website. WCAG 2.2 Guideline 1.3 is met by providing alternatives for non-text content using easily distinguishable icons, making the website accessible to users with visual impairments. The built-in coloring system of Bifrost adheres to WCAG 2.2 Guideline 1.4.3, ensuring that the color contrast between text and background meets the minimum requirements, making it easy to read for everyone. The website structure meets the WCAG 2.2 Guideline 1.3.1 by organizing the content into a top bar, left-side navigation bar, and a main body. This ensures that users can easily understand the structure of the website, find information they need, and navigate efficiently. WCAG 2.2 Guideline 1.4.11 is met using components with contrasting colors, ensuring that users with color blindness can easily distinguish between them. Finally, WCAG 2.2 Guideline 1.4.10 the left-side navigation bar is hidden into a dropdown menu on smaller screens making it so that users do not have to scroll or zoom in to view the content.

## System design

In the GitLab repository, there are two folders, one for frontend and the other for backend. All the group members had access to both folders to work on both backend and frontend. It was important to have a good structure of both the frontend and backend folder. The reason behind this is there is someone else at Intility that will take over the project after this bachelor's degree. Looking at the structure of other Intility applications was important in order to make this process as easy as possible for **the** next development team.

### Backend system design

Backend is structured in a way with a folder for each of the following: controllers version one, data, models, integrations tests and unit tests. All the folders are connected to each other where it is needed to use models or other related code from another folder.

The controllers folder consists of all the controllers which are handling the logic for the different endpoints. Some controllers have more than one request, this is because the endpoints are related to each other, or they have two different uses within the same endpoint. Figure xxxxxx below shows what a controller with different requests looks like in the Swagger tool mentioned in the technologies section. Each request in the controller is commented with response code which will give the user an indication if there is anything wrong. For example, if the response code is 500, then there is an internal server error.

The data folder is where all functions that are used in the controllers are located. There are functions to handle functionality related to updating and fetching information from the database. These functions handle the logic when a request is sent from frontend. There are also some functions which check the input from frontend before the database is populated with latest information. It is important to check for invalid input before the database is populated, even though there are checks for valid information frontend. This is to prevent SQL-injection if the user is not sending a request directly from frontend, but with the use of tool like Swagger or Postman.

Within the data folder, there is a manifestupdate folder

The models folder consists of classes of the structure. There are dto classes and regular classes. The dto classes are data transfer objects which are used to transfer data without updating the database. The regular model classes are used to update the database with the given information. There are also some minidto classes which are smaller sections of the dto classes used to reduce redundant information in the requests if the development team were to use the same data.

Integration test folder

Unit test folder is where all code related to testing of the different functions used in the different endpoints.

Figure(xxxxx)

Graphical user interface, text, application, email

Description automatically generated

### Frontend system design

Frontend is organized using a modular folder structure that includes several key folders that organize the different types of files used in the application. One of these directories is the “Assets” folder, which houses static files such as images and logos used throughout the application.

Another folder is the “Auth” folder, which contains all files related to authentication, such as login and authorization of users. It also includes a function for fetching data from the API with extra security authentication.

One of the most important folders in the frontend application is the “Components” folder. This folder stores reusable UI components that can be used across multiple pages and features. The use of reusable components helps to ensure consistency in the user interface and saves development time by avoiding the need to build the same UI elements from scratch for each new feature or page.

The components in the “Components” folder are organized according to how the sidebar of the website is listed. For example, the “Data\_Catalog” folder holds all components related to that category such as “Data\_Assets”. This organization is done to improve code organization and maintainability.

In addition to the component directory, there is a “Pages” folder that consists of the actual pages of the application, which may be composed of multiple components. This folder helps to keep the pages organized and easy to find.

Furthermore, there are three more folders: “Tests”, “Types” and “Utils”. The "Tests" folder contains files related to automated testing of the application. Tests are important for ensuring that the application works correctly and meets the desired requirements. The "Types" folder contains files that define and standardize the types and interfaces used throughout the application. One example is types for the API responses. The "Utils" folder contains utility related functions and configuration files.

Finally, there are files related to the setup and initialization of certain things, such as routing and test setup. These files are crucial for ensuring that the application is properly configured and set up before it is run.

Help

# Technologies

The development team chose to use the standard technologies they use in Intility. There are several reasons for this which will be enlightened in this chapter, but the main reason is that the group wanted to create a product which could be useful for a company like Intility. It was also important to use tools technologies that Intility use. This will make the further development of the product possible with just minor changes to our application.

## Backend

At the start of this project, the development team chose to use .NET for the backend development which is an object-oriented programming language designed by Microsoft. .NET is a language which is great to use with Web Api support and perfect for this assignment.

There are no problems with connecting the application to frontend and other external systems such as Azure. In one of our previous courses called “Integrasjonsprosjekt”, we struggled a little bit on connecting the frontend to backend and to make the website safe enough to have https. This was not a problem we encountered with this project due to better choice of backend technologies.

Another reason to use .NET is how easily it can be integrated with Swagger which we have used to manually test the endpoints. The figure xxxxx in chapter 5.2.1 ([Backend system design](#_Backend_system_design)) shows an example of how a controller for the endpoints look like. To use swagger, we had to authenticate with our Intility. The backend swagger was running at all time with the code pushed to Gitlab, bit we could also run the backend locally and check if the code worked as intended with localhost swagger. This was to ensure that the code pushed to Gitlab could run without any problems.

(https://ukad-group.com/blog/aspnet-core-8-pros-and-cons/)

The current application takes some time to load because it has a lot of information, therefore it is important to choose a programming language which runs fast and efficiently. As mentioned in chapter 3.7.1 ([Result goals](#_Result_goals)) and in chapter 3.7.2 ([Effect goals](#_Effect_goals)), our goals was to decrease the loading time for both the dbt docs and for the lineage graph. A possible solution to this was a fast backend which could cache large amounts of data ensuring that the load time will be faster after the first time opening the website.

(<https://hackr.io/blog/c-sharp-vs-java>)

### Alternatives to .NET:

Python is an alternative to .NET the development team could have used. There are some teams at Intility that are using python, but compared to .NET, python is not that good at handling high-performance computing, and it is not the best language for processing substantial amounts of data. (<https://invedus.com/blog/python-vs-net-choosing-the-right-language-for-your-project/>).

Golang is a familiar language which is like .NET, but it is a simpler language and has fewer features than .NET. Features like LINQ and pattern matching are not supported in Golang. (https://cshark.com/blog/c-vs-go-which-is-better-for-your-next-application/) LINQ is a fantastic addition to a language which supplies a consistent query experience to relational databases. (<https://www.tutorialsteacher.com/linq/what-is-linq>).

Java is one of the most widely used programming languages for development. There are a lot of reasons why java is well suited for the software industry. For instance, Java can run on any operating system with its compilers and Java Runtime Environment. That does not apply to .NET which only focuses on Windows users. This did not matter for the development team, because the application will be developed on computers with Windows and when the application is pushed to production, it can also be used by users without Windows.

A reason to decide on .NET instead of Java is because the development team are more familiar with Visual Studio as an IDE compared to IntelliJ which is the most used IDE for Java. This is not a huge reason, because most IDE’s are very similar, but we found it more suitable to go with the familiar IDE. This topic will be discussed in chapter 6.1.3 ([Backend compilators](https://studntnu-my.sharepoint.com/personal/martiran_ntnu_no/Documents/Backend%20compilators)). (<https://www.upgrad.com/blog/java-vs-net-difference-between-java-and-net/>)

All the three languages are exceptional for web and cross-platform development, but we did not find any other reason to why we should deviate from the standard Intility backend language.

### Manually testing the endpoints

In order to test and develop the backend endpoints before trying them frontend, the development team have used swagger. The advantage of Swagger is that you can quickly test the API as it creates documentation, example requests and data from the code. In the development phase this is a useful tool to keep the cost of change low. The development team could also have used Postman for the manual testing of the endpoints, but Swagger is well integrated with the verification tokens (bearer tokens) which are needed to get the correct response from the database.

### Backend compilators

The compilers used for this project are both Visual Studio Code and Visual Studio 2022. The reason behind this is different preferences within the group. There are several purposes with using Visual Studio 2022 and Visual Studio Code instead of, for example other compilers like IntelliJ. The main reason the development team are more familiar with Visual Studio.

It was also the preferred from Intility that we used either of the two compilers in order to make it as easy as possible to integrate the application with Intility’s existing applications. This is because the other developers at Intility use these compilers and all the systems they used are integrated with the two compilers. Another reason was that they could offer more help if that was needed.

### Backend testing tool

* xunit

### Database tool

The database tool used for this assignment is Microsoft SQL Server Management Studio 18, which is a graphical integrated development environment used to administrate and manage a Microsoft SQL Server. By using this tool, our backend developers could create tables within the code and use the query editor in SSMS 18 to write and execute SQL queries to see that the information in the database was correct. The was also used in the early stage of the development process, to find sources to for example marts in order to check if our endpoint for finding all information about an unique mart worked.

## Frontend

The frontend tools used in this project is Typescript with React. The main reason behind this combination, is that Intility preferred that they were the tools for the frontend. In the two following chapters, it will discussed why the technologies did not deviate from the standard Intility tools and compare the choice of tools with alternatives.

### Typescript vs Javascript

In the development of our project, we utilized TypeScript, a superset of JavaScript, which is strongly typed and adds additional features to the language. Although we could have used JavaScript for the same purpose, the decision to use TypeScript was recommended by Intility, due to its numerous benefits. TypeScript improves code readability and maintainability, enhanced error-checking and debugging capabilities, and increased code scalability. With TypeScript, the code is easier to read and understand as the type definitions make it clear what data types are being used. Additionally, it catches errors before the code is run, making debugging easier and reducing the time spent fixing issues. TypeScript also scales better for larger projects, as it allows for better code organization and modularization. Overall, the use of TypeScript was a wise decision as it enabled us to develop a more robust and maintainable codebase.

[TypeScript: JavaScript With Syntax For Types. (typescriptlang.org)](https://www.typescriptlang.org/)

[Javascript VS. Typescript: Which Is Better for Your Project In 2023? - JayDevs](https://jaydevs.com/javascript-vs-typescript/#:~:text=TypeScript%20is%20particularly%20useful%20for,choice%20due%20to%20its%20simplicity.)

### React vs Vuejs

On a previous project Vue.js was chosen as the framework. While Vue.js is also a great choice for building web applications, we ultimately decided to use React for this project. Both Vue.js and React are amazing frameworks for building web applications and have a lot of similarities. However, there are some differences between the two that has led us to use it for this project. One major factor was React’s performance, which is optimized for building large-scale and complex applications. Additionally, React’s popularity and large developer community make it easier to find resources and support. Since its used by developers in Intility they provided us with a course in React and have also helped when needed. Another reason why we chose React is its flexibility and modularity, which allows us to easily integrate it with other libraries and tools as needed. Overall, the decision to use React for this project was based on a careful evaluation of its strengths and our projects requirements.

[Comparison with Other Frameworks — Vue.js (vuejs.org)](https://v2.vuejs.org/v2/guide/comparison.html?redirect=true)

[Vue vs. React — Comparison. What is the best choice for 2022? - Fulcrum](https://fulcrum.rocks/blog/vue-vs-react-comparison#:~:text=One%20of%20the%20main%20reasons,is%20only%20a%20few%20milliseconds.)

### Frontend testing tool

Jest

### Bifrost

Bifrost is an Intility design system which is a useful tool for developers to quickly develop new projects. All Initility’s site use this design system in order to make the interface of the websites look the same. The Bifrost library consists of different models and examples for front-end development. The front-end team can look at the Bifrost website and use components from the website in order to develop their own website. The models from Bifrost are not tailored to each project, so in some cases modifications to the existing models are needed.

## Communication and helper tools

### Azure (alternativet til å ikke ha det i cloud)

### Gitlab

Gitlab has been the version control tool for this project. There are also other similar tools such as GitHub and Bitbucket, but the group have not been using them throughout the bachelor’s degree. At the beginning of the project, the group set up a pipeline for both the frontend and backend folder and preset some rules on Gitlab such that it is not possible to commit to the main branch.

At the start of the project, the group had a Gitlab workshop with an employee in Intility, which thought us to set up some necessary configurations like CI/CD for the pipelines. It was a useful workshop where the group learned more about best practices with naming conventions, usage of different branches and merging.

See meeting minutes xxxx in appendix (if there are any)

### Communication and documentation

**Teams**

Teams have been the main platform for meetings and communication with both Intility and the bachelor guidance counselor. There are several other options like Google meet, but Teams is what the development team are the most familiar with and it provides a lot of different functionality for scheduling meetings and asking questions to other employees in Intility. Slack is also a tool used to contact other employees about questions related to the project.

**Discord**

Discord has been the main platform for communication within the group. In Discord, it is possible to create a Discord server, with different text channels for various categories. This way, there have been stored links, meeting minutes and other important material related to the bachelor project.

**Jira**

The development team had a biweekly sprint period where they had several issues in the sprint. At the beginning of the process, there was an attempt to use issue tracking in Gitlab which is the standard issue tracking tool within Intility. However, the group found it easier to keep track of their issues using Jira over GitLab. The backlog is another feature provided in Jira. The backlog is a list of issues where the developer could select additional issues if there was any time left. Jira is an exceptional tool for this purpose.

# Implementation

Write about the whole process, how things are made and why

Introduction with the order of operations

Write about all the different pages implemented with screenshots

## Deployment

### Continuous Integration and Continuous Deployment

### Deploying to Azure

## API

## Database

The database was set up by Intility, but managed and populated with functions created by us. All the information is either added manually by the user of the application or added automatically from the manifest.json file. Example of information added manually are the business systems or the business glossaries.

* Purpose
* Design
* Data Entry and Managements
* Queries and Reports
* Data Security and Backup
  + Checking for correct input before database is populated

When sending requests from frontend to the database, there are functions checking the input fields. If the input is wrong, the request would fail and give an error message to the user. This will make sure that the user will try to fill in the input again, but with only the allowed characters.

There are still possible so send request through tools like Swagger or Postman to avoid this input validation. To avoid this problem, we had to create input validation function in backend as well to check for invalid input. This is done by precompiling the regex, then checking the input fields with the allowed characters in the regex. The reason behind precompiling the regex was to follow best practice when working with regex in order to have as fast as possible execution of the program. A cached regex can be reused, and this will be faster than compiling the same expression repeatedly.

* User Interface
* Future Enhancements

## Lineage graph

Displaying the lineage graph frontend was a major part of the assignment, and finishing it was a primary objective from the very beginning. It was important for Intility that the lineage was easy to use and user-friendly. A request from them, was that our lineage graph looked like the already existing lineage graph on the DBT docs site. They also wanted some extra features like Intility standardization (wcag) and earlier stages of the data jobs.

First, there were some sketching in Figma in order to have some visual on how there would be possible to create the function. It was a simple sketch, but it gave us a great understanding of

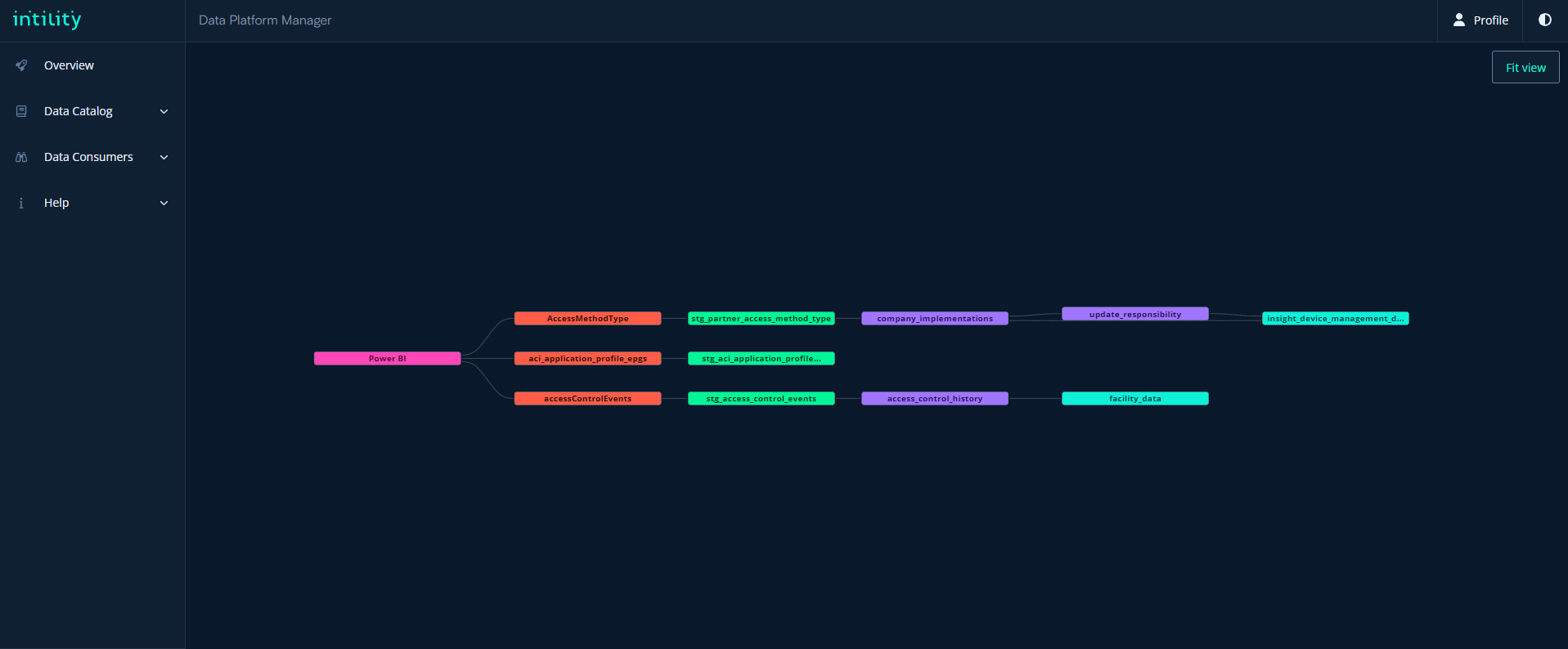
Et bilde som inneholder stridsvogn

Automatisk generert beskrivelse

After the sketching, we had to find a way to create it on a website. The data science department tipped us that another department had create a similar graph as we had sketched out. The picture below was the inspiration we had when we started creating our own.

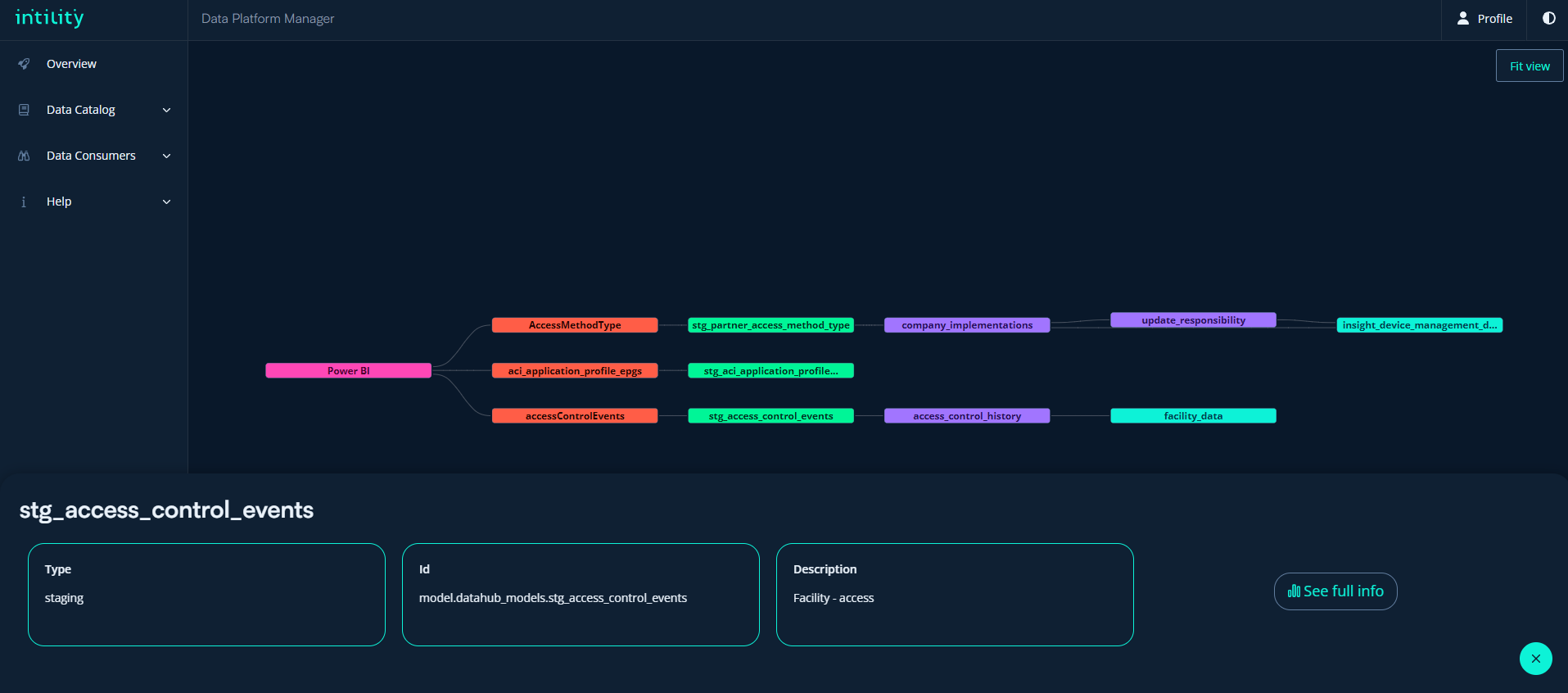
Et bilde som inneholder Nettsted

Automatisk generert beskrivelse



This is an example of the results of our lineage graph. The graph starts at a business system and finds all sources, stagings, marts and exposures. There are some similarities to the Network department’s lineage graph, but when the two pictures above are compared, it is easy to see that our graph is used for another purpose with less details and another way to display the nodes.

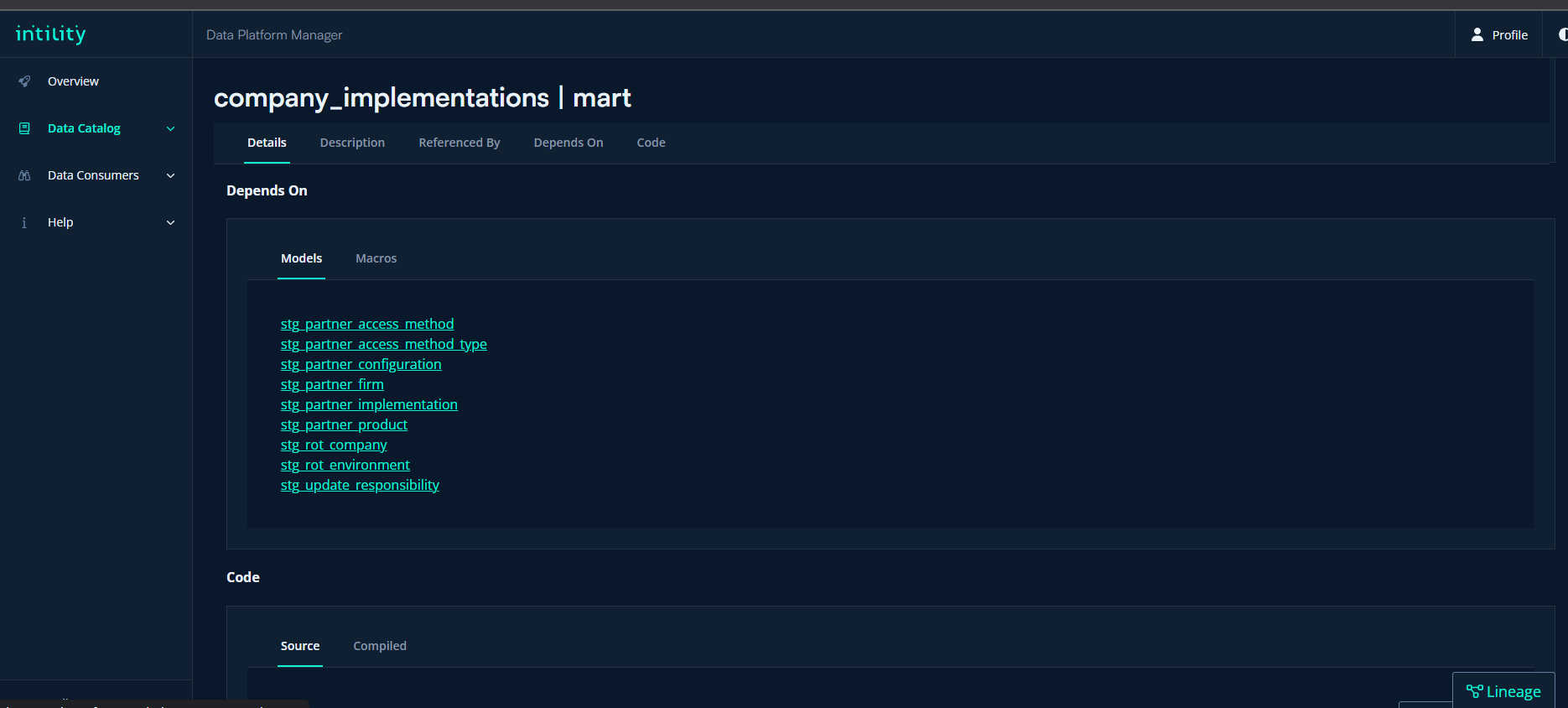
We wanted to create a graph which was easy to use and had clickable interactive functionality to display information. There are several interactive functionalities on the lineage graph. All the nodes are clickable and will display a drawer with basic information about the node, and it is a button within the drawer to display the full information. The “Fit view” button located in the top right corner of the screen will fit the screen back to default if the user has been zooming. As mentioned, zooming the graph is also an option in order to make larger graphs more readable. It is also possible to move all the nodes around without losing the relations. The line between the nodes will not disappear if a node is moved.



When the ‘See full info’ button is pressed, the website will be redirected to a page with more detailed information about the type. The two pictures below are the information that will be displayed.

Et bilde som inneholder tekst

Automatisk generert beskrivelse



There are different WCAG colors for the different types which are shown on the figure xxx above. By doing this, the user could find out the relation types in the graph. Look at the table below for the color codes.

|  |  |
| --- | --- |
| Pink | Business system |
| Red | Source |
| Green | Staging |
| Purple | Mart |
| Turquoise | Exposure |

We are extremely proud of how the lineage graph ended up looking. As mentioned, we have created a good looking, easily readable and interactive graph within the Intility standard. The graph is faster and more effective compared to the existing graph as DBT docs, which was one of our goals in the chapter 3.7.2 ([Effect goals](#_Effect_goals)).

* Innovative way to solve this task, good looking frontend and connecting business systems to a source

In the next two chapters, we will break down how the frontend and backend are programmed in order to display the graph shown on the previous figures in this chapter.

### Lineage graph frontend

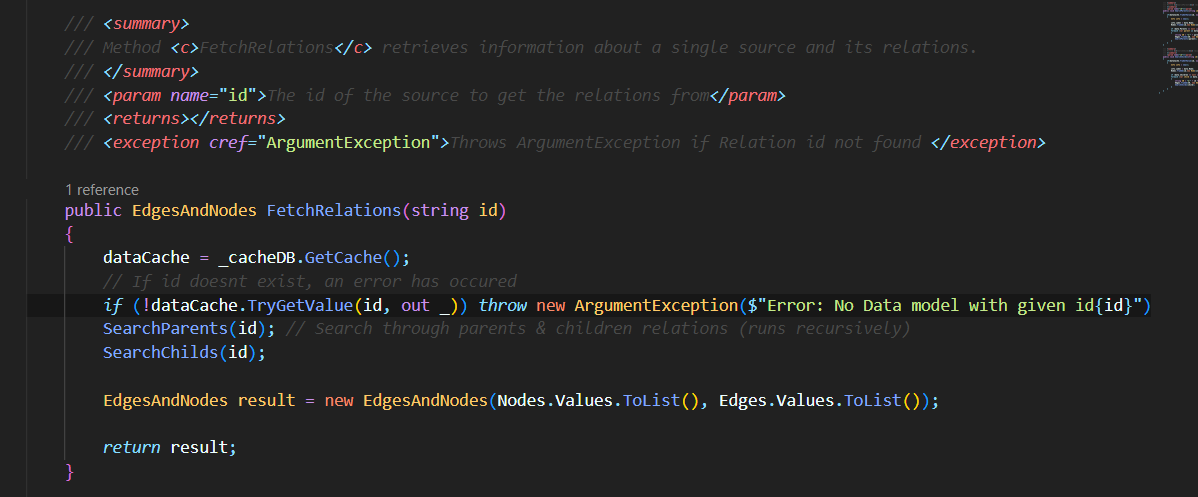
With little to no experience with visualizing graphs, this quickly proved to be a challenge. Luckily, there is a React component called React Flow, which was made for this purpose. After thorough research, the front-end team displayed a simple graph to the screen. Navigation throughout the graph and moving the nodes are part of the react flow component out-of-the-box.

The next major challenge was finding a way to arrange the nodes correctly in a tree shape for easy visualization and understanding. To accomplish this, the group utilized a package called “The Eclipse Layout Kernel” (ELK). This package introduces a collection of layout algorithms to simplify the process of getting the initial diagram layout right.

After the diagram showed the correct nodes and edges in the right position, the next issue was making the nodes interactive, to display a drawer of basic information whenever a node is clicked.

### Lineage graph backend

There were a lot of research in order to be able to display the whole lineage graph with different nodes and information for each node. The solution became to create recursive functions. This knowledge was taught to us in the Algorithm methods earlier this bachelor’s degree.



Et bilde som inneholder tekst

Automatisk generert beskrivelse

Et bilde som inneholder tekst

Automatisk generert beskrivelse

The ‘SearchParents’ and ‘SearchChilds’ functions, are looping through to find all relations. If there are any relations, an id consisting of an “e-“ and the combination of the id and the parent or child. The “e” stands for edge. When the edges are added, the function will run recursive again to find all the relations.

The two functions mentioned above is together in the ‘FetchRelations’ function which is used by the controller whenever a call to the endpoint is made.

These functions took a lot of time to create, and we were very proud of how it turned out. There are complex functions which needs to go through all nodes and find their child nodes and parent nodes. A node can have many nested children, so the function needed to continue to check the children’s nodes for even more children.

## Changelog

### Backend Changelog

### Frontend Changelog

# Testing

## Backend Testing

## Frontend Testing

For frontend testing we performed unit testing. We used the framework Vitest for testing which is excellent for unit testing, and a fast framework that includes additional features from the jest framework. For frontend we have a total of 6 tests. These are tests for testing the linear search function that we made. Including a function used inside the search function that removes special characters. We tested the search function as it is the function which is most likely to have flaws and issues. Doing tests on it we figured out that some of the functionality didn’t work as intended and we had to change the function accordingly.

**RemoveSpecialCharacters** function:

Et bilde som inneholder tekst, skjermbilde, Font

Automatisk generert beskrivelse

This is a unit test for a function removing special characters from a string. It sets the string to be “hello-world!”, and then the expected result should be “helloworld” as it removes the dash (-) and the exclamation mark (!) from the string. The test succeeds if the removeSpecialCharacters can remove special characters and equals the expected result. The removeSpecialCharacters function can be viewed here: link

**simulateSearch** function:

Et bilde som inneholder tekst, skjermbilde, programvare

Automatisk generert beskrivelse

For the simulateSearch function we did 5 tests. The tests are based on filtering, sorting and loading. This function simulates a linear search and returns all the data assets matching the search. The function is explained in this chapter: link

**Mock data:**

Et bilde som inneholder tekst, skjermbilde, programvare, display

Automatisk generert beskrivelse

**Test 1 (filters and sorts the search results correctly when value is not empty and does not contain an underscore)**

Et bilde som inneholder tekst, elektronikk, datamaskin, display

Automatisk generert beskrivelse

The first unit test filters and sorts the results based on the string “test”. This should then print out all results including that word. It then simulates the search, starts a timer and sets loading to false and then checking that the search function is returning the expected results.

**Test 2 (filters and sorts the search results correctly when value is not empty and contains an underscore)**

Et bilde som inneholder tekst, skjermbilde, programvare

Automatisk generert beskrivelse

Todo: explanation

**Test 3 (sets search results to an empty array when value is empty)**

Et bilde som inneholder tekst, skjermbilde, programvare

Automatisk generert beskrivelse

Todo: explanation

**Test 4 (sets loading state to true when simulateSearch is called)**

Et bilde som inneholder tekst, skjermbilde, Font, programvare

Automatisk generert beskrivelse

Todo: explanation

**Test 5 (sets loading state to false after a 1000ms delay)**

Et bilde som inneholder tekst, skjermbilde, programvare

Automatisk generert beskrivelse

Todo: explanation

## User testing

# Methodology

## Scrum

The Scrum process consisted of sprint review, sprint retrospective and sprint planning. The three parts were discussed and planned in the scrum meetings. At first, there was a sprint period of one week, but it increased to two weeks due to not having enough time to solve the more complex issues. When there were shorter sprint periods, the group did not create enough issues due to underestimating the workflow. With longer sprints, there were trouble-free to have complex issues which could take a lot more time than expected. The meetings with both Intility and the bachelor guidance consultant were primarily every week. The group tried also to have daily stand-up meetings when there was enough time and need to conduct a meeting.

## Jira

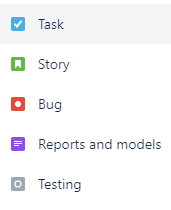
As mentioned earlier in this report, the development team used Jira to keep track of their issues. The group together had the responsibility to create enough issues for each member. All the issues had an issue type, and each issue was moved to the in-progress section when a group member started with the issue. The group member assigned the issue to themself, in order to not have more group members working on the same issue. It happened that the development team did not manage to finish all the issues. The reasons why that happened were discussed in the sprint retrospective part of the scrum meetings and the issues got moved to the next sprint.

Prioritizing the issues and talk about the order of the issues was important. This was not a problem, due to the good communication within the group. There were not any critical parts of this process, because the issues were often added to the sprint board after the sprint started. The reason the group decided to have this practice with the issues is because their features always appeared as the development progressed.

## Issue types

As mentioned above, there were different issue types. The most used was ‘task,’ which was used for all the regular features on the application. Whenever issues related to bugs were created, they were set to bug. The ‘report and models’ type was related to the project plan, the bachelor report and other documentation. The last type that frequently was used was ‘testing,’ related to the testing both frontend, backend, and the user tests. There is also a predefined story issue type in Jira, but there are no issues that had this type.

Having different issue types made it straightforward to see what the issue was about.



## Meetings with Intility

The development team had weekly syncs with Intility. There were mainly two representatives from Intility. One of them handled all the technical questions related to the assignment, and the other was responsible for the functionality. It also happened that five representatives from the team attended the user test(s). It was set aside one hour for every meeting, and it was scheduled on Friday, but it also happened that the meetings got scheduled for Tuesday instead. The motive behind these meetings, was to ask questions related to the task both if the group was struggling with some code problems or if there were any questions about how the application should look. It was important for the group to make the application as Intility requested, because then they could keep working on the application afterwards.

## Meetings with bachelor guidance consultant

A Status meeting with Johanna Johansen was planned every Wednesday at 13.30. There was one in the group which handled sending a status report to the bachelor guidance consultant. The focus of the meetings was feedback on the report and other relevant work at that given sprint period. Getting feedback on the report along the way, improved the workflow. After the feedback, the group had a meeting and talked about what they needed to refine. The group also got a clear indication on what they needed to include in the report in order to get their wanted grade.

See appendix xxx for examples of meeting minutes.

## Internal meetings

The group had several meetings when it was required. There were often extra group meetings right after the meetings with Intility and the bachelor guidance counselor. This was to discuss the notes from the meetings and talk about any further improvements. The biweekly scheduled meeting was where the group had the scrum meetings and discussed the previous sprint and the upcoming sprint. The available time and main goal of the upcoming sprint were major talking points, and the group created the necessary issues to reach their goal.

See appendix xxx for examples of the sprint planning meetings.

It was held daily standup meetings, where the group talked short about what they planned to work on that day. By doing so, all the group members were up to up to date on everyone's progress and everyone knew what they should work on. It happened that there were not enough issues for everyone. In that case, the group extended the standup meetings in order to talk about further issues.

# Discussion

## Work environment

At the start of the project the group was introduced to the office and the different departments within Intility. All group members got their own designated work spot with necessary equipment like monitors, headset, mouse, and keyboard. They also received laptops to be used for everything related to the project and the company.

Intility’s offices are located in Oslo, and therefore the group members were unable to travel there daily from Gjøvik and Lillehammer. The commute takes about two hours each way by public transport. As a solution, the company sometimes offered the students to stay in their commute apartments to attend the office physically several days in a row. The expenses related to the trips were also covered by Intility.

An effect of the physical presence in the office was a better relationship and collaboration with the client, and especially the data science department. The work spots were located right by data science, which led to seamless communication and assistance with the task.

## What we learned working for Intility

Working for Intility have been a period with a lot of learning and we would not be without. It was a great experience to have a try on how it works out in a real company before we all eventually will start working like this every day.

As mentioned earlier in this report, there were set up different workshops at the start of the bachelor’s degree. This was to make sure that we were familiar with all the tools that we were familiar with all the tools needed for this project. It was also important that we had an understand of how Intility as a company did certain parts of their workday. In the following chapter, there will be discussed about what we have learned working for Intility.

(Et kapittel per punkt!)

* Usage of git with branches and best practice commits
* Continuous integration and continuous delivery

At the begging of our period at Intility, there were several workshops that they wanted us to attend to. The workshops were about how the different departments worked and what they could help us with, the tools Intility uses for their application and best practices at Intility. There were a lot of new information to learn, and we were told that there was not expected that it was possible for us to learn everything with our short time at Intility. These workshops made the development process a lot easier, because we were more familiar with the tools and the different departments.

Working in team is something that the group was familiar with before the bachelor’s project, but working in a larger team with more resources and the option to get help across different teams was not familiar. When there were any questions related to the development process, we were given the contact information to the employees at Intility with the most knowledge regarding our problem. This made it possible for us to develop as much as we did within the timeframe, because we got the needed help if we were stuck and could not find a solution.

Creating a product on demand was something new for us. Throughout the previous courses, the larger assignments have not been strictly defined and a with a free approach on how to solve the task. For this project, we been having weekly meetings with Intility in order to meet their needs with the project that we develop for them. It is worth mentioning that the assignment has been very open, and we have decided a lot based on the time limits and what we were comfortable to create.

Working for meeting the need of Intility have been a process that we learned a lot from. In chapter 8.3 ([User testing](#_User_testing)), you can read more about how the user tests was done at some Intility employees. User testing a product is something that the previous courses have barely thought us. It was very helpful to get some additional reading materials from our bachelor guidance counselor. After the user tests was done, there were a lot to redo in order to meet the need of Intility. This have been discussed more in chapter 8.3 ([User testing](#_User_testing)).

# Conclusion

## Results

Write about if we met the goals

## Process

The development process was very educational, and the group learned a lot about how to work project-based in the industry. However, if the group were to carry out the project again, they would make some changes.

Primarily they would choose to spend more time on the planning phase, before starting to implement functionality into the application. To utilize scrum efficiently, the group would start by composing a backlog of all known issues for the project's duration. This would help contain a better overall flow throughout the development.   
Another aspect of the task that could have been improved was communication with the client and gaining a better understanding between the group and the client about specifications. The requirements given at the start of the process did not fully correlate with what the client really wanted, which led to some changes being introduced midway through the development. If the group set expectations for a more concise set of requirements, this could have been avoided.

## Future development

As mentioned in the introduction, part of why Intility wanted their own data platform manager application was to have the opportunity to customize and improve it further. During several meetings with the data science department, plenty of unique and useful features have been discussed. Not all these features were implemented during this bachelor project, mainly because of limited time.

The main feature discussed, and possibly the most useful, is to make the data platform manager more interactive by adding an option to refresh a report. This would be very useful for anyone using the data within Intility, and potentially also for Intility’s customers. Upon a button click, the data platform manager backend would trigger a refresh on every model a given report is dependent on. Starting from the business system, and going through every source, staging, mart and exposure the selected report depends on. This would provide updated data in seconds, without the hassle of manually finding every model needing an update.

Another useful feature would be an admin page. Giving only the employees within the data science department access to certain functionality like creating and editing business systems, changing the glossary or linking a source to a business system. These features are currently available for all users of the data platform manager, as the group does not have the privileges to integrate the roles.

References

Firesmith, D. (2020, January) *Engineering System Resilience Requirements*. Carnegie Mellon University. https://insights.sei.cmu.edu/

Barcheski, J & Collis, S (2023, March) *Dbt (Data Build Tool) Overview: What is dbt and What Can It Do for My Data Pipeline?*  
 *https://www.analytics8.com/blog/dbt-overview-what-is-dbt-and-what-can-it-do-for-my-data-pipeline/*

Gardner, M. (2021). Data Science for Beginners: An Introduction to Big Data, Analytics and Machine Learning. The Data Science Handbook.

Kelleher, J. D., & Tierney, B. (2018). Data Science: An Introduction. CRC Press.

* SQL Server Management Studio (SSMS) - Overview: <https://docs.microsoft.com/en-us/sql/ssms/sql-server-management-studio-ssms?view=sql-server-ver15>
* SQL Server Management Studio (SSMS) - Tutorial: <https://docs.microsoft.com/en-us/sql/ssms/tutorials/connect-query-sql-server?view=sql-server-ver15>

1. https://coursepress.lnu.se/courses/object-oriented-analysis-and-design/02-theory/domain\_class\_diagram [↑](#footnote-ref-2)
2. https://www.geeksforgeeks.org/non-functional-requirements-in-software-engineering/ [↑](#footnote-ref-3)
3. https://www.sealights.io/software-quality/software-maintainability-what-it-means-to-build-maintainable-software/ [↑](#footnote-ref-4)