TDT4290 - Customer Driven Project

FishFarmData - Sintef Ocean
Group 6

Sebastian Aas
Bjørn Andre Aaslund
Emanuele Caprioli
Turid Cecilie Dahl
Sander Kjetilson Kilen
Adrian Langseth
Ivar Nordvik Myrstad
Maria Storødegård

Department of Computer Science
Faculty of Information Technology and Electrical Engineering
Norwegian University of Science and Technology

Date July 14, 2021

Contents

1. Executive Summary

This report describes the work of Group 6 in the course TDT4290 - Customer Driven Project at the Norwegian University of Science and Technology (NTNU). The report documents the group's approach to the project planning, research, requirements specification, and development of a data management system for SINTEF Ocean.

The customer - SINTEF Ocean - operates SINTEF ACE, a laboratory project to support research and development for fish farming in realistic conditions. SINTEF ACE includes a full-scale fish farm facility with sensors that capture environmental measurements and other relevant measurements in a fish farm. The task is to develop a data management system that supports sensor data flow and storage, sensor metadata, and presentation of data to users of the system. The desired final solution of the entire system is comprehensive and not feasible to complete during the span of this course. The main focus of the customer is on solid research and architecture, to demonstrate the feasibility of the system rather than developing a production-ready system.

The group began the project by conducting a pre-study of the problem space in which different possible solutions were researched and compared to the requirements specification. Next, the group designed the architecture and the database, while simultaneously using the requirements specification to design a visual prototype. For the development phase, a modified version of Scrum was used, adding elements from Extreme Programming and Kanban.

The final delivered product consists of a full stack web application that offers an easy way to visualize sensor data, upload data for different sensors, manage and view sensor metadata, and subscribe to the sensor data API. The implementation of the product is documented thoroughly in this report.

2. Introduction

This chapter introduces the customer of the project and their work, followed by an overall description of the customer's assignment. The last section presents the stakeholders of the project, as well as their interest in the system's requirements.

2.1 Customer

The customer is SINTEF Ocean, with Finn Olav Bjørnson as the customer representative. SINTEF Ocean, a subsidiary of the independent research organization SINTEF, conducts research and innovation related to ocean space for national and international industries. SINTEF Ocean has been granted an Aquaculture Research license to support research and development for fish farming in realistic conditions. The project is called SINTEF ACE and is a full-scale laboratory facility designed to develop and test new aquaculture technologies. Facility users include researchers and others conducting practical experiments [1].

2.2 Task

The task is to explore the possibility of creating a data management system for SINTEF ACE through open source components. Data management includes everything from the data flow into the system, data storage along with associated metadata, and provide the data to different users and user groups.

The data flow includes SINTEF's sensors, connections to external systems, as well as quality assurance of the data. The data storage consists of secure storage in the short and long term, as well as linking the data to metadata to simplify querying. The availability of the data involves creating and maintaining several different user groups with different access levels. The data which is to be provided includes everything from simple sensor status, graphs for specific time series, as well as an API for direct retrieval of data.

The group's task is to create a proof-of-concept of the final solution. In this report, the prototype will be referred to as *the product* and the final data management system as *the final solution*. The purpose of the project, creating the product, is to determine the feasibility of the final solution. Therefore, the customer's main focus in this project is on solid research and architecture.

The codebase can be found at: `https://github.com/Sanderkk/CustomerDrivenProject`

2.3 Stakeholders

The stakeholders of the system and their interest in the system's architecture is presented below.

2.3.1 Customer

The customer - SINTEF Ocean - is the main stakeholder of this project. SINTEF Ocean has a large amount of sensor data that they want to organize in a time series database. The customer is mainly concerned about getting an architecture with high scalability and modifiability for a functioning value chain of sensor data.

2.3.2 End Users

The end users - engineers, researchers, and customers - are interested in the usability and utility of the system. The system needs to be easy to use for users without high technical competence, but also provide enough features and customizability to be useful for more advanced users.

2.3.3 Group Members

The group is concerned with building a solid product for the customer using the experience and theoretical basis which the group members have acquired in current and previous courses. Additionally, through the course of the project, the group wants to acquire new knowledge and experience to use in future studies and work.

2.3.4 Course Staff

Due to the nature of the project, the course staff is a stakeholder of the product. The course staff will be interested in the quality of the product and the theory behind it, in addition to the documentation on the group's process, requirements, challenges, and choices.

3. Planning

Project planning is crucial to deliver a product of high quality within the deadline, and helps the team to get an overview of the entire project. The process ensures that the team can track its progress, and make adjustments to complete all aspects of the project. This section describes the project plan, how the team is structured, and how the project will be organized.

3.1 Project Schedule

The Gantt chart in Figure 3.1 describes the overview of the project and the dependencies between the different tasks. The group has divided the project into four main task groups. These are shown in the Gantt chart as solid black ranges. The task groups are further split into nine tasks, covering the main periods of the project. Supplementary, three milestones are included to signify the main achievements the group is aiming for, accompanied by the aimed achievement date.

3.2 Team Organization

The group discussed which organizational roles would be necessary to cover the entirety of the project responsibilities at the first group meeting. By assigning one organizational role to each group member, the different parts of the project will be overseen more thoroughly, and therefore lead to a higher quality product. An overview of the roles is shown in Table 3.1. In addition to the organizational roles, each group member is a developer. The developer roles are front end, back end, and full stack developer.

Table 3.1: Organizational roles, responsibilities and assignment

Role	**Responsibilities**	**Assignee**
Team Leader	Attend meetings with the reference group. Book meeting rooms and ensure that every team member has assigned tasks.	Sander
Deputy Leader	In the case of an absent team leader,the deputy leader inherits the leader's responsibilities.	Ivar
Scrum Master	Organize and lead Scrum meetings, and help the group with external resources. Responsible for updating the sprint backlog, and following up on actions from retrospectives to ensure their implementation.	Turid
Customer Contact	Plan meetings with the customer. Convey information between the group and the customer.	Adrian
Report Manager	Create and update an overview document containing all tasks relating to the report. Delegate tasks when necessary.	Maria
Secretary	Create agendas and reports for each meeting. Responsible for the weekly reports and the customer reports.	Bjørn
Quality Manager	Establish code conventions and best practices to maintain code quality. Ensure the use of established conventions and code reviews.	Sebastian
Test Master	Set up a testing framework and Continuous Integration (CI). Ensure that the test coverage goals are achieved.	Emanuele

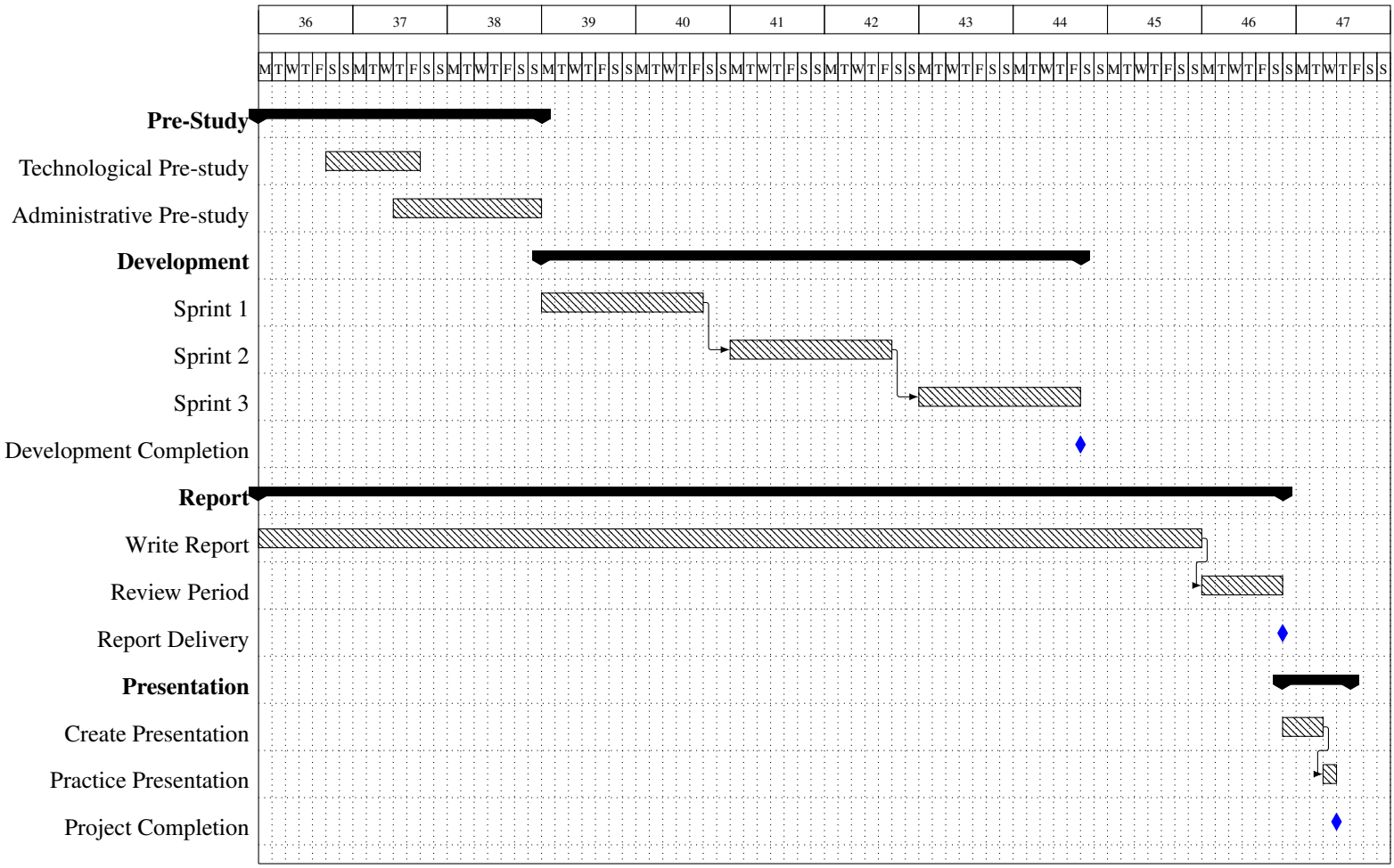


Figure 3.1: Gantt Chart for visualizing project scheduling and dependencies over the weeks 36 through 47. Milestones are blue diamonds, task groups in solid black ranges, and tasks as hollow black bars.

3.3 Meeting Plans

Regular meetings are important to share and discuss new information, make decisions, and to strengthen the group's morale. The group scheduled regular team meetings, customer meetings, supervisor meetings, and team leader meetings to keep the different stakeholders up to date with the progress. Examples of notes from these meetings are shown in appendix E.

3.3.1 Team meetings

The group agreed to conduct meetings every Monday from 08:15 to 14:00, and every Thursday from 16:15 to 18:00. The main focus of these meetings was to discuss topics concerning the whole group, and delegating tasks. The coordination gained from the meetings was necessary to obtain a common understanding of the requirements from the customer, and to avoid duplicated work. Despite spending many hours in meetings, the group believes the meetings were necessary to enforce a steady workflow. In addition to meetings, the group conducted a Daily Scrum meeting every Wednesday, which will be described thoroughly in section 5.2.1 about Scrum.

3.3.2 Customer meetings

The group held meetings with the Customer in accordance with Scrum standards. Customer meetings were conducted for every sprint demo, although, the group had more frequent communication with the customer by email. The communication with the customer was valuable as it guided the group towards developing the customer's wanted solution. This resulted in less ambiguity and uncertainty about the customer's needs for the product.

3.3.3 Supervisor meetings

The group met with the supervisor every Tuesday from 12:15 to 13:00. The scope of these meetings was mainly the project report and the group's progress. The group described what they had completed the previous week, and which problems they had encountered. The supervisor followed up on the progress and made sure that the group was on the right track.

3.3.4 Team leader meetings

The course staff facilitated meetings with the team leaders to give an update on their respective progress. At team leader meetings, there was also an opportunity to discuss the challenges that the group had encountered. By listening to the different solutions, the group leader gained inspiration on how to solve different challenges, and conveyed it to the group.

3.4 Version Control Procedures and Tools

For agile development, it is important to use tools that support the development process and improve the efficiency of the team. In this section such tools will be described, and there will be given a brief overview of why they were chosen.

3.4.1 GitHub

GitHub is a service that provides hosting for version control and software development using Git. GitHub has features including pull requests, commit history, and branching structure, which contributes to the efficiency of the group. GitHub

was chosen due to the group members' previous experience with the service.

3.4.2 Microsoft Teams

Microsoft Teams is a software solution for team collaboration, including chat, video call, and calendar functionality. The group has used Microsoft Teams as the main solution for communication. Microsoft Teams was preferred above other solutions, such as Slack, due to the integration of video conferencing and that it was integrated in the Office 365 package at NTNU.

3.4.3 Jira

Jira is an agile development software platform, which the group used for issue tracking in the project. The reason the group decided to use Jira instead of other solutions such as Trello, was due to the automatic generation of burndown charts and its integration with GitHub. This platform helped the group keep track of the remaining issues, and to see where each group member was assigned.

3.4.4 Google Drive

Google Drive is a file storage provider, which the group used to store all the files relating to the project. This included documentation of meetings, agendas, different sketches and figures, reports, and effort registration. Keeping all of the files in one place helped the group to stay organized. As many of the different file storage providers have the same functionality, Google Drive was chosen due to the group's familiarity with the product.

3.4.5 Overleaf

Overleaf is an online editor for LaTeX, which supports real-time collaboration between several members. Overleaf is one of the leading LaTeX editors and was a natural choice for the writing of the report.

3.5 Quality Assurance (QA)

ISO 9000:2015[26] defines quality assurance to be a "part of quality management focused on providing confidence that quality requirements will be fulfilled". Quality assurance is an important task in software development to ensure that the product has high quality. Bugs are a well-known phenomenon for developers, but good quality assurance procedures can catch these mistakes before the product is released to the customers.

3.5.1 Time of response

To ensure effective communication between group members, it was created rules for the response time. The agreed-upon rules were written in the group contract. These rules were to check Microsoft Teams at least once a day and reply within 24 hours.

There was also written a customer contract with the customer representative, including rules regarding the time of response. The first rule was to answer emails within one working day. If the group had any questions for the customer, they must have sent it at least one working day before the meeting.

3.5.2 Git Conventions

To improve the procedures regarding software development everyone must follow the same Git conventions and branching structure. This is both important to ensure consistent behavior between members and to make it easier to prevent and fix mistakes if they appear. The Git conventions are explained in appendix B.

3.5.3 Code Review

To improve the overall quality of the code base, it was decided early on to have a rigorous code review policy. The group decided to have a policy for code reviews for both the development branch and the main branch. The goal was to be as comprehensive as possible, not only going through the logic of the code but also looking for ways to condense or make it easier to understand. An example of a code review is shown in appendix D.1.

An important aspect of code reviews is to share knowledge and to facilitate discussion. Is this the best solution or can it be solved in a better way? Code reviews help to give insight into the development of the rest of the group. Since not everyone had experience with the languages or frameworks used, it was also a good way to share experience and knowledge. To give and receive constructive criticism was a great way of learning.

3.5.4 Templates and Standards

The group used two programming languages in this project: C# and JavaScript. It was decided to follow popular code conventions for these languages. For C# it was decided to use Microsoft's C# code conventions [4]. For JavaScript, it was decided to use the Prettier extension [5] in Visual Studio Code.

To improve the consistency of the paperwork in this project, the group created templates for recurrent documents. This includes documentation of the different stakeholder meetings. An example of the group meeting template can be seen in appendix D.3.

Another measure implemented to improve the quality assurance procedure is the pull request template shown in appendix D.2. A pull request template is a template that needs to be filled out when creating a new pull request. This is to make the developers explain clearly what has been done, and if all the quality assurance steps have been fulfilled. This makes it easier to observe what has been done and to create consistency. The pull requests also create a paper-trail. If a mistake happens in the future, a paper-trail can make it easier to locate the error and fix it.

3.5.5 Testing

Testing is an essential part of QA, for more information about testing see chapter 10

3.6 Risk Management

An important part of the planning is to identify which risks that could occur during the project. This makes the group aware of different events that could harm the project. It can be challenging to handle risks during a project if they happen unexpectedly. Therefore one should come up with strategies and actions to handle the risks effectively if they occur. The group wanted every member to take part in this process to get an accurate overview of the project's potential risks. The reason for this is that every group member has different experiences with aspects that can go wrong during a project.

The group started the risk management by defining different risk categories. The conclusion was that the biggest risks were most likely to be connected to personal issues, estimating, technology, corona, and the customer. For each type of risk, the group came up with different risk factors, consequences, and strategies. Only the most important risk factors were kept, and they can be viewed in Figure 3.3. A scale from 1 to 5 was used for the probability and consequence to easier separate the risks in the risk matrix in appendix Figure D.4. A probability of 5 reflects that the risk is very likely to occur, while a probability of 1 reflects that the risk probably not will occur. The consequences describe how damaging the risk is for the project, where 5 reflects that it would be hard to complete the project and 1 is a minor damage. Each group member estimated the risks individually, and thereafter the group performed a median. The risk matrix illustrates that the risks related to the customer were most important since they are located in the upper right corner of the matrix. Hence, the main focus of the group was the strategies reducing the risks connected to this risk type.

3.7 Effort Registration

The time spent on the course was tracked in an accessible spreadsheet, shown in appendix subsection D.1.1. Each group member logged their work daily, within one of the predetermined categories. These categories were lecture, research, development, documentation, meeting, administration, and miscellaneous. This system gave the group an overview of the distribution of time between the different categories and the workload of each group member. The group used this information to better estimate how much time could be spent on development, and to improve delegation of tasks between the group members to ensure an equal workload. Another important aspect of the effort registration was to make sure that each group member worked the amount they had agreed upon in the group contract. Figure 3.2 shows the amount of hours each group member used on the course each week, together with the group members' weekly work hour aim of 24 hours. Other graphs generated from the spreadsheet data is located in the appendix subsection D.1.1.

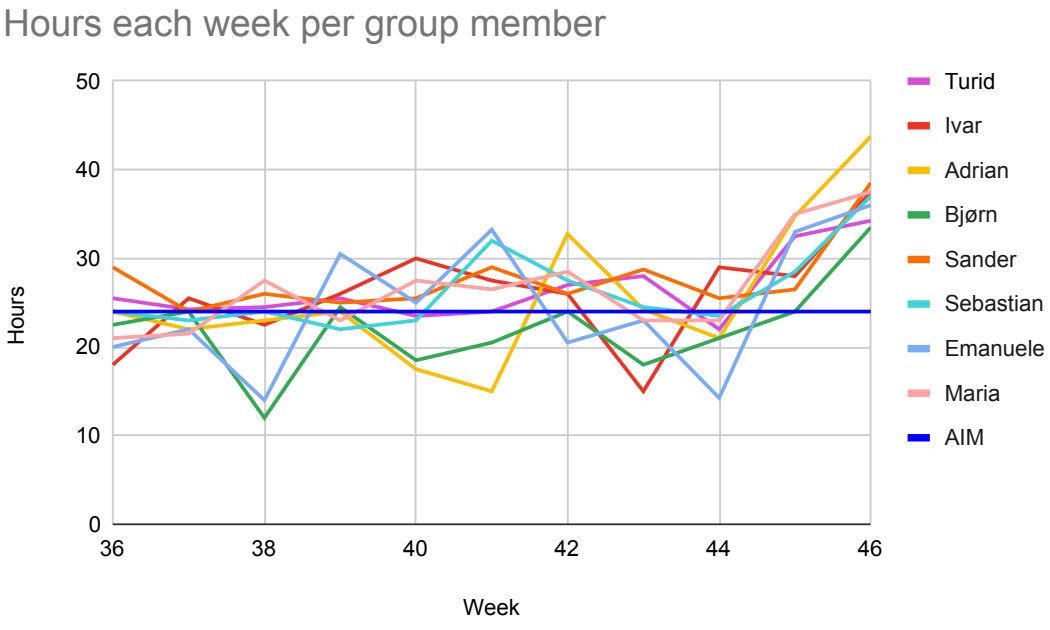


Figure 3.2: Line graph showing hours used on the course each week per group member.

Risk management

Type	Nr	Activity	Risk Factor	Consequences	Probability	Strategy and actions	Deadline	Responsible
Risk type		Which of the activities of the project are affected	Catching the name of the risk factors	Start with 1, 2, 3, 4 or 5 before describing the consequences	1, 2, 3, 4, or 5	Select strategy: Avoid, Transfer, Reduce, or Accept. Then on the next lines describe the measures	Set a clear dead line	Give one person the responsibility
Personal	1	All	Student dropout	4: The remaining students need to pick up the slack.	1	Accept: Divide the group member´s tasks between the remaining students	Continues	Adrian
	2	All	Group member is late for meeting	1: less time for meeting, and bad mood inside the group	4	Reduce: Have regular meetings every week, and use the same locations.	Continues	Bjørn
	3	All	Group member does not complete his/her tasks on time	3: The group is left with less time for future planed tasks	4	Reduce: Set clear deadlines, and follow up on other group member´s work	Continues	Emanuele
	4	All	The group has difficulty finding times to gather	4: Not possible to work as much together as intended.	3	Reduce: Prioritize the course over other things	Continues	Ivar
Estimating	5	Code	Software development takes longer than expected	5: can cause the project to not be delivered on time	4	Reduce: Improve the estimation process	End of coding	Maria
	6	Code	Repair of faults and / or defects takes longer than expected	3: can cause delays in the project	4	Reduce: Improve the estimation process	End of coding	Sander
	7	Report	Report takes longer time to write than expected	2: A lot of work to do in the last weeks, and maybe not able to deliver a complete report.	2	Avoid: Start early to write the report, and distribute the work load between the weeks.	Submission of report	Sebastian
	8	Meeting	Not enought time to go through all the planned things during the customer meeting	1: Schedule an adittinal meeting, or complete the meeting over mail	3	Avoid: Keep an eye on the time during the meeting, and set up a schedule before the meeting	Until last customer meeting	Adrian
Tech	9	Code	Errors in the thirdparty software used	5: The consequences depend on the error in the thirdparty software	1	Transfer: Find a new thirdparty software or take contact with people involved.	Continues	Bjørn
	10	Code	Limitations in thirdparty software	5: Maybe need to find another software, or come up with a workaround	2	Reduce: Do good research before using the thirdparty software to ensure that it has all the neccesarry functionality	Start of sprint 1	Emanuele
	11	Code	Requirements are not doable	4: Convince the customer to change the requirements	1	Avoid: Think about how the requirements can be solved before agreeing on them	Start of sprint 1	Ivar
Corona	12	All	The team is in Corona lockdown	1: One team mebmer have to work from home	3	Accept: Solve this issue by using Microsoft Teams	Continues	Sander
	13	All	Closed Campus	1: Have to work from home	4	Accept: Solve this issue by using Microsoft Teams	Continues	Sebastian
Customer	14	Code/planning	Misscommunication between customer and project group	5: Need to do a lot of changes, or start over to meet the customer´s requirements	3	Reduce: Make summaries from meeting available for the customer. Try to be as clear as possible when we communicate, and ask questions when we are not completely sure what the customer says	Continues	Adrian
	15	Code/planning	Customer is not able to meet the contract	4: Do not recieve all the neccesarry information to create a product which the customer is satisfied with	3	Accept: We can not force the customer to meet the contract in this project	Continues	Bjørn
	16	Code	Change in requirements	4: Need to do some changes in the code	5	Reduce: Include the customer in the sprint planning	Sprint planning	Emanuele
	17	All	Customer does not care about the project	2: Hard to work with the customer	1	Reduce: Be motivated in the meetings, and show enthusiasm	Continues	Ivar

Figure 3.3: Identified risks and how to handle them.

4. Pre-study of the Problem Space vs. Solution Space

This section describes the preliminary studies that led to the choice of technical solutions. The following sections describe the problem at hand and discuss the possible solutions that meet the customer's wanted situation.

4.1 Situation and Solutions of Today

SINTEF ACE performs measurements with several different sensor types. Each of the sensor types produces data with different configurations, which therefore need to be handled differently. The current solution consists of different systems for each of these sensor types, and a lot of the work is done manually. Due to the necessity of manual labor, some of the sensors have delays up to a month from when the sensors have performed measurements until the data can be used by the researchers at SINTEF Ocean. The solution for the sensor type "Seawatch Buoy" is illustrated in Figure 4.1. The Seawatch Buoy sends its measurements to the Fugro server, which runs automatic scripts, converting the data into CSV format, and uploads it to an FTP server. The data server queries the FTP server each hour, to check for new data. The data server thereafter runs a MySQL script, which writes these values into the MySQL database. To prevent any raw data from being uploaded to the web server, the data server preprocesses the data into graph images before uploading it. This is the only sensor type for which the data is visualized in graphs on the web.

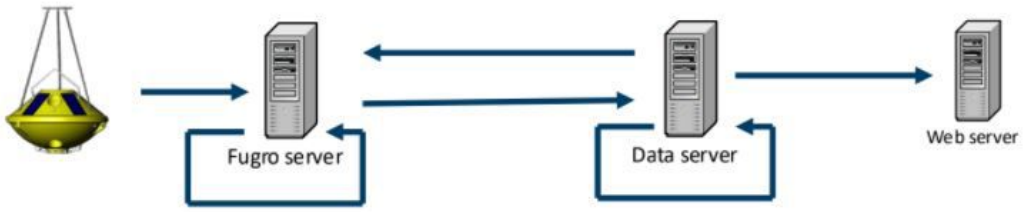


Figure 4.1: Today's solution: environment data from the Seawatch buoy

Currently, the data retrieved from the project SINTEF ACE is saved both in relational databases and large file structures on the servers, making it challenging to find and compare old data. The structure of the database has become a legacy system and all the original developers have retired. In the database, all measurement data is saved in the Observation-Measurement table, which makes this table long and unstructured. Due to this, it is tedious for an end user to find the desired data. The rest of the database's structure is complex and outdated, and only its original developers fully know how to operate it.

4.2 The Wanted Situation

SINTEF is looking for a solution where all their time series data and its associated metadata is stored in the same database, to make it easy to find and compare data from several years ago. The customer wants to transition from MySQL to a time series database and is particularly interested in using an open source database. The database should be able to store acquired data for 10 years, without downsampling or altering the data in any way. It is desired to create one system that can handle all the sensor types, to avoid different solutions for each of these. To automate parts of the current manual labor, it is desired to create an automatic notification system that external systems can subscribe to. In such a solution, the external system can subscribe to a given sensor, and be notified whenever new data for that sensor is uploaded to the database.

In an ideal solution, the users can lookup data within the last 10 years via a simple user interface without needing prior knowledge of the database or file structure. They should also be able to view metadata and upload new data connected to existing sensors or create a new sensor. Access control for each user group is considered to be very important, as the time series data is classified. The customer wanted the data to be visualized in graphs as shown in Figure D.8. The final solution should support multiple sensors in one graph and different graph types.

SINTEF is planning to use a provider to develop the final solution. The customer's goal is for the group to create a prototype of this solution as a proof-of-concept. This project is therefore important to discover which areas that need extra attention, and how such a system can be structured architecturally. When developing the final solution, the customer wants to use the group's experiences and architectural documentation to be better prepared.

4.3 Main Business Requirements

The group formalized the main business requirements based on the stakeholders' interests in the product. The main business requirements describe data flow, data storage, and availability. These requirements are the basis for the functional requirements of the product described in Table 6.1.

Table 4.1: Formalized business requirements.

ID	**Description**	**Priority (1-10)**
MBR1	The system needs to offer a simple way to visualize the desired data.	10
MBR2	Time series data and its associated metadata should be stored in the same database.	8
MBR3	The users need to be authorized to access sensor data	6
MBR4	It should be possible to connect metadata to sensors and later modify the metadata.	5
MBR5	Users should be notified in the event of a sensor failure or abnormal time series	5

4.4 Evaluation Criteria

The evaluation criteria for which the group based the technical solutions on were derived from the customer's requirements. The most important criteria that impacted the choice of technical solutions are shown in Table 4.2.

Table 4.2: Evaluation criteria

Name	Description
Complex user access management	Users should be divided into three user groups: researchers, engineers, and customers. Each of the user groups should have different access rights such as which time series they are allowed to view and if they can modify data. It should be possible for a user to be part of both the engineer and the researcher group. There should be different access rights within the researcher group.
Visualize data	The ability to find desired data through a simple user interface without requiring knowledge of the database structure.
Complex graphs	The ability to create complex graphs such as 3D graphs and heat maps. The graph library of choice must also support a large amount of data with minimal lag.
Introduce new sensors	The ability to introduce new sensors that are not on any of the formats previously known to the system.
Open source components	The system should be developed with only open source components.
Long term storage and data persistence	The database should have long term storage of the data, preferably 10 years of unaltered storage.
Search metadata	The ability to find time series through metadata, e.g. location or installation date.

4.5 Description and Investigation of Alternative Solutions

As described in the previous sections, the customer has specific requirements for the final solution. This section will therefore discuss the possible solutions regarding the visualization of data and databases that can lead to the customer's wanted solution.

4.5.1 Visualization of Data

To meet the evaluation criteria of visualizing data, the time series data must be visualized in graphs. The group has three main choices when it comes to visualizing the data: use an already existing platform, create a front end application, or a combination.

Grafana

Early in the project cycle, the customer representative mentioned Grafana [32] as a possible platform for visualizing data. Therefore, the group started to research Grafana and which features it could offer. Grafana is an open source and interactive web application that allows its users to create graphs from time series data through a simple user interface. Grafana is also a popular component used in combination with time series databases such as InfluxDB, Prometheus, and Graphite [17], which are some of the databases discussed in subsection 4.5.2. In Grafana, it is possible to split the users into three user groups with different access rights, which are admin, editor, and viewer. Therefore, Grafana offers several features that are desired by the customer; it is an open source component compatible with several of the databases considered, provides basic user management, and an intuitive interface that does not require that the user has any prior knowledge of the database structure. Additionally, using Grafana could save the group a lot of work, by not having to develop a front end application.

Using Grafana would also have drawbacks, particularly regarding the evaluation criteria of complex user access management. When using Grafana, any user within an organization can query the organization's data source. [31]. This entails that any user can find any data in the database, which contradicts the criteria for complex user access management. Upgrading to Grafana Enterprise could solve most of the issues related to complex user access management, but it would violate the evaluation criteria of only using open source components. It is also not possible to grant customized access rights within the groups, without making another team for that user. Therefore, the user access management would require a lot of work from an admin user. Another drawback of Grafana is that it does not provide functionality for visualizing metadata. To fulfill the evalution critera of searching metadata, the group would have to develop an application for connecting the sensor data to its metadata. In addition, Grafana does not meet the evaluation critera of complex graphs.

React Application with Embedded Grafana Frames

Another possibility is to create a React application with embedded Grafana frames. By doing this, the user access management is customizable and the functionality from Grafana can still be used to create graphs. After further research, the group noticed that Grafana does not interact easily with React, and every graph would have to be embedded individually. Another drawback of this solution is that it introduces an additional login step.

React Graph Libraries

The other alternative next to Grafana was to use a React library to create graphs. Three different libraries were considered: Recharts [21], React-timeseries-charts [13], and Highcharts [23]. Highcharts is the most popular library with around half a million downloads each week, and Recharts is right behind. React-timeseries-charts is less popular, with around 4000 weekly [41].

Recharts is a simple component-built open source library for React. It is a lightweight library that is easy to master, and according to [11] it is the number one React graph library for 2020. Since Recharts is a popular library, it has extensive documentation and is regularly maintained. The downside to Recharts is that it only provides basic graph types and does not facilitate time series graphs. Recharts does therefore not meet the criteria regarding complex graphs.

React-timeseries-charts is an open source React library created specifically for time series. Therefore, it has extensive functionality that would suit the requirements. The library has support for selecting a time range and visualizing multiple sensors from the same time range. A drawback of this library is the lack of built-in functionality regarding visual layout and graph interactions as opposed to Recharts and Highcharts. It would therefore require more work to create the interactive layout. As this library is not nearly as popular as Recharts and Highcharts, there is a higher risk that the library will become deprecated and the customer would have to restructure the application to use a different library. This library is not documented as well as Recharts and Highcharts, which is coherent with it having fewer responsible developers.

Highcarts is an interactive graph library trusted by 80 out of the world's 100 largest companies [24] and is used by SINTEF Ocean in today's solution, described in section 4.1. With such high popularity, the customer can trust that the library will not become deprecated, which makes it a safe choice for the long run. Highcharts provides different interactive graph types, such as 3D graphs and heatmaps, with a simple user interface. Therefore, the library meets the criteria regarding complex graphs and visualization of data. Another upside of its popularity is frequent updates, extensive documentation, and compatibility with most modern browsers. Highcarts is also easy to learn due to all of its

embedded functionality, which makes it a good choice for the project with little time to master additional technologies. Finally, Highcharts has support for time series and possibilities for processing a lot of data in one graph, which is important to reduce lag. One downside of Highcharts is that it is not open source, although it is free for non-commercial use.

4.5.2 Databases

To meet the evaluation criteria applying to the database, the group has considered and compared four different time series databases.

The database should be set up on a server of the group's choosing. This is a demand which falls under the evaluation criteria for open source, as a database being open source overlaps with it being available to set up on own servers. The evaluation criteria regarding the introduction of new sensors necessitate a push-based system due to the possibility of sensors with irregular time series. A push-based system is a system in which the application must push the data to the database any time new data is to be added to the database. A pull-based system is a system in which the data must be published at an endpoint from which the database pulls data at a set frequency. The pull-based system is not fit for the project as it does not allow for irregular, event-driven data collection, such as sensor detection of salmon louse. A pull-based system may still be viable if it integrates a push-based gateway. The evaluation criteria of long term storage and data persistence disqualifies any database which automatically does any downsampling without the possibility of disabling this feature. Lastly, to meet the evaluation criteria of searching metadata, the group must either set up a separate relational database or integrate the metadata with the data.

In the next paragraphs, four of the most-prevalent and suitable time series databases will be considered: InfluxDB, Prometheus, Graphite, and TimescaleDB. They will be assessed in light of the aforementioned criteria.

InfluxDB
InfluxDB is an open source time series database developed for commercial integration on InfluxDB Cloud. The database is free to download and can be used on self-hosted servers. InfluxDB is originally a push-based system, but recently added support to its Kronograf for implementing pulling of data through the Prometheus scrapers [25]. It allows definition of retention policies that control how data is downsampled and removed. Through retention policies, InfluxDB can be instructed to meet the evaluation criteria of long term storage and data persistence.

The database has no integrated solution for storing and coupling metadata, which would require the setup and connection of multiple external relational databases to meet the evaluation criteria of searching metadata. The lack of metadata support is a drawback to InfluxDB as a viable candidate. Although InfluxDB requires more work to implement, it can still meet the evaluation criteria.

Prometheus
Prometheus is a free open source database optimized for time series. However, Prometheus uses a pull-based system [43]. An application must publish metrics at its endpoints and Prometheus periodically fetches this data. Prometheus provides an intermediary gateway in the *pushgateway*[42]. This gateway does allow for pushing data to the database, but only allows a single timestamp which must be the timestamp of the data upload, which is not necessarily the sensor timestamp at the time of measurement. Hence, it is unsuitable for the project which requires support for both regular frequential time series data and irregular event-triggered data, which means that Prometheus does not meet the evaluation

criteria of introducing new sensors.

Graphite

Graphite is an open source monitoring system with a three-component system: Graphite-Web, Carbon, and Whisper[19]. Graphite-Web is a web application providing data analysis, Carbon is a data-processing daemon, and Whisper is a time series database. Whisper has a defined resolution for data retention, meaning that it keeps a higher resolution for recent data and lowers resolution for the data as the data gets older[18]. This is incredibly useful in many use-cases based on the temporal locality principle. On the other hand, this downsampling violates the evaluation criteria of long term storage and data persistence.

TimescaleDB

TimescaleDB is a free open source relational database optimized for time series, built on top of PostgreSQL [53]. Similar to InfluxDB, it offers hosting in its cloud infrastructure in addition to allowing external hosting services. Being a relational database supporting full SQL, TimescaleDB supports event-based data insertion[51]. This allows for customized data handling, as well as setting up insertion-queries for event-driven, irregular data. The metadata can be stored for each time series in the database or a separate relational database. This results in an easy metadata-based search, saving the group large amounts of overhead to implement metadata-based querying. The database enables custom data retention policies through automated data management tasks and a scheduling framework[52]. To conclude, TimescaleDB meets the evaluation criteria for open source components, introducing new sensors, long term storage and data persistence, and searching metadata.

4.6 Choice of Technological Solution

For the visualization of time series data, the group quickly discovered that Grafana could not provide the complex user access management, complex graphs, or search in metadata, described in the section 4.4. Therefore, the group decided to create a front end application with the use of a graph library, with customized user access management. By making a front end application, the criteria of searching metadata can be met. Highcharts is the only of the three above-mentioned graph libraries that fulfills the criteria of complex graphs and was therefore the leading choice. Since Highcharts is maintained professionally, the group concluded that this would be a good choice for a long term library with little deprecation risk. Although Highcharts is not open source, the customer representative agreed that it would be a good choice as it is free for non-commercial use.

Of the four investigated databases, TimescaleDB is the database that best fulfills the evaluation criteria. It is an open source time series database that allows for customization of retention policies. It is built for both regular and irregular data while being a relational database and therefore allows metadata storage and querying. InfluxDB can also fulfill the evaluation criteria, but it lacks an integrated solution for storing and coupling metadata. With the time constraint on the project, TimescaleDB was chosen due to its advantage in metadata integration.

5. Development Methodology

In project development, it is important to have a methodology to follow to be able to apply methods and principles to a project. A methodology systematizes the project and enables quantifying a project's progress [35].

5.1 Choice of Methodology

There are many different methodologies to choose from, and one can also choose to combine certain parts of different methodologies if that is what fits the group best. This subsection explains the main methodologies that the group considered, with their respective pros and cons.

5.1.1 Scrum

Scrum is an agile framework, used to manage team projects, often within software development [45]. The main idea behind Scrum is to leave much of the responsibility to the development team. This way the team will have to be self-organized and make decisions in unity if problems occur. Scrum is usually the best fit for smaller teams, fast-moving development projects, and short sprints. However, Scrum has some disadvantages, such as scope creep and high change of project failure in case of uncooperative group members [3]. Scrum is also designed for teams working on the project full time, and daily meetings can be difficult to conduct for part time projects.

5.1.2 Extreme programming

Extreme programming (XP) is an agile framework following definite values and specific practices with a focus on high quality [14]. Some pros of using XP in this project are that it is intended for short projects, has practices for better communication, and improves teamwork [10]. Nonetheless, it is made for teams with a customer that shares location, constant changes that are hard to document, and tight deadlines.

5.1.3 Kanban

The lean method Kanban is focused on continuous work with management based on the workers' available capacity. It is mostly known for the Kanban board, which visualizes tasks and progress in a project [27]. Using the Kanban methodology increases flexibility, reduces the time cycle, and is an easy methodology to understand. Some downsides of the method are lack of timing and the complication of outdated boards [28].

5.2 Methodology in this Project

Scrum has been chosen by the group as the primary software development methodology. The methodology is enhanced with parts from XP and Kanban.

5.2.1 Scrum

The main source of methodology strategies and exercises has been inspired by the book *SCRUM AND XP FROM THE TRENCHES How We Do Scrum* [22]. The chosen strategies and exercises, and how the group incorporated them, are described below.

Sprints

As the project only lasts 12 weeks, the group wants to have short sprints. This was also influenced by the short iterations of XP. Therefore, the group decided that each sprint should span two working weeks. Each sprint starts with a sprint planning meeting and has a mid-sprint meeting on the first day of the second week. After the sprint is done, the group conducts a sprint demo and retrospective before planning the next sprint.

Backlog, User Stories, and Issues

After the group received the requirements specification from the customer, the group created the backlog. The backlog consists of all the requirements the customer wants to fulfill in the final solution, turned into user stories. These will later be divided into sub-tasks called issues. When creating the backlog, priorities were added to each story in compliance with the customer representative.

Sprint Planning Meeting

Before the sprint planning meeting, the backlog must be in a good state. As the name suggests, the next sprint is planned in this meeting. Estimation of Story Points is performed for the user stories in the backlog, which is explained in detail in section 5.2.1. Next, the group decides upon a sprint goal. The sprint goal, along with the priority of the user stories, decides which user stories to include in the sprint. Thereafter, the user stories are divided into issues and assigned to specific group members.

Usually, it is encouraged to have the customer take part in the sprint planning meeting. However, the customer representative did not have time, because of other meetings. Therefore, the group conducted the sprint planning without the customer representative and sent him the user stories to receive feedback.

Scrum Board

To maintain the different user stories, the group decided to have a Scrum board. As the group does not have a dedicated room or office, the group decided to use a software-based Scrum board. Jira was chosen as the Scrum board since it has a lot of built-in functionality to manage Scrum boards, such as priority and Story Points.

Planning Poker and Story Points

To estimate the Story Points required to finish a user story, the group decided to use Planning Poker. Planning Poker was used because estimates often fail when working with a new group and technologies [8]. To estimate the effort required for a user story, each group member estimates the number of Story Points. Next, the group members reveal their estimate to the rest of the group and the final estimate is calculated based upon the median. The available paper printed cards to select from were 0, $\frac{1}{2}$, 1, 2, 3, 5, 8, 13, 20, ?, ∞ and a coffee cup, which are shown in appendix Figure D.9. The coffee cup indicates that the group should take a break. Story Points are used instead of hours to make it easier to have correct estimates [7] [8].

Mid-Sprint Meeting

The group decided to have a mid-sprint meeting at the beginning of the second week of each sprint. This meeting would be used for a quick review and demonstration of finished user stories or issues [15]. The remaining of the meeting is used to work on the project together in the same room, to easier solve more complex problems.

Demonstration, Sprint Retrospective and Review

After the completion of a sprint, the group goes through the sprint to see what was finished and not. Thereafter, the group performs a demonstration of the user stories finished in the sprint for the customer representative. After the demonstration, the group performs a sprint retrospective. During the retrospective, the Scrum Master summarizes the sprint, followed by a roundtable discussion [33] of what was good, what could be better, and suggested improvements for the next sprint. Next, the group checks if the estimated Story Points were accurate for the finished user stories. The group also looks at reports and diagrams generated by Jira for the sprint, to see if there was anything noteworthy about them. An example of such a report is the burndown chart. Lastly, the Scrum Master creates actions to make sure that what could have been better is improved for the next sprint.

5.2.2 Extreme Programming

Since Scrum covers management and organizational practices, it can be supplemented with a methodology that focuses on programming practices [22]. This room for a supplement is why the group decided to incorporate some of XP's practices as part of this project's methodology.

Pair Programming

One main practice from XP is pair programming, where groups of two sit and code together on one computer [22]. The group used pair programming when a group member had a complex task that required help from someone with more knowledge of that subject. The group only used pair programming if both members agreed to.

Continuous Integration

Continuous integration was set up so that all tests would automatically run to check if everything still works as intended once a member creates a pull request. [22]. If a test fails, the group would be notified of the failed test and which part of the new code that caused it to fail.

5.2.3 Kanban Board

The group decided to use some elements from the Kanban board mixed with the Scrum board. Because of the group's little prior experience, the group wanted to have a flexible sprint backlog. Therefore, the group decided that sub-tasks of user stories are allowed to be modified during a sprint. However, the stories themselves are not allowed to be modified.

5.3 Rationale

As a result of the group members already being familiar with Scrum and XP, the group decided to mainly combine those methodologies. The group also discussed the usage of Kanban, Feature Driven Development, Crystal, and Dynamic Systems Development Method. Nonetheless, those methodologies were not as appropriate, due to the short project time and the size of the group. However, as mentioned above, the group did include a bit of the flexibility of the Kanban board into this project's methodology.

6. Requirements Specification

In this chapter, the requirements for the project will be discussed and divided into functional requirements and quality attributes. The final section consists of use case diagrams demonstrating some of the different ways that a user might interact with the system. Use case diagrams show a high-level overview of the relationship between use cases, actors, and systems [56].

Appendix E.1 shows the original requirement specification that the group received at the beginning of the project. The specification was general and did not go into much detail about how the customer envisioned the final solution. A few weeks later the group received a more detailed requirement specification for the final solution, which led the group to a more precise understanding of the tasks to be completed and the role of the group in the SINTEF ACE project. The detailed specification is not included in the appendix due to the document being confidential. The functional requirements from the latter requirement specification is shown in Table 6.1. The group started the development process based on the former requirements specification, believing that the requirements were flexible. The clarification of the project scope led to a re-evaluation of the project implementation.

6.1 Functional Requirements

This section describes the functional requirements for the final solution. The task of this project is to create a prototype of the final solution which means that the group is not supposed to complete every functional requirement but prioritize the ones necessary to create a minimum viable product. The requirements are given a priority ranging between lowest, low, medium, high, and highest. The motivation of the priorities is to facilitate the process of determining which functional requirements to be implemented in this project. The priorities are made in consultation with the customer. The functional requirements and their priorities are presented in Table 6.1.

Table 6.1: The functional requirements of the product

ID	Description	Priority
FR1	As an engineer, I want to be able to add new data with different formats.	Highest
FR2	As an engineer, I want to be able to save metadata connected to time series.	Highest
FR3	As a researcher/user, I want to be able to view data.	Highest
FR4	As a user, I want to be able to access an API to retrieve data.	Highest
FR5	As an engineer, I want to be able to add new sensors.	High
FR6	As an engineer, I want to be able to upload data manually.	High
FR7	As an engineer, I want to be able to modify metadata for a time series.	High
FR8	As a researcher/engineer, I want to be able to store large amounts of data.	High
FR9	As a user, I want to be limit data viewing with authorization.	High
FR10	As a researcher/user, I want to be able to visualize data easily (private dashboard).	High
FR11	As a user, I want to be able to view the data on the web.	High
FR12	As a researcher, I want to be able to download relevant data.	High
FR13	As an engineer/researcher, I want to get notified of a loss of sensor data.	Medium

Continued on next page

Table 6.1 – continued from previous page

ID	Description	Priority
FR14	As an engineer, I want to be able to write custom drivers to send in data.	Medium
FR15	As a user, I want to be able to search for time series with metadata.	Medium
FR16	As a customer, I want to view data on a static page.	Medium
FR17	As a researcher, I want the data to be transparent.	Medium
FR18	As a researcher, I want to be able to access unmodified data from 10 years ago and unmodified or modified from 20 years ago.	Medium
FR19	As an engineer/researcher, I want to get notified of irregular time series.	Low
FR20	As a user, I want to be able to see time series metadata from Landax.	Low
FR21	As a user, I want to be able to aggregate data.	Low
FR22	As an engineer, I want to be able to save independent file formats and view connected metadata.	Lowest
FR23	As a user, I want to be able to interact with graphs.	Lowest

6.2 Quality Attributes

This section presents the quality attributes that the group and customer consider as architectural drivers for the project. Quality attributes are non-functional requirements used to evaluate the performance of a system. The chosen quality attributes are modifiability, usability, and security. Quality attribute scenarios are used to specify the quality attribute requirements in a common form. According to [30, p. 68], a scenario is defined in six parts:

- **Source of stimulus**: The trigger of the scenario.
- **Stimulus**: What is done to trigger the scenario.
- **Artifact**: Part of the system being affected.
- **Environment**: The state of the system.
- **Response**: How the system reacts to the stimulus.
- **Response measure**: How the reaction is measured.

6.2.1 Modifiability

Given a task with many possibilities for expansion of functionality, a modular architecture is often a wise decision. Modifiability defines how easily new functionality can be added to the system, and how easily already existing functionality can be extended by developers [30, p. 117]. A modifiable system in this project will allow the product to scale, and the consequences will be less severe if the product changes direction. Table 6.2 and 6.3 show the scenarios for modifiability.

Table 6.2: Modifiability scenario - M1: Add new request

Source of stimulus	Developer
Stimulus	Add a new request to the API
Artifact	Back end
Environment	Design time
Response	Query added and accessible
Response measure	Within 4 hours

Table 6.3: Modifiability scenario - M2: Add data from a new file format

Source of stimulus	Developer
Stimulus	Add data from a new file format
Artifact	Back end
Environment	Design time
Response	Component created and connected
Response measure	Within 1 work day

6.2.2 Usability

One of the goals for this project is to create a system where the user easily can learn how to use the product. This is challenging due to the variety in user groups and their different experiences working with graphs and querying data from a database. Usability is concerned with how easy it is for the user to accomplish the desired task and the kind of user support the system provides. Usability has proven to be one of the cheapest and easiest ways to improve a system's quality from the user's perception [30, s. 175]. A user guide has also been created to further improve the usability of the system and is shown in appendix A.1. Table 6.4 and 6.5 show the scenarios for usability.

Table 6.4: Usability scenario - U1: Create dashboard for measurements

Source of stimulus	Researcher
Stimulus	Create a new dashboard
Artifact	Graphical user interface
Environment	Runtime
Response	Understand how to create a dashboard with a graph cell
Response measure	Within 3 minutes

Table 6.5: Usability scenario - U2: Modify metadata

Source of stimulus	Engineer
Stimulus	Modify metadata for a sensor
Artifact	Graphical user interface
Environment	Runtime
Response	Understand how to change and save metadata for a sensor
Response measure	Within 3 minutes

6.2.3 Security

The data from sensors should be stored securely because the data is confidential. Due to the confidentiality of the data, only authorized users should have access to the data in the database. Table 6.6 and Table 6.7 show the scenarios for security. Plausible security threats regarding this project and how to assess them are further discussed in chapter 9.

Table 6.6: Security scenario - S1: Admin page access

Source of stimulus	Unauthorized user
Stimulus	Attempts to enter admin page
Artifact	Graphical user interface
Environment	Normal operations
Response	Page is protected from unauthorized access
Response measure	Immediately

Table 6.7: Security scenario - S2: Dashboard access

Source of stimulus	Unauthorized user
Stimulus	Attempts to fetch dashboards
Artifact	API
Environment	Normal operations
Response	Data not accessible without valid credentials
Response measure	Immediately

6.3 Use Case Diagrams

Figure 6.1 shows a use case diagram for the main part of the product. The use case diagram models the details of the system's users (also known as actors) and their interaction with the system [56]. The dotted square defines the system boundary which contains the features and functionality of the system. The actors to the left (engineer, researcher, and customer) are users of the system and their relations to possible actions in the system are illustrated by arrows. The last actor, Authentication, is a separate service that provides authentication to the system without utilizing the system directly. The motivation of creating a use case diagram is to visualize the scope of the system, which goals the system helps the actors achieve, and how the system interacts with users and external systems [56].

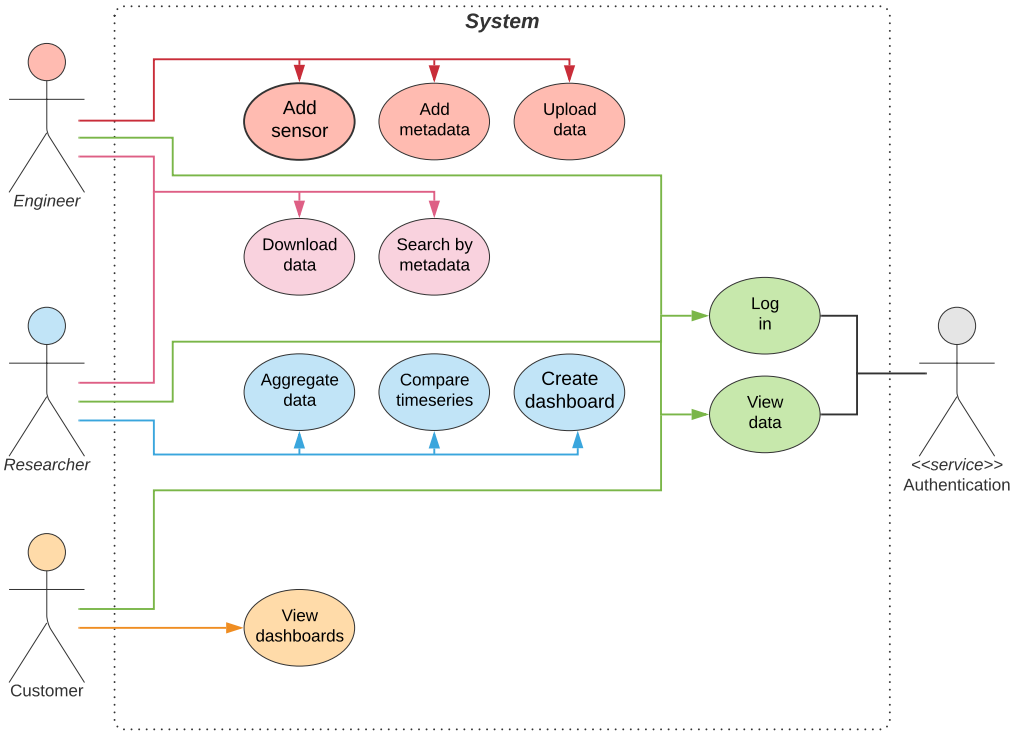


Figure 6.1: Use Case Diagram

7. Architecture

This chapter explains the project's architecture design and the rationale behind it. The architectural drivers were functional requirements, quality attributes, and business requirements, which can be viewed in chapter 6.

7.1 Architectural and Design Patterns

Table 7.1: The architectural and design patterns used in the architecture design

Pattern	**Description**
Client-Server	The client-server pattern consists of numerous clients and a server. This is useful for the project's ability to serve multiple users with access to the application and database. The server and client will communicate through a GraphQL API, see appendix A.2.2 for more information.
Model-View-Controller (MVC)	To achieve a modular and scalable architecture, the model-view-controller pattern will be used. The architecture divides the design into three main parts: the front end as the view, the back end as the controller, and the database as the model.
Service-Oriented Architecture (SOA)	This pattern describes a collection of distributed components, which can use and provide services. This is useful in this project as it has multiple external components that it will have to communicate with and use. For example, the usage of Microsoft Authentication, and having an API open for subscription by external systems.
Publish-Subscribe	The messaging pattern publish-subscribe is a special usage of the observer pattern. The pattern will be used in the project to let external systems subscribe to different data streams, and is alerted automatically when an event, such as the upload of new data, occurs.
Shared Data	By having multiple data accessors, the usage of a shared data store will be utilized. All database traffic will pass through the shared data store named "Database".
Decorator	Decorator is a design pattern that is used to add additional functionality to an object without affecting it. It will be used to make the page more secure through access restrictions.

7.2 4 + 1 Architectural View Model

To describe and visualize the project's architecture, the 4+1 architecture view model will be used [29]. It consists of four main view types: logical, process, development, and physical. Each representing different stakeholders' points of view. Table 7.2 gives a short description of the different views, and how they are represented and used in this project. To ensure an easier understanding of the architectural views, all diagrams use the same colors for the same elements.

7.2.1 Logical View

The logical view focuses on the end users' functionality perspective and decomposing the system down to a few key abstractions [29]. This view will look more closely at the object-oriented decomposition, and how the different patterns have been utilized in the architecture. The group has created a simplified class diagram, shown in Figure 7.1, to aid

Table 7.2: The group's selection of architectural views.

View	Purpose	Target audience	UML Notation
Logical view	Illustrates what features the finished system will be able to provide for end users.	Developers, course staff, end users, customer.	Class diagram
Process view	Shows how the system works during runtime, and how communication occurs between different components of the system.	Developers, course staff, end users, customer.	Sequence diagram
Development view	Depicts sub-systems, packages and libraries used to create the main components of the system.	Developers, course staff, customer.	Package diagram
Physical view	Describes the physical topology and how the software components maps to the hardware.	Developers, course staff, customer.	Deployment diagram

with the description of the logical view. To increase readability, the diagram is simplified by not showing all classes and functional components, and also leaving the variables and methods hidden.

The model-view-controller architectural pattern is used as a main guideline for the architecture. The pattern is used to divide the application into three main parts: front end, back end, and database, where these are respectively: the view, the controller, and the model.

Another well known and used pattern for the project's architecture is the client-server pattern. JSON elements will be sent from the front end to the back end as a POST HTTP/HTTPS request. The application will implement the client-server pattern with the usage of GraphQL API. The pattern enables multiple users to access the API, which retrieves the requested information from the database.

The service-oriented architecture (SOA) pattern will be one of the most important patterns for the project's architecture. This is because the architecture should be able to sign in with Microsoft Authentication, give access to data to external systems, and also use other systems to retrieve data. In the architecture represented in Figure 7.1, one can see both the front end communication with Microsoft authentication (Azure Active Directory), as well as the API communication with external systems. As such, the system will be able to provide service consumers with service providers. In the SOA pattern, the API can also be seen as an enterprise service bus (ESB), because it is an intermediary element between service providers and consumers. The API will also be able to perform security checks when requests from web application or external systems arrive.

For external systems to be able to retrieve data during runtime and also keep up to date, the publish-subscribe pattern will be used. The external systems will be able to communicate with the Subscriptions part of the API, shown in Figure 7.1. The designed publish-subscribe system transfers data through the use of WebSockets [59], which enables a two-way stream of data, with event-driven responses between the client and the server. The usage of the publish-subscribe pattern is discussed further in appendix A.2.6.

The folder named Database in Figure 7.1, will be the architecture's usage of the shared data pattern. The Database folder is viewed as the shared data store, where all data accessors have to pass through to be able to communicate with the database. The communication between the database and the Database folder is shown in the class diagram in Figure 7.1, as the Database folder is the only one associated with the database.

The decorator pattern is used in the front end of the application, in the functional component AccessCheckerDecorator. This is a decorator-wrapper used around the main pages of the application. The decorator connects with the Microsoft Authentication system and validates whether a user has valid access to the page they are trying to reach with the URL.

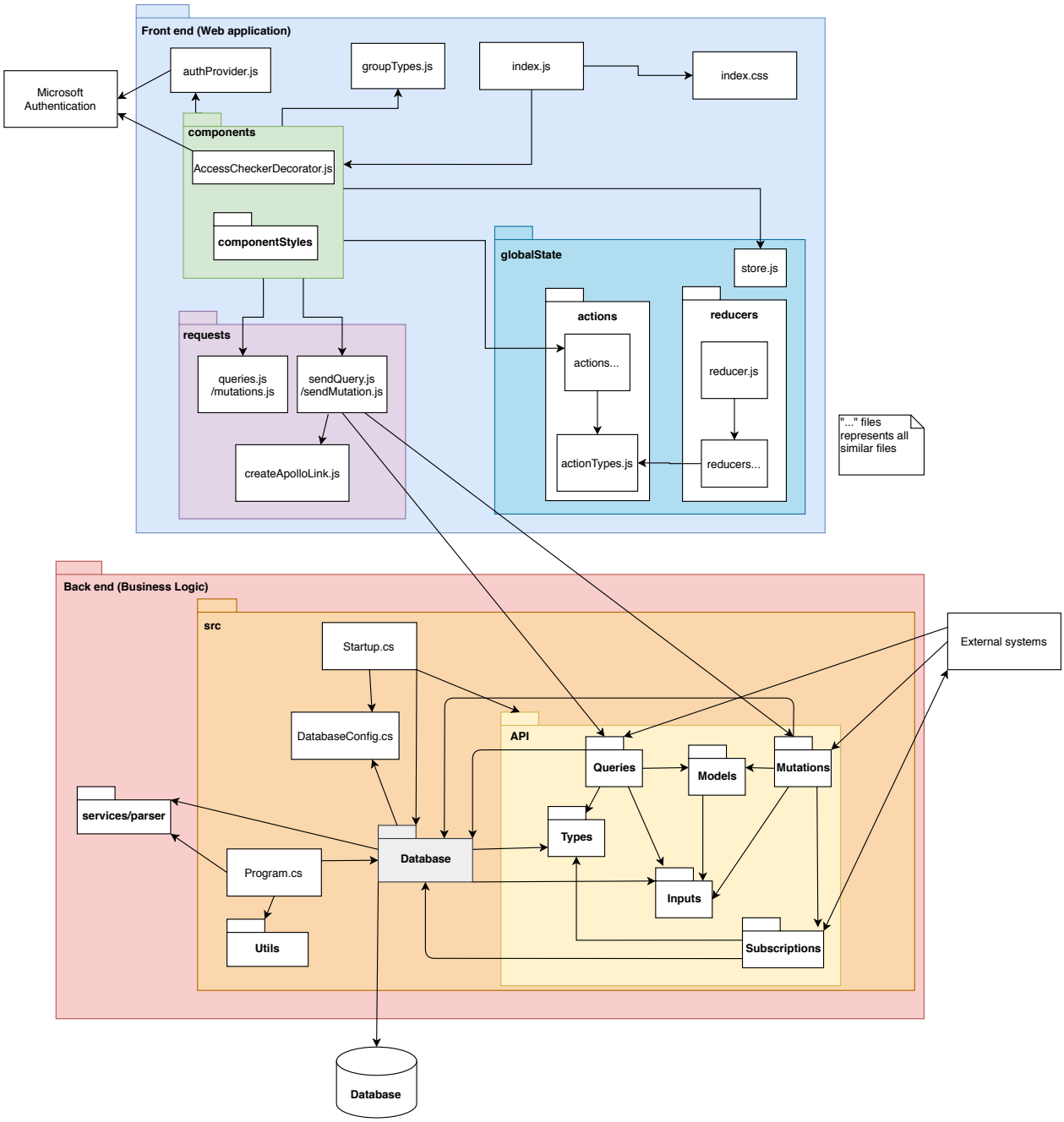


Figure 7.1: Simplified class diagram with UML-notation, representing the logical view, which shows the most important packages and classes/components, and their associations with other parts of the system.

7.2.2 Process View

The process view focuses on the flow of the system's central functionality during runtime [29]. To visualize the communication and synchronization, the group has created two sequence diagrams. Figure 7.2 illustrates the data flow when a sensor sends its data to the system, while Figure 7.3 shows the process of retrieving data from the database to the web application.

In Figure 7.2, the first actor is the sensor, which may be one of several types of sensors such as a thermometer or air saturation sensor. When the sensor has gathered new data, the data is sent to the API with the UploadDataMutation call. The data and related configuration are first encoded by the client before it is sent to the server through the UploadData mutation. It then continues to deserialize the encoded data and configuration received. The data and configuration are subsequently transferred to the parser service, where the configuration is used to decode the uploaded data file, and the data is sent to and stored in the database. After the data is uploaded to the database, an event is triggered. This event is caught by the GraphQL subscription, triggering the server to send the new data to the subscribed clients.

Figure 7.3 illustrates the workflow when a user tries to retrieve data from the database, which in this example, the user's dashboards are fetched. The process starts with the user clicking into the Dashboards tab, followed by using sendQuery to fetch the dashboards for the user. The user's access token is also sent as part of the header, however, this is not shown in the diagram. The query is sent to the GraphQL API endpoint which directs it to the Database folder, where there the correct data is fetched from the database and sent back to the web browser. The browser re-renders and shows the retrieved dashboards.

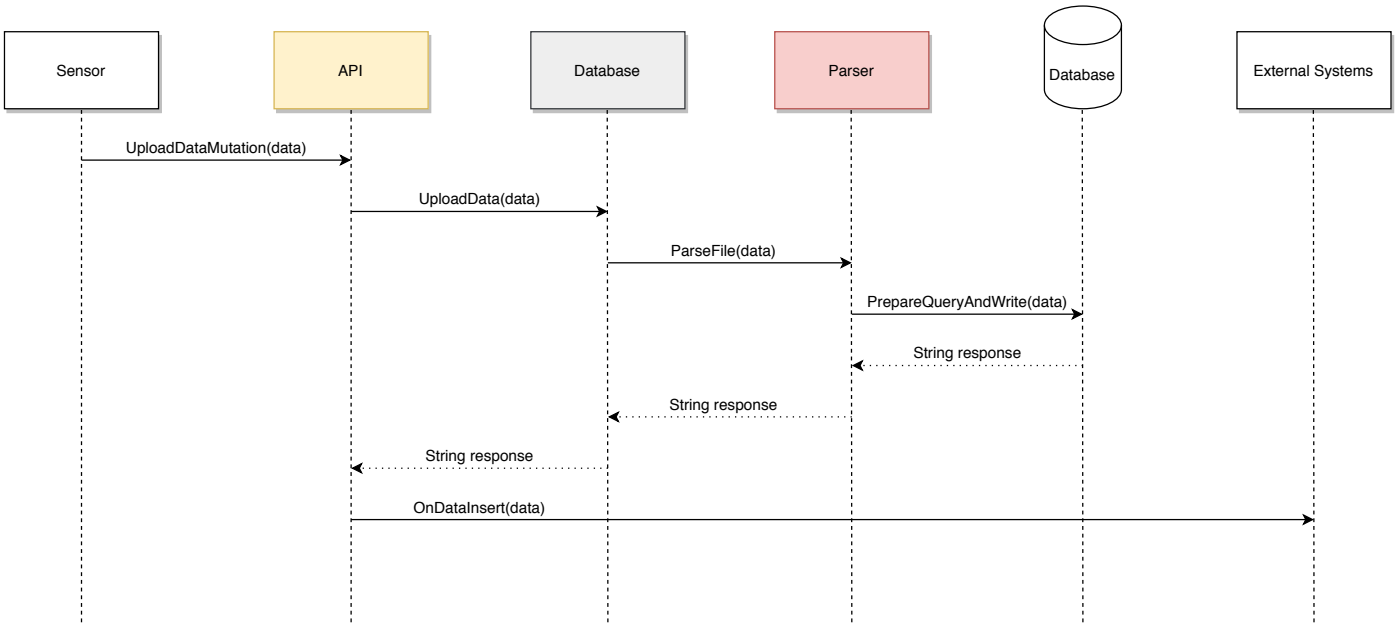


Figure 7.2: Sequence diagram representing the process view. Shows the process of uploading data from a sensor.

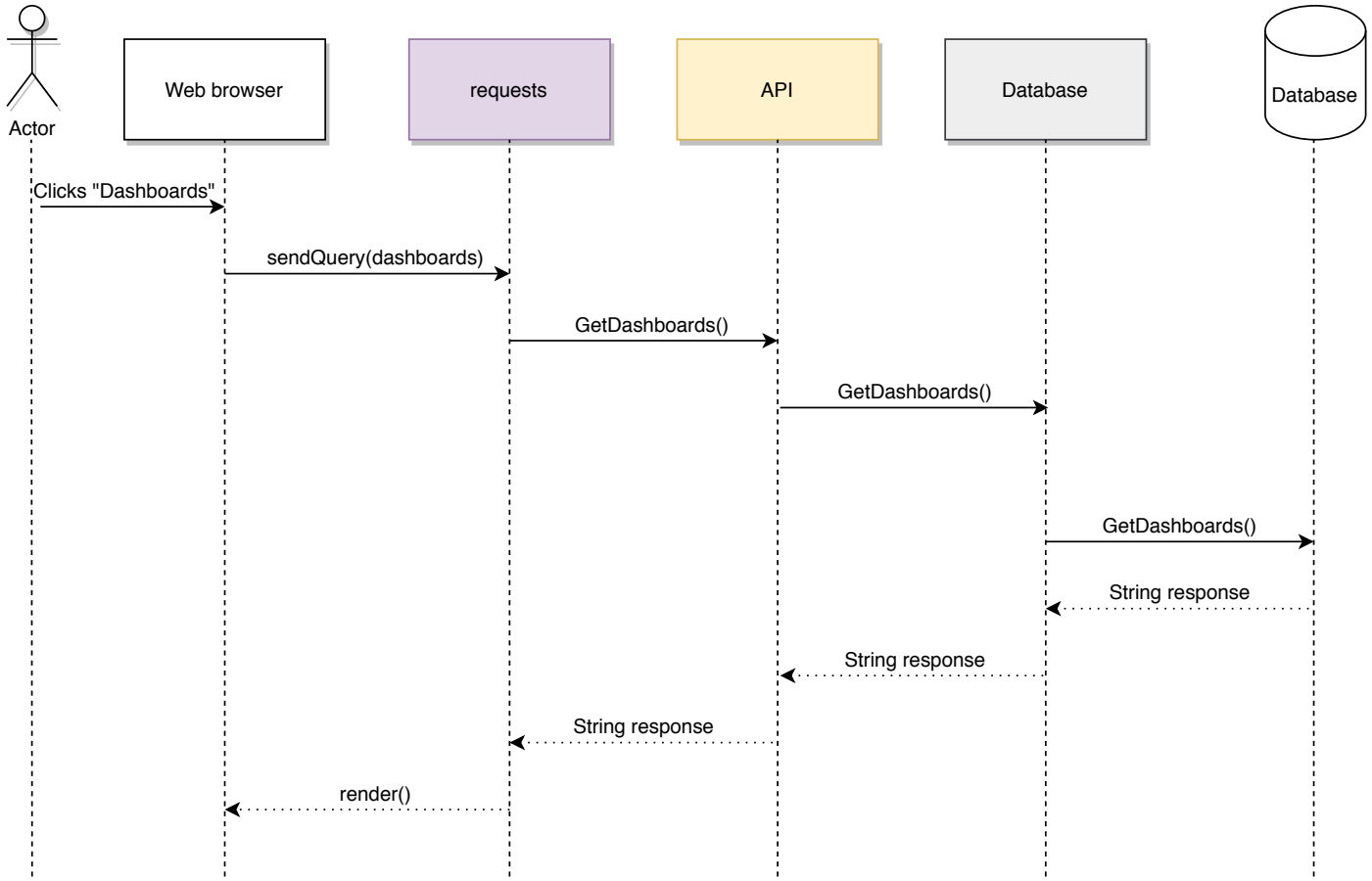


Figure 7.3: Sequence diagram representing the process view. Shows the process of retrieving dashboard data from the database to the web browser.

7.2.3 Development View

The development view focuses on the actual organization of the software modules, packages, libraries, and subsystems [29]. To visualize this organization of exports and imports between different system elements, the group has created a package diagram, shown in Figure 7.4. The diagram displays the relationship between packages with use-arrows, showing their dependencies. A more detailed version of the package diagram is shown in appendix Figure D.15.

The package diagram in Figure 7.4 shows all important packages, both internal and external. In the diagram, one can also view where the various test folders will be placed. As the front end is built using the react library, several react packages will be used. Most noteworthy: react-redux, highcharts-react-official, and react-aad-msal. The web application will also use an Apollo Client. More information about the packages can be found in appendix A.3.1. Figure 7.4 also shows back end's usage of the most important packages: HotChocolate, Newtonsoft, Npgsql and AspNetCore. Information about the packages used in the back end is located in appendix A.2.2.

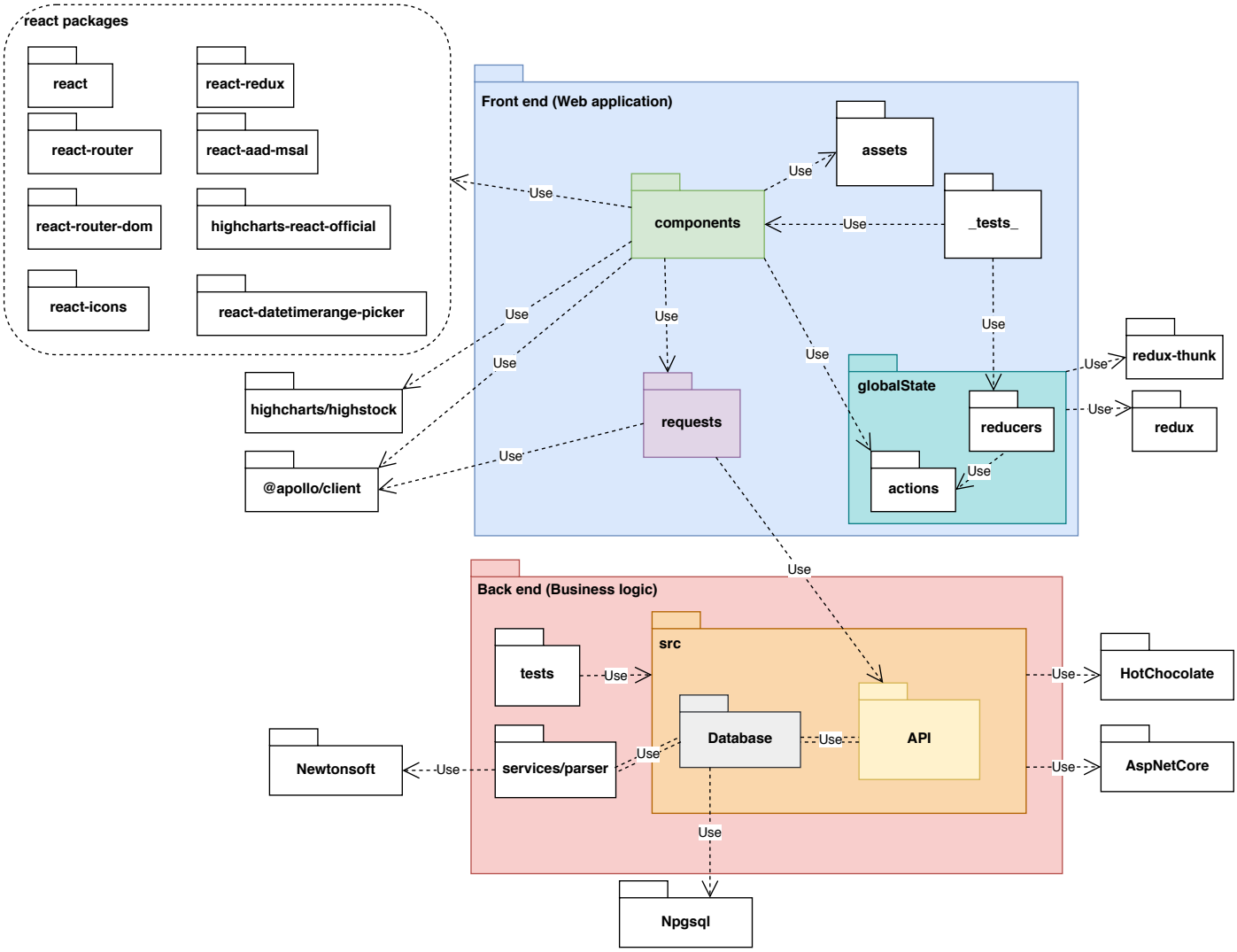


Figure 7.4: Package diagram representing the development view, showing dependencies between packages.

7.2.4 Physical View

The physical view shows the mapping between software components and the hardware [29]. Figure 7.5 illustrates a deployment diagram of the physical systems on which different parts of the application will run. As shown in the diagram, all of the main system components are running on the same Microsoft Azure server. The front end is also located inside an NGINX service. Outside of the Microsoft Azure server, web users can access the web application through the browser, and the web application communicates with the Azure AD to sign in the user. External systems that want to subscribe to different data can do so directly with the business logic.

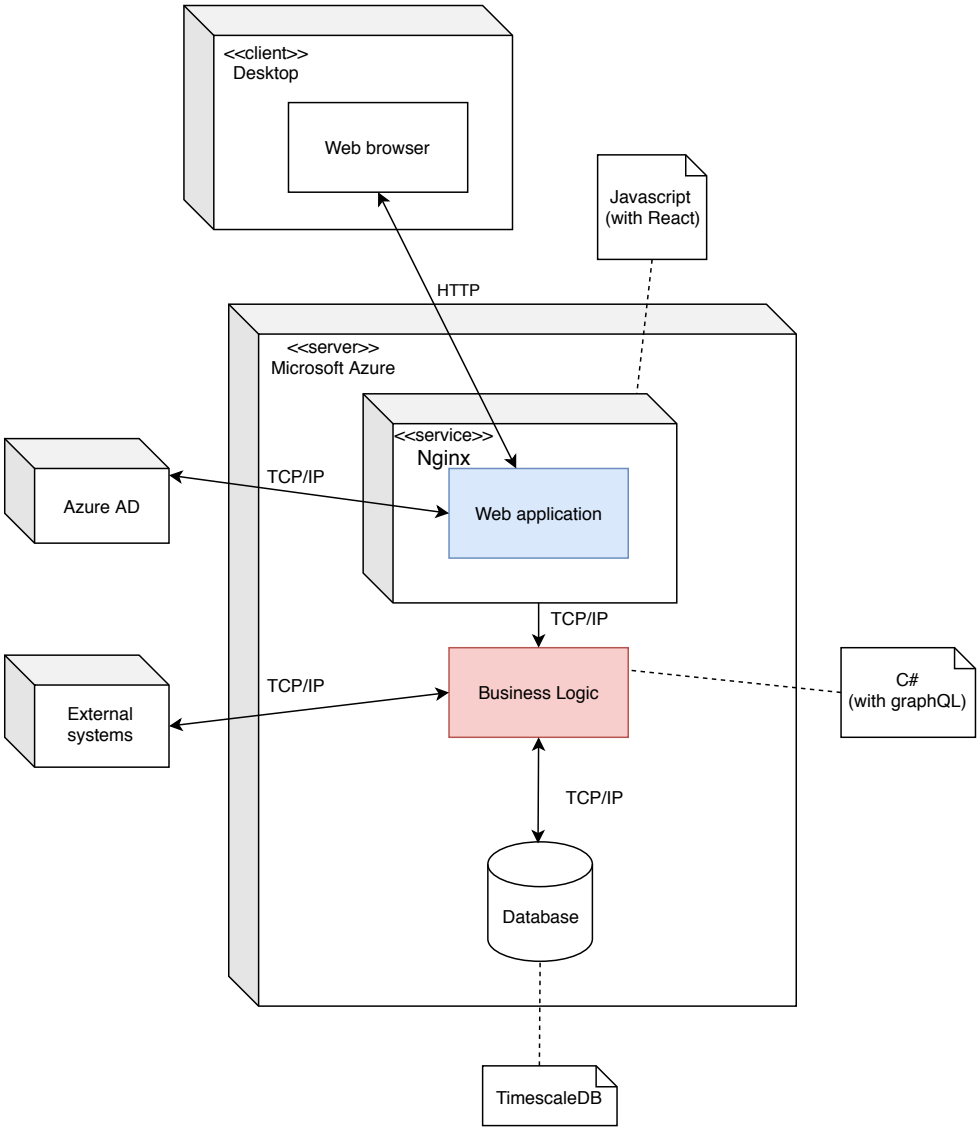


Figure 7.5: Deployment diagram representing the physical view.

7.3 Architectural Rationale

The group believes this architecture can be used as a design to fulfill both the functional and non-functional requirements, and that it satisfies the primary quality attributes. By having multiple users that want to retrieve data, the client-server pattern is a well-known solution. However, one must remember that the server can become a performance bottleneck and or a single point of failure. So in the final solution, one can utilize multiple servers or at least a cold spare.

In addition to utilizing a client-server pattern, the design is also a service-oriented architecture (SOA), which can be viewed as a special version of the client-server pattern. Since this system should communicate and utilize different services, the components of the pattern, such as an enterprise service bus (ESB) and orchestration server, are very useful. Creating the API as an ESB promotes interoperability and two of the primary quality attributes for this system: modifiability and security. Since this design is for a small proof-of-concept project, the architecture does not utilize an individual component as an ESB and does not have an orchestration server. However, it might be smart to consider implementing these in the final solution.

The model-view-controller pattern is a well-known application structure which gives the architecture higher modifiability. For example, it gives the possibility to change the look and feel of the front end, while not having to change any of the business logic, which is why the group used this pattern in the architecture.

As external systems want to be updated as soon as new data is stored in the database, the publish-subscribe pattern is a favorable solution for this context and problem. By making the components ignorant to the other's existence, it increases modifiability and scalability regarding adding and removing sensors and listeners, while increasing latency. Since performance is not a primary quality attribute for the customer, this trade-off is acceptable. The architecture design of subscribing to certain data flows through the GraphQL API is functional and scalable. However, the current design is based on ease of implementation for a proof of concept. A preferable, but more time-consuming implementation, would be the Data Distribution Service (DDS). This is a better publish-subscribe implementation for the domain of use. DDS is recommended to be made in place of the current designed implementation, without needing adaptation. For more information about this pattern and its features, additional reading material is located in appendix A.2.6.

By having a shared data store, modifiability is increased as it decouples producers from consumers. It also centers around qualities such as scalability, security, availability, compatibility, and data consistency, which are highly valued qualities for this project. Nonetheless, the pattern can have negative effects similar to those of the client-server pattern, where the shared data store can become a performance bottleneck and a single point of failure. Therefore, the final solution may consider a cold spare or multiple shared data stores.

8. Implementation/Sprints

This chapter contains documentation on how the preliminary study and sprints were conducted. The project started with two weeks of preliminary studies, followed by three coding sprints. The sprints were designed based on the group's chosen methodology, which is explained in section 5.2. Each sprint consisted of a planning meeting, backlog, review, retrospective, demo, changes, and feedback from the customer.

8.1 Preliminary Study

The preliminary study started right after the first group meeting and ended when the first sprint began, from August 25 to September 9 for a total of two weeks. As part of this phase of the project, the group's focus was on getting to know each other, the customer, the advisor, and a clear understanding of the customer's task.

As part of getting to know each other, the group decided to have a social event, and also conduct an Experts in Teamwork (EiT) exercise. The social event was a platform for the group members to get to know each other, which in turn created a better work environment. By using the EiT exercise competence triangle [37], the group members in turn shared practical, theoretical, and personal knowledge. The exercise brought the group closer and increased the group's awareness of what expertise was present in the group.

8.1.1 Group Contract

During the first group meeting, a group contract was created. The contract was made according to the EiT teamwork contract's recommendations [37], as well as aspects of group contracts previously used by group members. The group decided to include group rules, where a rule violation required the respective member to bring cookies to the next meeting. Some of the group rules are to meet at the agreed-upon time, EiT's check-in and check-out [37], and Scrum's daily stand-up [22] in meetings. The group contract can be found in appendix E.

8.1.2 Product Backlog

Prior to any sprint planning, the group created a product backlog. This backlog consisted of all user stories for the project and was created by having the group members go through the customer's requirements specification together. The group created all necessary user stories to represent the scope of the final solution. After creating the stories, the group added priority to the stories on a scale of lowest, low, medium, high, and highest. After the product backlog was completed, the group presented it to the customer representative. The customer representative made suggestions to split up some of the stories, as well as some minor priority changes. After these alterations were executed, the backlog was approved.

8.2 Sprint 1

The first sprint started on September 9 and ended on September 17, for a total of one week. The group decided that all sprint names would be fish-related and go from small to larger fish. Therefore, the first sprint was named "Smolt". The goal for Sprint 1 was *General architecture and technology stack, overall design and prototype*, which included researching what technologies to use, designing the architecture, and setting up the environment.

8.2.1 Sprint Planning Meeting

The sprint planning meeting took place at September 7. Firstly, the group decided upon the sprint goal, sprint duration, and time of the review, retrospective, and demo. Furthermore, the group settled on a definition of when a story is considered done, which was after it successfully goes through testing and is checked by the Quality Manager.

8.2.2 Sprint Backlog

Since the sprint goal was to research, and not to start the development of the project, the group did not include any of the user stories in this sprint. Instead, the group made tech stories [22], which can be viewed in Figure 8.1. As the tech stories were fundamentally theoretical of nature, effort estimation proved difficult. Therefore, the group decided not to estimate Story Points for these stories.

8.2.3 Sprint Review

At the sprint review, the group members presented their work in turn, explaining what was and was not finished. The stories which were addressed are shown in Figure 8.1. Finally, the group went through the product backlog and updated it.

Completed Issues

View in Issue Navigator

Key	Summary	Issue Type	Priority	Status	Story Points (-)
FFD-43	Designing the architecture	Task	Highest	DONE	-
FFD-44	Look at alternatives to influxDB	Task	Medium	DONE	-
FFD-45	Look at metadata database (ex index DB)	Task	Medium	DONE	-
FFD-46	Explore possibilities for Grafana integration with React application	Task	Medium	DONE	-
FFD-47	Rough visual prototype	Task	Medium	DONE	-
FFD-50	Test framework exploration (ex selenium)	Task	Medium	DONE	-
FFD-52	Integration between jira and github	Task	Medium	DONE	-
FFD-86	Explore which server to use	Task	Medium	DONE	-

Issues Not Completed

View in Issue Navigator

Key	Summary	Issue Type	Priority	Status	Story Points (-)
FFD-49	Pipeline setup for CI / CD (proof of concept)	Task	Medium	TO DO	-
FFD-53	Explore parsers (generalization. Files or own DB)	Task	Medium	TO DO	-
FFD-84	Set up environment	Task	High	IN PROGRESS	-

Figure 8.1: Screenshot from Jira of which tasks were and were not completed during Sprint 1.

8.2.4 Sprint Retrospective

During the retrospective, the group discussed significant sprint changes, what went well, what could be better, and actions to follow for the next sprint. The team followed the retrospective plan and used post-it notes and were given some time to write down good things and what could be better. The members were then allotted 3 points each to distribute upon the "could have been better" notes. The notes with the most points were addressed with actions to improve upon in the next sprint. A photo of how the retrospective board looked is placed in appendix D.16. The main positive notes were communication, flexibility, punctuality, completion of issues, teamwork, and report writing. The aspects that could have been better and their corresponding actions are shown in Table 8.1.

Table 8.1: What could have been better in Sprint 1 and their corresponding actions

Could be better	**Action to improve**
Broader participation in discussions.	Roundtable discussion [33] when a discussion occurs.
Clearer communication with supervisor.	Send email to supervisor for clarification after meetings.
Limit unnecessary meetings.	Joint responsibility for meeting agendas.
More consistent time management among members.	Estimate Story Points on stories and distribute work thereafter.

8.2.5 Sprint Demo

At the start of the Sprint 1 demonstration, the group explained how the sprint went, as well as important decisions and changes. Following this, the group went through all finished user stories, for example showing the architectural design and justifications of the technological choices. The customer representative seemed pleased with the results of the first sprint and did not have any comments on the work done so far.

8.2.6 Sprint Changes

The group wanted to have the sprint planning meeting earlier. However, because of the slow release of the requirements specification from the customer, sprint planning was delayed a week.

The sprint was originally planned to be two weeks, rather than just the one. However, the group noticed that they could finish the stories in just one week. Also, according to recommendations from the advisor, development should start week 4 of the project. By using only one week instead of two, the project development would start in week 4 instead of week 5 of the project, and bring the group up to speed with the advisor's estimate. Therefore, half-way into the sprint, the length was changed.

8.3 Sprint 2

The second sprint, named "Salmon", started on September 21 and ended on October 2, totaling in two working weeks. The sprint goal for Salmon was to *Create a minimum viable product*. The group concretized the sprint goal into more technical terms of how detailed the minimum viable product would be: finish design, implement graphs and general front end layout, create a parser for existing formats, set up timescale, general back end calls, and authentication.

8.3.1 Sprint Planning Meeting

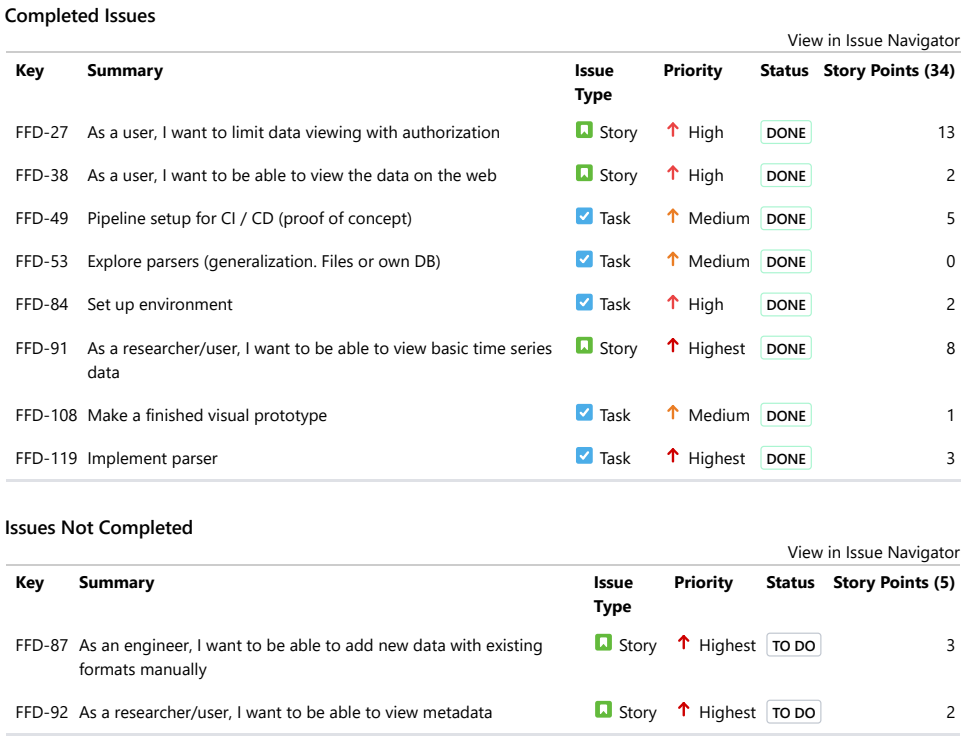
Sprint 2's sprint planning meeting took place at September 21. The group followed the same procedure as in the previous sprint, with one exception, Planning Poker, which is explained in section 5.2.1. Before the group decided which stories would be part of the sprint, Planning Poker was used to estimate Story Points for all user stories with a priority of Highest or High. The reason why the whole product backlog was not estimated, was the time limit and the group's little knowledge of the accuracy of the group's first estimates.

8.3.2 Sprint Backlog

Since the goal of the sprint was to create a minimum viable product, the group added a few tech stories but tried to add as many user stories as possible. During the Planning Poker, the group members took a rough estimate of one Story Point being about 4 man-hours. Sprint 2 was two working weeks long, and the group members were expected to work 24 hours a week. Taking into consideration the large overhead of hours due to lectures, meetings, and administrative work, the group planned for 40 Story Points. The finalized sprint backlog can be viewed in Figure 8.2, where the numbers on the right are the amount of Story Points for that story.

8.3.3 Sprint Review

The sprint review was carried out in the same manner as the previous sprint review. The stories which were completed and not can be viewed in Figure 8.2. Several of the stories were finished, however, there were a few unfinished stories. The group decided to add the non-finished stories to the backlog of the next sprint. Burndown charts for Story Points and issues completed throughout the sprint is located in appendix D.5.2.



Completed Issues

View in Issue Navigator

Key	Summary	Issue Type	Priority	Status	Story Points (34)
FFD-27	As a user, I want to limit data viewing with authorization	Story	High	DONE	13
FFD-38	As a user, I want to be able to view the data on the web	Story	High	DONE	2
FFD-49	Pipeline setup for CI / CD (proof of concept)	Task	Medium	DONE	5
FFD-53	Explore parsers (generalization. Files or own DB)	Task	Medium	DONE	0
FFD-84	Set up environment	Task	High	DONE	2
FFD-91	As a researcher/user, I want to be able to view basic time series data	Story	Highest	DONE	8
FFD-108	Make a finished visual prototype	Task	Medium	DONE	1
FFD-119	Implement parser	Task	Highest	DONE	3

Issues Not Completed

View in Issue Navigator

Key	Summary	Issue Type	Priority	Status	Story Points (5)
FFD-87	As an engineer, I want to be able to add new data with existing formats manually	Story	Highest	TO DO	3
FFD-92	As a researcher/user, I want to be able to view metadata	Story	Highest	TO DO	2

Figure 8.2: Screenshot from Jira of the Sprint 2 report.

8.3.4 Sprint Retrospective

This sprint retrospective was carried out in the same manner as the previous sprint retrospective. Several members were absent and one member participated digitally. The main things that went well were communication, completion capability, pleasant meetings, report writing, and that it was easy to ask other group members for help. The note with the most points regarding what could have been better was vague and overlapping sub-tasks. The aspects that could be better and their corresponding actions are shown in Table 8.2.

Table 8.2: What could have been better in Sprint 2 and their corresponding actions

Could be better	**Action to improve**
Vagueness in the description of sub-tasks.	Formulate concise, specific, and independent sub-tasks.
Code quality.	Be more strict when reviewing pull requests.
Lack of communication between the front end and back end development teams.	Use the upcoming Thursday to discuss the product and its development.

8.3.5 Sprint Demo

The group conducted acceptance testing, going through the product implementation completed throughout the sprint. The customer introduced previously unknown details regarding the project's application and data, enabling us to make the necessary changes at an early stage in the development.

8.3.6 Sprint Changes

The task of user authorization proved to be too large and was therefore split into two user stories pertaining to the back end and the front end implementation, separately. The front end story was considered completed, while the back end story was moved to sprint 3.

8.4 Sprint 3

The third sprint, named "Salmon, but older", started on October 5. and ended on October 16. This encompasses the working days of two weeks. The goal of the sprint was to *Complete the main traits of the product regarding data flow*. The group concretized the sprint goal into achievable goals, which were metadata viewing and modification, data upload, data visualization, data flow through API, authorization, and authentication.

8.4.1 Sprint Planning Meeting

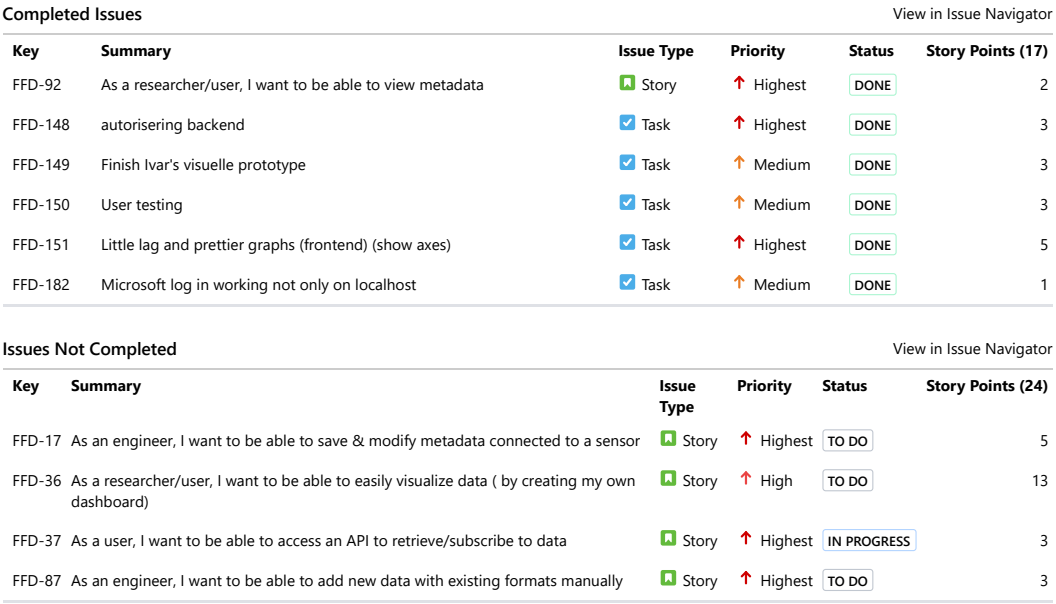
The sprint planning meeting took place October 5. The customer representative approved the sprint plan, following the Sprint 2 demonstration. The group followed the same procedure for the sprint planning meeting as in the previous sprint. Planning Poker was used to estimate Story Points for the stories with Medium priority and the additional tech stories. The group chose not to estimate the remaining stories, as there was not enough time to finish the stories with priority Low or Lowest. The Planning Poker process was a bit more challenging in this sprint, due to one team member attending digitally, and the use of paper printed cards.

8.4.2 Sprint Backlog

To reach the sprint goal, the group found it necessary to add tech stories, mostly to improve the minimum viable product created in Sprint 2. The group planned to finish 41 Story Points in Sprint 3, including the stories not completed in Sprint 2. An overview of the stories included in this sprint is shown in Figure 8.3.

8.4.3 Sprint Review

This sprint review was carried out in the same manner as in the previous sprint. An overview of the issues addressed in the sprint is shown in Figure 8.3. Four of the stories were not completed in this sprint. There were two main reasons for this: poor estimation of Story Points and dependent work. Although some of the stories were not finished in this sprint, much work was finalized within the stories. The group decided to add the stories that were not completed to the backlog of Sprint 4. Burndown charts for Story Points and issues completed throughout the sprint is located in appendix D.5.3.



Completed Issues — View in Issue Navigator

Key	Summary	Issue Type	Priority	Status	Story Points (17)
FFD-92	As a researcher/user, I want to be able to view metadata	Story	Highest	DONE	2
FFD-148	autorisering backend	Task	Highest	DONE	3
FFD-149	Finish Ivar's visuelle prototype	Task	Medium	DONE	3
FFD-150	User testing	Task	Medium	DONE	3
FFD-151	Little lag and prettier graphs (frontend) (show axes)	Task	Highest	DONE	5
FFD-182	Microsoft log in working not only on localhost	Task	Medium	DONE	1

Issues Not Completed — View in Issue Navigator

Key	Summary	Issue Type	Priority	Status	Story Points (24)
FFD-17	As an engineer, I want to be able to save & modify metadata connected to a sensor	Story	Highest	TO DO	5
FFD-36	As a researcher/user, I want to be able to easily visualize data (by creating my own dashboard)	Story	High	TO DO	13
FFD-37	As a user, I want to be able to access an API to retrieve/subscribe to data	Story	Highest	IN PROGRESS	3
FFD-87	As an engineer, I want to be able to add new data with existing formats manually	Story	Highest	TO DO	3

Figure 8.3: Screenshot from Jira of the Sprint 3 report

8.4.4 Sprint Retrospective

The retrospective was carried out in the same manner as the previous retrospective. The main positive notes were the amount of effort put in by all team members, a pleasant work environment, and good code reviews. During the meeting, a team member said that they did not feel heard and did not like the group dynamic. They felt that the other group members shut down their ideas and were not patient enough when there was something they did not understand. The rest of the group considered the concerns to be very serious, and came up with actions to prevent it from happening again, shown in Table 8.3. In addition, inaccurate estimation of Story Points was also something the group needed to take into account when estimating Story Points for the next sprint.

Table 8.3: What could have been better in Sprint 3 and their corresponding actions.

Could be better	**Action to improve**
Communication when working on overlapping tasks.	Before one starts to work on a task that overlaps or can affect other team members, send that person a message.
Interrupting each other during meetings.	Before interrupting, ask if the subject is relevant for everyone. If the person bringing it up says yes, continue the discussion. The group also needs to be more patient with each other.
More effective meetings.	Write your discussion points on the agenda before every meeting, so everyone has time to prepare.

8.4.5 Sprint Demo

As in Sprint 2, the group conducted acceptance testing with the customer representative, going through the product implementation completed throughout the sprint. The customer representative was understanding of the stories not being finished, pointing out that the group's experiences and choices regarding the product was of bigger interest than the product itself. The customer asked the group to conduct a time estimate for development of a final solution. Additionally, the representative requested in-depth documentation beyond the requirements for the report, particularly regarding the database and its connection between time series data and its metadata, in addition to user access management.

8.4.6 Sprint Changes

Due to a spike in COVID-19 cases in Trondheim [39], as well as multiple group members being in quarantine, the group decided to have meetings digitally through Microsoft Teams. Although this made collaboration more difficult, the employment of digital meetings did not overall noticeably affect the group's efficiency.

8.5 Sprint 4

The fourth sprint, named "Goldfish", started on October 19, after the retrospective, demo, and review of Sprint 3. Sprint 4 lasted until October 30, which encompasses two working weeks. The sprint goal was to *Make the product pretty and presentable and limit functionality to only the most important parts. The functionality that is there should work well.* When turned into achievable goals, the group needed to develop the main functionality regarding dashboards, finish the remaining work regarding adding and modifying metadata, and the ability to subscribe to a data flow through an API. This sprint also involves finishing the visual prototype and modify the product, so the styling matches the visual prototype.

8.5.1 Sprint Planning Meeting

The sprint planning meeting took place on October 19 and the group followed the same procedure as in the previous sprints. Sprint 4 was originally planned to be a full coding sprint, but due to the additional documentation requested by the customer representative, the group decided to allocate a significant portion of the time to write the report in this sprint.

8.5.2 Sprint Backlog

Seeing as Sprint 4 was the last coding sprint, it consisted of user stories of higher priority. The goal for this sprint was to make sure the implemented functionality works well. Therefore, the group decided to focus on finishing the stories that were not completed in the last sprint. Due to poor Story Point estimations from earlier sprints, the group decided to use Planning Poker to re-estimate the Story Points for the remaining work of these stories. These estimations and the stories included in Sprint 4 are shown in Figure 8.4. To ensure that the group had time to finish all the stories included in this sprint, as well as writing the report, the group decided against including more than 30 Story Points in this sprint.

8.5.3 Sprint Review

The sprint review was carried out in the same manner as in the previous sprint. An overview of the issues addressed in the sprint is shown in Figure 8.4. As shown in this figure, the group completed all the stories planned in this sprint. Burndown charts for Story Points and issues completed throughout the sprint is located in appendix D.5.4.

Completed Issues View in Issue Navigator

Key	Summary	Issue Type	Priority	Status	Story Points (27.5)
FFD-17	As an engineer, I want to be able to save & modify metadata connected to a sensor	Story	Highest	DONE	3
FFD-25	As a researcher/engineer, I want to be able to store large amounts of data	Story	High	DONE	0.5
FFD-36	As a researcher/user, I want to be able to easily visualize data (by creating my own dashboard)	Story	High	DONE	13
FFD-37	As a user, I want to be able to access an API to retrieve/subscribe to data	Story	Highest	DONE	8
FFD-87	As an engineer, I want to be able to add new data with existing formats manually	Story	Highest	DONE	2
FFD-188	Visual prototype 2	Task	Medium	DONE	1

Figure 8.4: Screenshot from Jira of the Sprint 4 report.

8.5.4 Sprint Retrospective

The retrospective was carried out in the same manner as previous retrospectives, however, performed digitally. The main positive notes were good estimations and completion of the sprint stories, structured meetings, and a pleasant work environment. The three points with the most votes for what could have been better are shown in Table 8.4, along with their corresponding actions. Due to the retrospective action of conducting the remaining meetings digitally, the group removed the biscuit penalty for not showing up on time.

Table 8.4: What could have been better in Sprint 4 and their corresponding actions.

Could be better	**Action to improve**
Show up on time.	Show up 10 minutes before the meeting starts. Five minutes before the meeting starts, send a message to those who have not yet appeared. Nevertheless, the meeting starts at the planned time.
Definition of done.	When writing the report, a section is not completed until another group member has reviewed and approved it.
Communication with both physical and digital attendees.	All meetings will be digital throughout the remainder of the semester.

8.5.5 Sprint Demo

The demo was planned to occur November 2, although due to the customer representative being ill, the meeting was postponed until November 4. As in the previous sprints, the group conducted acceptance testing with the customer representative, going through the product implementation completed throughout the sprint. The customer representative was pleased with the progress, and agreed on the plan of focusing on the report and external documentation for the remainder of the project.

8.5.6 Sprint Changes

Several meetings were conducted digitally, due to the worsening state of the pandemic. Although cooperation was harder, the safety of the group members is paramount. There were no noticeable adverse consequences to conducting most meetings digitally.

9. Security

The application consists of three main components: a front end client, a back end and a database, as shown in chapter 7. The client is accessed by the user through a web-browser. The back end exposes a GraphQL API over TCP/IP, communicating with a SQL database on the server. With the application exposed to the web, there is a series of security threats to address. Such common threats and vulnerabilities are identified in OWASP [40] and STRIDE [50], which list several of the most important and critical security risks to web applications. This section describes a threat assessment of the application, as well as the currently implemented security- and suggested measures for improving application security.

9.1 Threat Assessment

Every web-based application communicating over the internet is exposed to a series of threats regarding its security. Some threats are more serious than others, but as with any system, all security threats should be considered. Most of the security threats listed are known from OWASP [40] and STRIDE [50]. Section 9.3 lists some ways of counteracting these threats.

9.1.1 DoS

The impact of a denial of service (DoS) attack is one threat the application faces. Malicious attackers could flood the application with too many requests for the server to handle, threatening the end users' ability to access the application and its assets [9]. The application is hosted without load balancing, making it easily susceptible to such attacks [34].

9.1.2 Man in the Middle

"Man in the middle" (MITM) attacks threaten the security of the application [36]. There are several different types of MITM attacks, where all attempt to steal the users' data. Sniffing is a similar threat, attempting to access the data between the client and server [46]. Such attacks could entail an attacker attempting to access data from the connection between the client and the server, or impersonate the site to steal the users' data. In regards to the group's product, there are several exploitable points [36].

Sniffing attacks are common on web applications communicating over insecure connections and unencrypted networks [46], while MITM attacks are based on hijacking the connection between client and server, and can make an untrustworthy connection seem secure [36].

9.1.3 Root Access

A pressing security risk is the possibility of an attacker acquiring root access to the host server or database, enabling several possible attack methods. Such attacks could entail stealing the SSL certificate key, accessing the server files, and logging identity data like username and password sets, leading to several threats like data leakage and tampering. These attacks would be a critical breach of security.

9.1.4 Deprecated Frameworks

Security flaws in project dependencies could lead to a security breach, which could be caused by deprecated or broken security. Previously unknown flaws with the frameworks used in the application could cause security threats. Using old and deprecated frameworks might lead to attackers utilizing errors that are resolved in newer generations of the software. As a result, regular security updates of the frameworks and technologies are necessary.

9.1.5 Injections

There are several types of injection threats facing web-based applications. In this project, JavaScript and SQL injections are most relevant. JavaScript injections might occur from uploading malicious JavaScript snippets as text. The text is rendered in the DOM of the web client, executing malicious code snippets. Such an injection attack would lead to running unauthorized JavaScript code in the client, also known as an XSS threat [40].

SQL injections utilize flaws in the back end, attempting to execute unauthorized SQL queries. A SQL injection could enable an attacker to access unauthorized data, or to alter the data stored in the system, leading to a *data leak* of sensitive information such as login credentials, personal information.

9.1.6 Azure

The current application authenticates the user using Azure authentication. If the Azure service gets compromised, it would also leave the application exposed.

9.2 Implemented Security Measures

9.2.1 Server and database access

The current application supports security measures to prevent access from unknown or malicious attackers, one of these being the authorization of accessors of the server and database. The server is open for SSH connections, requiring a username and password for connecting. The same access restrictions are also used for the database, requiring a username and password [48]. The result of this is thus not allowing for direct connections to the server or database without valid credentials. An alternative to this would be to disable SSH login with username and password, and only allow for connections for clients with valid SSH-keys.

9.2.2 SSL

The application is secured with SSL encryption serving the site solely over HTTPS. Requiring a connection through HTTPS and SSL encryption mitigates the threat of leaking data through observing activity on unencrypted networks (Sniffing). SSL encryption is also required in the client application for user authentication with Azure.

9.2.3 Azure

Azure Active Directory is used for authenticating users. The authentication is conducted through the front end client, resulting in a user cookie and a JavaScript Web Token (JWT). The JWT is used as a bearer token when accessing the GraphQL API, for verifying and identifying the user. The token usage restricts unauthorized users from access-

ing secured data or endpoints through the API. Authenticating users through Azure is more secure and reliable than implementing authentication as part of the application.

9.3 Suggested Security Measures

9.3.1 VPN

One way of securing the application is by securing it behind a private network. For an attacker to access the application, they would first have to breach the secure network. Securing the application within a private network would add an additional layer of security, and is used in the customers current solution. To access the application from outside the private network the user would need to connect to the private network through a virtual private network (VPN) [58].

Although using a secure private network is a great way of securing the application, it is not always a viable option. The application should support other security measures as well. The application may have to be accessible for use cases outside the private network, or for securing the application in the case of a network breach. Therefore, other security measures should also be implemented to address the threats to the application.

9.3.2 SSL

Encrypting the data sent between the client and the server limits an attacker's ability to observe the data being sent, and is a direct counter measure to sniffing attacks. If the application is exposed to the internet, both the client and the server must be served over HTTPS. HTTPS encrypts the connection between the client and the server, making it more difficult for an attacker to access the data [49]. The connection should not be served over HTTP, and should force a strong SSL encryption. For SSL to be a viable security measure, the certificate key must not be available to attackers.

9.3.3 Injections

Injection threats are common threats facing web applications [40]. There are several types of injections, but as mentioned in section 9.1, JavaScript injections, and SQL injections are the most relevant for this project. There are several ways of addressing these problems.

SQL injections are caused by the application design, and measures must be taken in the design process of the application. In situations where SQL commands are created with string concatenations and mutations, SQL injections can be difficult to counteract. A solution to such problems is to use other frameworks for running SQL queries. In the case of .Net, the entity framework and LINQ framework counteracts this problem by denying traditional SQL injection attacks. Using the entity framework, SQL queries are not created by manipulating a string as used to be the norm, but through the use of SQL parameters [12].

To mitigate JavaScript injection threats, new code should not be allowed to be loaded into the DOM, which means invalidating JavaScript *script* tags loaded as data on the client. One way of invalidating such tags is through rendering these tags as invalid tags, making the code unable to execute. Another way of mitigating JavaScript injections is by not allowing data to be loaded and executed from untrustworthy sources. Such simple design steps would reduce the risk of such injection attacks.

9.3.4 Exposed Configuration Values for Authentication

The Azure configurations are currently stored as clear-text in the app-settings in the code. Leakage of these values could have implications on the security delivered by Azure. For a production system, these values should be stored safely and added when compiling, as with the use of continuous delivery services.

9.3.5 Project Dependencies

Security issues can be caused by flaws in the technologies and frameworks used to develop the application. Over time, the security of the project dependencies might be compromised, causing the application to be exposed to security threats. Newer versions of the dependencies usually improves upon the security and might solve some of the flaws present in older versions. Therefore, upgrading the project dependencies regularly helps to secure the application from threats caused by vulnerable dependency versions.

9.3.6 Load Balancing

The threat of a DoS attack can be mitigated by the use of load balancing. Load balancing means that the requests to the application are distributed to several different servers, reducing downtime and improving scalability [34]. Distributing the requests made to the application would make the application more resilient, as it scales more efficiently.

10. Testing

Testing is of great importance in modern software development, as it can hugely improve the overall experience and quality of the product. The process of testing is to detect faults and defects in the product before release. A well-tested product is often more reliable and gives an assurance that the product works as intended [47, p. 227]. Software testing ensures the quality of the implemented functionality and requirements in the project. In section 10.1, the implemented test strategy will be described, and each test type will be described in the following chapters.

10.1 Test Plan

A good test plan is important for specifying how different aspects of the application should be tested, as well as how to organize and structure the testing. This can help get a good overview of which areas in the product is problematic and avoid writing duplicate tests [47, p. 231]. A test plan helps to discover undisclosed flaws, as it diversifies the testing of the product. This can then be fixed before release, and help increase the overall quality of the product and mitigate errors.

Table 10.1: Test plan

Test type	**Description**	**Test subjects**	**When**
Unit testing	Isolated tests for individual parts of the code base.	Project	Continuous
Integration testing	Testing method where individual parts are combined and tested as a group.	Project	Continuous
End-to-end testing	Tests to check if the application as a whole is performing as expected.	Project	After development
Manual testing	A manual form of testing.	Project	Continuous
Usability testing	Testing of the applications usability.	Users	After the design is finished
Acceptance testing	Testing by the customer to see if the implemented system performs to the customers expectations.	Customer	In sprint meetings
Functional requirement testing	Testing to validate the software system against the functional requirements specification.	Project	Sprint demonstration
Quality requirement testing	Testing to verify that the product will perform as expected, as specified by the customer.	Project	Sprint demonstration

10.2 Unit Testing

Unit tests are isolated tests for testing individual parts of the codebase. The goal of unit testing is to assure that the individual parts of the program are working correctly. A unit test is the testing of the smallest software components, which can not be broken down further [47, p. 232]. The benefits of unit testing are easier execution and analysis due to the small test scope. The defects revealed by a unit test are also easier to locate and fix since they are dependent on only a small part of code.

As the project is a proof-of-concept, it was decided early on to prioritize testing primarily of its most fundamental parts. Faulty logic in fundamental methods can be difficult to discover and it may affect other aspects of the application. It was concluded that the most important part of the back end was the logic of parsing data to the correct format. If the parsed

data is in the wrong format, it may have fatal consequences later on and be difficult to correct. As this functionality may also be used for the final solution, it was prioritized. The test coverage for the parser can be seen in Figure 10.1. The main focus for unit testing on the front end was the reducers for the Redux store. The Redux store handles the global state in the client. Errors in this logic can cause abnormal behavior, such as dashboards not updating or graphs showing the wrong data.

Name	Covered	Uncovered	Coverable	Total	Line coverage	Covered	Total	Branch coverage
parser	**86**	**123**	**209**	**422**	**41.1%**	**25**	**56**	**44.6%**
parser.Config.DatabaseConfig	0	1	1	17	0%	0	0	
parser.Config.ParserConfig	11	0	11	32	100%	0	0	
parser.ParserFilePath	37	9	46	88	80.4%	11	14	78.5%
parser.ParserString	38	16	54	98	70.3%	14	22	63.6%
parser.PrepareWritingDataToDB	0	42	42	69	0%	0	16	0%
parser.Program	0	16	16	51	0%	0	0	
src.Database.WriteToDB	0	39	39	67	0%	0	4	0%

Figure 10.1: The test coverage for the parser class, responsible for parsing data files containing sensors data and writing it to the database.

10.2.1 Test Frameworks

For unit testing the back end, xUnit was employed. xUnit is a free, open source, and community-focused unit testing tool for the .NET Framework [60]. The group agreed to follow Microsoft's "Unit testing best practices with .NET Core and .NET Standard" [57] guidelines, to reinforce a consistent test style.

On the front end, the testing framework Jest was employed. Jest is a JavaScript testing library with a focus on simplicity. It was a natural choice as some of the members had prior experience with the library.

10.3 Integration Testing

Integration testing is a method in which individual components are combined and tested as a group [47, p. 48]. A component may work isolated but have issues when interacting with others. Components can be developed by different teams, which creates a necessity to have integration tests to validate their compatibility. Bugs found in integration testing are often easier to fix than at a later stage in the testing pipeline, for instance, during end-to-end testing.

The suitable parts on which to use integration testing are functionality that uses several components, such as the API layer. Due to the short time and the nature of this project, the integration tests were of low priority. Therefore, only a few integration tests were created. This includes integration tests for several queries and mutations against the API, such as time series queries and sensor queries. A new dedicated database was set up to ensure concise behavior for the integration tests.

10.4 End-to-End Testing

End-to-end testing is a test method that evaluates system integrity and flows in end user scenarios. The test method can catch errors that are not captured in the other tests, for instance, if the server is down. An end-to-end test simulates user interaction with the application, which helps to detect issues that are otherwise difficult to test, such as UI-issues and edge-case defects [47, p. 241].

Initially, the group planned to execute end-to-end testing using Cypress [6], which is a JavaScript end-to-end testing library. Cypress works by simulating an end user and can mock user interaction with the application, such as navigation

and clicking. A few end-to-end tests were created early on, but after introducing external authentication services with Azure, these tests failed to execute. The login to the site uses two-factor authentication, which was difficult to pass using Cypress. Because of this, increased focus was allocated to manual testing in an attempt to mitigate the effects of none end-to-end tests.

10.5 Manual Testing

Manual testing was a big part of the group's testing strategy and consisted of mimicking the behavior of a potential user. Manual testing is a testing method where you manually inspect the application in search of irregularities. The advantages of manual testing are that it can simulate a real user experience and handle difficult use cases better than automated testing [47, p. 247]. Manual testing can prove to be repetitive in the long run, with a high resource cost, but due to the project scope, increased focus on manual testing proved valuable.

Each group member was responsible for conducting a manual test when adding new functionality to the application. Manual testing consisted of running the application and inspecting the added functionality, as well as the whole application. The testing was an important step to ensure that the added functionality was not bringing defects to the product.

10.6 Usability Testing

Due to the project being a proof-of-concept, the scope of the usability testing was reduced. The scope of the testing was to investigate the feasibility of the product's features getting a high System Usability Scale score [2] in the final solution, rather than performing a full usability test. This entails a single lightweight and streamlined usability test iteration, using a design prototype, which focuses on the further implementational possibilities to attain better usability. The design prototype used is documented in appendix A.3.2.

10.6.1 Subject Profiles

Due to the circumstances with the ongoing COVID-19 pandemic, the group was forced to make compromises on the usability test setup. It was not possible to match the usability testing subjects to the end users. The end users would mainly consist of researchers and engineers from SINTEF, in addition to customers of SINTEF, with the majority of users ranging from 30-60 years [54]. The best available test subjects were university students with varying computer knowledge. Due to the aforementioned pandemic, the group could not safely perform the test on subjects within the appropriate age segment and field of knowledge. The group was not able to perform tests in the application's natural circumstances, and it was therefore performed in a lab environment.

10.6.2 Testing Results and Remarks

Through the testing, the subjects were prompted on a range of tasks, shown in Table 10.2, which were performed on the design prototype documented in appendix A.3.2. These tasks were selected to span the core features of the current solution.

Some points of improvement were found in the design prototype, where most of these were shortcomings in the explanations of segment-specific terminology. These problems are not expected to be replicated when the project is tested on the intended end users. Another issue was the ambiguous wording of the buttons and site sections, leading to a disconnect

Table 10.2: Tasks performed by the test subjects in usability testing, based on the general roles of the system.

User role	**Description**
Customer	Log in and locate profile information.
Customer	Locate graphs pertaining to a dashboard.
Customer	Navigate to the dashboard overview.
Customer	Log out.
Researcher & Engineer	Log in and create a new dashboard.
Researcher & Engineer	Add a cell to the dashboard with data.
Researcher & Engineer	Find metadata for the created cell.
Researcher & Engineer	Share the dashboard with another user.
Researcher & Engineer	Edit a graph.
Researcher & Engineer	Expand a cell to get a better view of it.
Researcher & Engineer	Upload new data to the system.
Researcher & Engineer	Edit the metadata for a sensor and save it to the system.
Researcher & Engineer	Edit which users are members in a user group.
Researcher & Engineer	Create a new group.
Researcher & Engineer	Find out which user have access to a particular dashboard.
Researcher & Engineer	Log out.

between the user's mental model and the system image. The last major issue was a population issue, where the small amount of data included in the prototype led to issues where some pages were underpopulated. This confused some of the test subjects and led to a larger usage of time on the tasks than what was expected.

10.6.3 Adaptations and Reflections

The issues rooted in the mismatch in the capabilities and segment-specific vocabulary, between the intended end user and the test subjects, were in some instances remedied by changing the wording to be more widely understood. In instances where the wording is more important, the vocabulary was kept. The risk of these issues was assessed to be low when used by the targeted end users, who are familiar with the vocabulary. The issue of ambiguous wording was taken into account, and the associated buttons and site sections were altered to convey the functions and usages. The population issue is of little concern due to the considerable amounts of data which is to be stored in the final solution.

10.7 Acceptance Testing

The purpose of the acceptance testing was to determine if the desired requirements were met [30, p. 372]. Acceptance testing is a test conducted by the customer representative to see if the implemented system is acceptable regarding the expectations. In the case of this project, it was critical to verify that the user-stories were addressed in the implemented system.

At the meetings with the customer, usually at the end of each sprint, an acceptance test was conducted. This was done to ensure that the progression of the product was satisfactory to the customer and that the user-stories were correctly interpreted. These tests were conducted by going through the completed user stories implemented in the product since

the last conducted test. The results of these stories were presented to the customer for verification. Conducting the tests at such close intervals enabled the group to quickly adapt to changes caused by miscommunication or misinterpretation of the tasks and story-points. These tests provided valuable insight into the customer's vision for the application, and helped us with making alterations to the application at an early stage before it was too late.

10.8 Functional Testing

To validate that the functional requirements in section 6.1 are implemented, functional tests have been conducted. The purpose of functional testing is to assure that each function in the software application works as expected. This can be tested by providing input to the desired methods and verifying the output [16]. The results of these tests can be seen in appendix C.1. The IDs of the functional requirements refer to the IDs of the requirements in the requirement specification in section 6.1. Due to the project being a proof-of-concept, not all of the functional requirements were implemented. Only the implemented requirements were tested.

10.9 Quality Attribute Scenarios Testing

Quality attribute testing consists of testing the non-functional requirements scenarios of a software. These scenarios for the project are defined in section 6.2. The goal of the testing is to measure how a system operates, rather than specific behaviours of the system. The results of these tests can be seen in appendix C.2.

11. Evaluation

To be able to improve future project work, the group needs to evaluate to find aspects to improve. The course staff also relies on good evaluations from the students to be able to improve the course for the next year.

11.1 Internal Processes and Results

In general, the group has worked well together, and the group dynamic has been positive. The majority of the group did not know each other from before the project, resulting in the typical group development stages forming, storming, norming, and performing [55]. During these stages, the group members achieved better relationships, enhancing the work ethics, and making every group member want the other group members to succeed. Every group member contributed to the best of their abilities. There was a common understanding that the group members had different qualities, and members with greater knowledge in some topics were happy to help other group members, leading to a culture where the group's goals were more important than each group member's individual goals.

At the beginning of the project, the group had issues establishing a common interpretation of the project tasks, due to a lack of thorough group discussions. During the project, the communication improved and the group members assured that everyone had the same understanding of what the tasks included. In retrospective, the group should have discussed the task in more depth before starting to develop. Another challenge was the combination of physical and digital communication. It was difficult to have a good discussion when some group members were attending digitally, and others attended physically. The group discovered this challenge quickly and decided to conduct the remaining meetings digitally to enable full participation in discussions.

During the retrospective for sprint 3, a group member brought up an issue regarding the internal processes; the group member shared the feeling of often being interrupted and constantly having his suggestions downvoted in discussions. The group member showed great courage by sharing these feelings with the rest of the group. This incident was discussed thoroughly, and the group agreed upon an action for the next sprint to avoid this from happening again, shown in Table 8.3.

The group reached the project goals, which were to design and implement a prototype for the customer, as well as improve the ability to work in groups. During the project, the group members have learned how to implement a system for a customer outside NTNU. The group had trouble estimating Story Points, however, the group grew cognizant of the issue. As discussed in subsubsection 5.2.1, estimation is challenging, but over the course of the project, the group has improved on this ability. Lastly, the group has learned about group dynamics and its importance. The group members have learned that effective communication is a key characteristic of a well-functioning group.

11.2 Customer and Project Task

During the first meeting with the customer representative, the group was presented with the customer's wanted solution. This was a description of a complete solution, with no specific requirements. As discussed in chapter 6, the group struggled to understand what to develop as the presentation was quite vague. However, when prompted on the specifics of the project, the customer was very cooperative and created a requirement specification for the complete solution. The requirement specification was presented to the group after the pre-study was completed. The functional requirements

generated from the customer's requirement specification are described in Table 6.1. The specification of the requirements provided the group members with a clearer understanding of the wanted solution.

The communication with the customer representative was regular and illuminating. The customer representative allocated time to meet with the group when it was necessary and was perceived by the group as interested in the development process. The group asked if the customer representative could join the Sprint 1 retrospective meeting, however, the representative did not have time available. As the group finished their first retrospective meeting, the group decided that it was not necessary for the customer representative to be present on the retrospective meetings.

11.3 Communication with Advisors

At the beginning of the project, the group experienced communication issues with the supervisor. To handle this challenge, the group created an action of writing an email to the supervisor with a summary of the weekly supervisor meeting. The action enabled the supervisor to confirm that the group had understood them correctly, which improved the group's utilization of the supervisor's guidance. The responses from the supervisor were quick and thorough, particularly regarding the report, which enabled the group to continuously improve the report. Additionally, the supervisor ensured that the group was on track with the progress, which was communicated at the end of each supervisor meeting. To summarize, the group received the necessary supervision.

11.4 Further Work

The goal of the project was to create a proof-of-concept to investigate and demonstrate the feasibility of creating the final solution using open source technology. The group believes they have achieved the goal of creating such a proof-of-concept. However, a significant amount of work remains to complete the final solution covering the requested functionality made by the customer. The following paragraphs present the necessary work, with a rough time estimate. The estimates are the reached consensus among the group for each category. The group has based the estimates on the project experience, as well as the customer representative's description of the future developer's skill set, which was a junior consultant with 5 years experience.

The group assesses the remaining work to be contained in five main categories: authorization and access control, metadata, dashboards, alerting, sensors, and further improvement of the existing product. These are all based on user stories developed in conjunction with the customer representative in the early stages of the planning.

In authorization and access control, two necessary features are to be developed. These features are access control of dashboards and sensors, as well as establishing user group-specific and individual access to certain time series and sensors. The group has estimated this work to require 400 man-hours of development.

Search features based on metadata remains to be implemented. Furthermore, a log must be added for each sensor to allow for supplying notes of special incidents and remarks regarding the sensor. Metadata should also be connected to Landax through its API. These tasks are estimates to require 200 man-hours.

Regarding the dashboards, the group has not implemented the ability to share dashboards with other users. There should also exist functionality to download these dashboards and the corresponding graphs. The group has only implemented one basic graph type, but the end system requires more complex graph types. It would be natural to implement them

in cooperation with the implementation of more complex sensor types. Implementing new graph types should not take much time when using Highcharts, but the total estimate of the remaining workload within dashboards is 200 man-hours.

Alerts should be issued on abnormal time series data to encourage checks of the sensor's function. Alerts should also be issued on changing sensor status and disconnection of sensors, which can be implemented independently or through a third-party solution such as Grafana Alerting. A rough estimate of man-hours for these tasks would be 150 man-hours.

The current sensors implemented are rather rudimentary for the proof-of-concept product, and therefore only supports basic sensors, unless configuration files are provided. The space of integrated sensors should be expanded to include more complex sensor types such as echo sounder. Furthermore, there should be support for uploading atypical data types such as video. The ability for sensors to upload automatically remains to be completed, as the current system only supports manual upload. Engineers should be able to write drivers and connect to the system. This can be implemented using the Telegraf from the TICK stack, although it is not currently used in the system. In total, the implementation of these features would require around 250 man-hours.

The current system is a proof-of-concept prototype to demonstrate the implementational possibility of as many aspects of the final solution as possible. Therefore, some aspects of the delivered system are not of the highest possible quality. The group estimates that 100 man-hours are necessary to adequately enhance the current sub-optimal implementations.

A significant amount of work remains in order to reach the standard set for the final solution. In total for all these additions and improvements, the group has made a total estimate of 1300 man-hours.

11.5 Suggestions for Improvement

In general, the group is satisfied with the course as it is, but there is still room for improvements. The first improvement is regarding the creation of groups. The group included five students with a specialization in artificial intelligence and none with a specialization in databases. Therefore, the database-oriented nature of the task was challenging for the group. The group suggests that the creation of groups should not be entirely random in the future. A possible solution would be to let the members of the different specializations be distributed equally among the groups.

The group had the impression that the course staff did not have enough requirements towards the customers. Therefore, the group suggests that there should be a necessity that all customers should have a list of requirements for their projects, approved by the course staff, before the start of the project. A boot camp at the start of the project between the group and customer could lead to a better understanding of the wanted solution at an earlier stage of the project.

Finally, the group has some comments on the lectures. The lectures had relevant topics, but some of them could contain more specific examples concerning the projects. For example, the security lecture did not contain any specific examples regarding how to implement the security measurements. Some of the lectures were scheduled poorly, such as the lecture concerning project management, which was conducted after the groups were scheduled to finish the planning part of the project. The group believes it would have been impactful to have this lecture at an earlier stage.

Bibliography

[1] SINTEF ACE. https://www.sintef.no/en/all-laboratories/ace/. Accessed: 2020.09.07.

[2] John Brooke. Sus: a "quick and dirty'usability. *Usability evaluation in industry*, 1996.

[3] Chadana. Scrum Project Management: Pros and Cons. https://www.simplilearn.com/scrum-project-management-article, 2019.10.04. Accessed: 2020.09.02.

[4] Microsoft's C# code conventions. https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/inside-a-program/coding-conventions. Accessed: 2020.09.07.

[5] Prettier extension. https://prettier.io/. Accessed: 2020.09.07.

[6] JavaScript End to End Testing Framework. https://www.cypress.io/. Accessed: 2020.11.10.

[7] Chuparkoff Dan. A new introduction to jira & agile project management. https://www.youtube.com/watch?v=TsG3OWTDAFY, 2018.08.14. Accessed: 2020.09.04.

[8] Radigan Dan. Story points and estimation. https://www.atlassian.com/agile/project-management/estimation. Accessed: 2020.09.04.

[9] What are Denial of Service (DoS) attacks? DoS attacks explained. https://us.norton.com/internetsecurity-emerging-threats-dos-attacks-explained.html. Accessed: 2020.11.08.

[10] Panayotova Elina. What Are the Pros and Cons of Extreme Programming (XP)? https://simpleprogrammer.com/pros-cons-extreme-programming-xp/, 2018.12.24. Accessed: 2020.09.04.

[11] Eden Ella. Top 5 React Chart Libraries for 2020. https://dev.to/giteden/top-5-react-chart-libraries-for-2020-1amb, 23. oct 2019. Accessed: 2020.09.09.

[12] Entity Framework. https://entityframework.net/linq-prevent-sql-injection. Accessed: 2020.11.08.

[13] ESnet. React Timeseries Chart. https://software.es.net/react-timeseries-charts/#/. Accessed: 2020.11.12.

[14] Extreme Programming. https://www.agilealliance.org/glossary/xp. Accessed: 2020.09.04.

[15] Siddiqui Farzan. The Importance and Benefits of a Mid-Sprint Review. https://dzone.com/articles/mid-sprint-review-importance-amp-benefits, 2018.10.17. Accessed: 2020.09.04.

[16] Functional testing. https://en.wikipedia.org/wiki/Functional_testing. Accessed: 2020.11.12.

[17] Grafana. https://en.wikipedia.org/wiki/Grafana, 23. oct 2020. Accessed: 2020.11.04.

[18] Graphite. The whisper database. https://graphite.readthedocs.io/en/stable/whisper.html, Jan 2015.

[19] Graphite. Faq. https://graphite.readthedocs.io/en/stable/faq.html, Apr 2017.

[20] GraphQL. https://graphql.org/. Accessed: 2020.11.11.

[21] Recharts Group. Recharts. https://recharts.org/en-US. Accessed: 2020.11.12.

[22] Kniberg Henrik. *SCRUM AND XP FROM THE TRENCHES How We Do Scrum*. C4Media, 2nd edition, 2015.

[23] Highcharts. Highcharts. https://www.highcharts.com/blog/products/highcharts/. Accessed: 2020.11.12.

[24] Highcharts. Interactive Javascipt charts for your webpage. https://www.highcharts.com/. Accessed: 2020.09.10.

[25] InfluxData. influxdata kapacitor 1.3 | monitoring push vs pull and the support of both_2020. https://www.influxdata.com/blog/monitoring-with-push-vs-pull-influxdb-adds-pull-support-with-kapacitor/, 2020.

[26] ISO 9000:2015(en) Quality management systems — Fundamentals and vocabulary. https://www.iso.org/obp/ui/#iso:std:iso:9000:ed-4:v1:en. Accessed: 2020.10.07.

[27] Kanban (development). https://en.wikipedia.org/wiki/Kanban_(development), 2. sep 2020. Accessed: 2020.09.16.

[28] Kanban Vs Scrum Benefits, Similarities, Pros and cons. https://www.yodiz.com/blog/kanban-vs-scrum-benefits-similarities-pros-and-cons/, 2016.08.01. Accessed: 2020.09.16.

[29] Philippe Kruchten. Architectural Blueprints—The "4+1" View Model of Software Architecture. https://www.cs.ubc.ca/~gregor/teaching/papers/4+1view-architecture.pdf, 1995. Accessed: 2020.08.22.

[30] R. Kazman L. Bass, P. Clements. *Software Architecture in Practice*. Pearson Education, Inc., 3rd edition, 2015.

[31] Grafana Labs. Permissions. https://grafana.com/docs/grafana/latest/permissions/. Accessed: 2020.11.04.

[32] Grafana Labs. What is Grafana? https://grafana.com/docs/grafana/latest/getting-started/what-is-grafana/. Accessed: 2020.11.12.

[33] Stacia levy. The Roundtable Discussion. https://busyteacher.org/24153-roundtable-discussion-esl.html. Accessed: 2020.11.09.

[34] What Is Load Balancing? https://www.nginx.com/resources/glossary/load-balancing/. Accessed: 2020.11.08.

[35] Methodology. https://en.wikipedia.org/wiki/Methodology. Accessed: 2020.09.01.

[36] What is a man-in-the-middle attack? https://us.norton.com/internetsecurity-wifi-what-is-a-man-in-the-middle-attack.html. Accessed: 2020.11.08.

[37] Andersen Nina H., Anderson Martha M. K., Brandshaug Sigrid W., and et al. *Eksperter i team 2019 Håndbok for landsbyledere og læringsassistenter*. NTNU, 10 edition, 2019.

[38] Npgsql. http://www.npgsql.org/. Accessed: 2020.11.11.

[39] Tore Oksholen. Fem nye ntnu-studenter smittet av korona, Oct 2020. Accessed: 2020.11.09.

[40] OWASP Top Ten. https://owasp.org/www-project-top-ten/. Accessed: 2020.11.08.

[41] John Potter. npm trends. https://www.npmtrends.com/react-timeseries-charts-vs-recharts-vs-highcharts. Accessed: 2020.09.10.

[42] Prometheus. Pushing metrics. https://prometheus.io/docs/instrumenting/pushing/, Sep 2019.

[43] Prometheus. Frequently asked questions. https://prometheus.io/docs/introduction/faq/, Aug 2020.

[44] React AAD MSAL. https://www.npmjs.com/package/react-aad-msal, Apr 2020. Accessed: 2020.11.11.

[45] Scrum. https://www.mountaingoatsoftware.com/agile/scrum. Accessed: 2020.09.02.

[46] What is a sniffing attack? https://nordvpn.com/no/blog/sniffing-attack/. Accessed: 2020.11.08.

[47] Ian Sommerville. *Software Engineering*. Pearson, 10 edition, 2016.

[48] SSH. https://www.ssh.com/ssh/. Accessed: 2020.11.08.

[49] What is an SSL certificate? https://www.digicert.com/ssl/. Accessed: 2020.11.08.

[50] The STRIDE Threat Model. https://docs.microsoft.com/en-us/previous-versions/commerce-server/ee823878(v=cs.20)?redirectedfrom=MSDN. Accessed: 2020.11.08.

[51] TimescaleDB. Data model. https://docs.timescale.com/latest/introduction/data-model.

[52] TimescaleDB. Data retention. https://docs.timescale.com/latest/using-timescaledb/data-retention.

[53] TimescaleDB. Timescaledb overview. https://docs.timescale.com/latest/introduction.

[54] Hans Torvatn, Birgit Kløve, and Andreas D Landmark. Ansattes syn på digitalisering. *Sintef rapport*, 2017, 2017.

[55] B. Tuckman. Developmental sequence in small groups. *Psychological bulletin*, 63:384–99, 1965.

[56] UML Use Case Diagram. https://www.lucidchart.com/pages/uml-use-case-diagram. Accessed: 2020.09.10.

[57] Unit testing best practices with .NET Core and .NET Standard. https://docs.microsoft.com/en-us/dotnet/core/testing/unit-testing-best-practices. Accessed: 2020.11.02.

[58] What is a VPN. https://www.expressvpn.com/no/what-is-vpn. Accessed: 2020.11.08.

[59] The WebSocket API (WebSockets). https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API, 2020. Accessed: 2020.11.07.

[60] xUnit. https://xunit.net/. Accessed: 2020.11.02.

A. Internal and External Documentation

A.1 Installation Guide and User Guide

The installation and user guide is supplemented in the README.md page on GitHub. The README.md page can be found here `https://github.com/Sanderkk/CustomerDrivenProject/blob/dev/README.md`.

A.2 Back End Documentation

The group implemented the back end design mocked in the first sprint Figure D.10. Under development, the group realized that some architectural changes were necessary. These changes resulted in the final design, which can be seen in Figure 7.1.

The general flow of information starts with the front end calling the back end GraphQL API, retrieving or updating data in the database, and finally returning data back to the front end. An example of the flow is showed in Figure D.12. The GraphQL endpoint receives the request and input data through a POST request, delivering it to the back end logic. The request, either a query, mutation, or subscription, is handled by the back end logic, and executed against the database. A SQL query is based on the parameters of the request, and is created using mutations and concatenations of strings. The queries are executed using the npgsql library in the .NET Core framework, and the response from the database is parsed and returned to the front end. The API requests uses proprietary methods for creating these SQL query strings, before querying the database, processing the returned data, and lastly responding to the client calling the API.

A.2.1 Database Documentation

The database is designed with the notion of outsourcing authentication to a third party, in this case Azure Active Directory. Authentication through Azure requires the user to be registered with a unique email, doubling as the unique ID of the user. The unique ID attribute is used for controlling access to restricted resources, or retrieving data connected to the user.

The selection of a Timescale database over other open source database alternatives, like InfluxDB, is presented and discussed in Section 4.5.2. The explanation of the structure and core ideas behind the design of the group's database is split into two sections: Data management and Data Access management.

Data Management

The core of the data management is the "sensor" table, shown in Figure A.1. New sensors, with potentially new data formats, are also supported by the system. The issue was solved by designing the "sensor" table as an index containing an unique ID for each sensor, the name of the table where the data is is stored ("sensor_value" is used as an example in the ER-diagram, shown in Figure A.1 and in Figure A.2), and a string containing the columns of the respective table. The design enables high flexibility in adding new tables, and the ability to query the data tables without knowledge of the table names or column names.

Metadata related to a sensor is stored in the "metadata", "location", and "log" tables, while data regarding the dashboards and cells are stored in the "dashboard" and "cell" tables. A overview of the tables can be found in Figure A.2. More details on how metadata is stored can be found in the API section A.2.2.

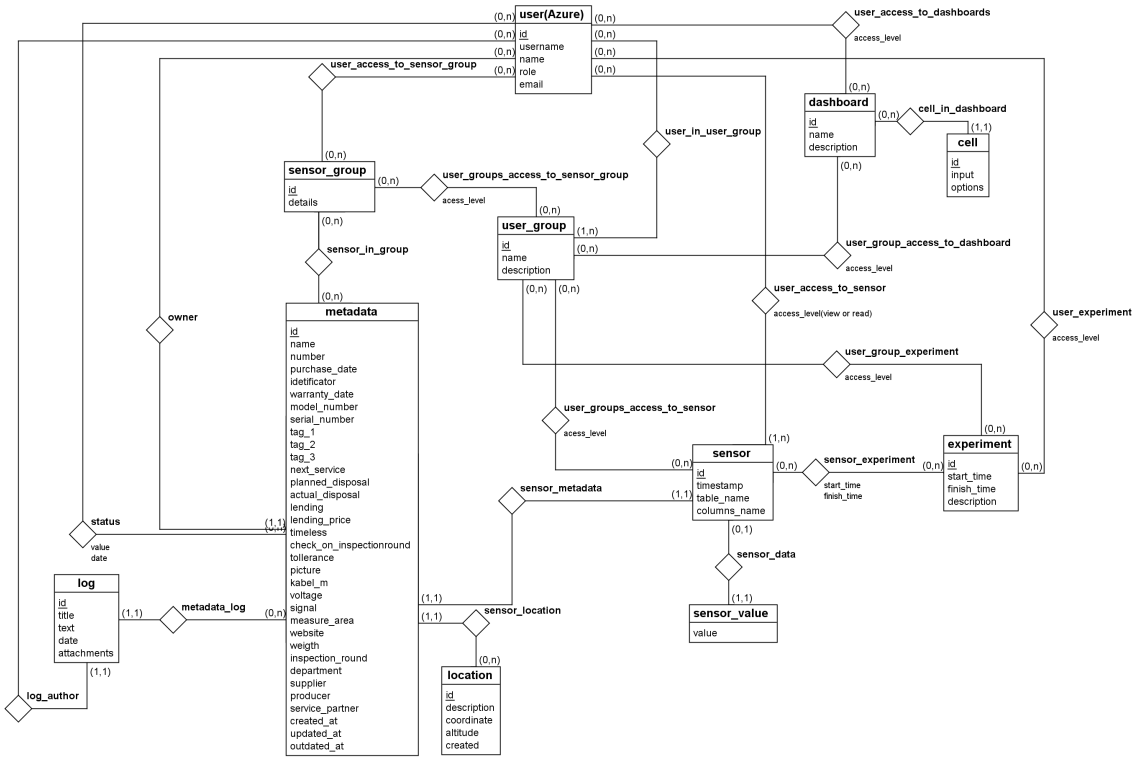


Figure A.1: ER-Diagram of the database.

Data Access Management

The possibility to define access to a sensor by classifying them in groups or on a singular basis, was a requirement from the customer. Additionally, the possibility to share a dashboard or to limit the data that can be accessed by a user or a group, was another requirement. In the database implementation, there are therefore 10 tables for regulating access control across users, sensors, and dashboards.

Users can be organized in groups and attain access to sensors, groups of sensors, dashboards, and experiments. Users access can be limited through access levels , specified in the "access_level" field. The tables for such functionality is shown in Figure A.3.

The "Experiment" table is a way to group sensors over a timespan, so that giving access to the experiment will also give access to data of the related sensors in that timespan.

Database Controller

Due to the magnitude of this project, the group decided to pursue multiple service repositories instead of a single controller design. The service repositories are located under the database folder, organized by a category name that represents the domain of each controller. Each repository contains a query builder for creating SQL queries, as well as controllers for executing the queries and processing the results.

Tables for main functionality:

sensor(id, table_name, table_column)
id PK not null,
owner_id FK referring “user(azure)” table and not null.

sensor_value(sensor_id, timestamp, value)
timestamp PK not null,
sensor_id FK referring “sensor” table and not null.

metadata(id, sensor_id, company, service_partner, department, location_id, name,owner_id, number,..., weight, status)
id PK not null,
sensor_id FK referring “sensor” table and not null,
location_id FK referring “location” table and not null.

location(id, location, altitude, description, created)
id PK not null.

log(id, metadata_id, author_id, title, text, date, attachment)
id PK not null,
metadata_id FK referring “metadata” table and not null,
author_id FK referring “user(azure)” table and not null.

dashboard(id, name, description)
id PK not null.

cell(id, dashboard_id, input, options)
dashboard_id FK referring “dashboard” table and not null.

experiment(id, start_time, finish_time, description)
id PK not null.

sensor_experiment(experiment_id, sensor_id, start_time, finish_time)
experiment_id FK referring “experiment” table and not null,
sensor_id FK referring “sensor” table and not null.

Figure A.2: Database tables for data management.

Tables for regulating groups and access level to resources:

user_groups(id, name)
id PK not null.

user_in_user_group(user_group_id, user_id)
user_group_id FK referring “user_group” table and not null.

sensor_group(id, details)
id PK not null.

sensor_in_sensor_group(sensor_id, sensor_group_id)
sensor_id FK referring “sensor” table and not null,
sensor_group_id FK referring “sensor_group” table and not null.

user_group_access_to_sensor(user_group_id, sensor_id, access_level)
user_group_id FK referring “sensor” table and not null,
sensor_id FK referring “sensor” table and not null.

user_group_access_to_sensor_group(user_group_id, sensor_group_id, access_level)
user_group_id FK referring “user_group” table and not null,
sensor_group_id FK referring “sensor_group” table and not null.

user_access_to_sensor(sensor_id, user_id, access_level)
sensor_id FK referring “sensor” table and not null,
user_id FK referring “user(azure)” table and not null.

user_access_to_sensor_group(sensor_group_id, user_id, access_level)
sensor_group_id FK referring “sensor_group” table and not null,
user_id FK referring “user(azure)” table and not null.

user_access_to_dashboard(user_id, dashboard_id, access_level)
user_id FK referring “user(azure)” table and not null,
dashboard_id FK referring “dashboard” table and not null.

user_group_access_to_dashboard(group_id, dashboard_id, access_level)
group_id FK referring “group(azure)” table and not null,
dashboard_id FK referring “dashboard” table and not null.

user_experiment(experiment_id, user_id, access_level)
experiment_id FK referring “experiment” table and not null,
user_id FK referring “user(azure)” table and not null.

user_group_experiment(experiment_id, user_group_id, access_level)
experiment_id FK referring “experiment” table and not null,
user_group_id FK referring “user_group” table and not null.

Figure A.3: Database tables for data access management.

A.2.2 API Documentation

GraphQL

GraphQL is a query language and a runtime for fulfilling queries with existing data [20]. Documentation of the GraphQL language can be found on the official GraphQL pages: `https://graphql.org/learn/`. The GraphQL query language permits running queries in a modular fashion. The query language allows for requesting multiple data flows at a time, as well as defining the data that should be returned from these requests, through modular requests. Request types ensures that clients only request valid fields, returning clear and helpful feedback when errors occur. Types can be used to avoid needing to implement parsing code on the client-side. In addition, GraphQL is self-documenting due to the aforementioned types and server-side implementation.

Hot Chocolate

Hot chocolate is a package for .NET, enabling the development of a GraphQL layer in the back end API. The package supports several different code styles for implementing the GraphQL API layer. An in-depth explanation of the framework can be found on its web page: `https://chillicream.com/docs/hotchocolate/v10`.

Hot Chocolate allows the use of C# syntax to define queries, mutations, subscriptions, and their respective types. Queries are used for querying data, mutations are requests used for altering data, and subscriptions is the GraphQL implemented publish-subscribe system using WebSockets for creating connections. The Hot Chocolate package also supports authentication and authorization through the use of the ASP.Net Identity service, which is used for authenticating the users.

A.2.3 Npgsql

Npgsql is a .NET library enabling applications to connect to Postgres and Timescale databases. Npgsql offers a data provider through the ADO.NET API, and also provides optional providers for communicating with the database over either the Entity Framework or the Entity Framework Core [38]. The current solution utilizes the Npgsql provider compatible with the ADO.NET API. The documentation for Npgsql can be found on their official website: `http://www.npgsql.org/doc/index.html`.

A.2.4 ASP.NET Core

The "Microsoft.AspNetCore" package is used in the API for building and serving the GraphQL API, as well as its interactive playground. The group uses a Hot Chocolate integration with ASP.NET to coordinate the authentication through Azure Active Directory. The documentation for ASP.NET can be found in the official Microsoft ASP.NET documentation: `https://docs.microsoft.com/en-us/aspnet/core/?view=aspnetcore-5.0`.

Metadata Management

The metadata connected to a sensor is stored in a separate Metadata table with the Sensor id of the sensor. The prototype was intended to support transparency of data, meaning that when new metadata is added to a sensor, the old metadata is not deleted. To achieve this, timestamps of metadata validity are added to the metadata to keep track of the changes. If the metadata and sensor already exist in the table, the new values are added to the metadata table with a new timestamp. In the case where no previous metadata exists for the given sensor, the back end creates a new row in the “sensor” table, shown in Figure A.2. The back end proceeds by adding a new row into the “metadata” table with the “sensorID”, as well as the unique “number”, equal to the stored value in the Landax system.

The back end then checks whether the new metadata has a specified location. In this section, position refers to 3 fields called “coordinate”, “altitude” and “location_description”. If the new position is equal to that of previous metadata, the “locationID” value is used in the new row. However, if the position in the old metadata is not specified or the new metadata has a different position, the back end creates a new instance in the “location” table and fills the new row with the new “locationID”. If no position is specified in the new metadata, but a position exists in the previous metadata, the back end fills all position fields of the new metadata with Null values.

Type objects

Cell

Represent a cell in a dashboard.
FIELDS
cellId: Int!
dashboardId: Int!
input: CellGraphData
options: CellOptions

CellOptions

Containing options data of a cell.
FIELDS
lYAxis: String
rYAxis: String
title: String

Dashboard

Representing a dashboard.
FIELDS
dashboardId: Int!
description: String
name: String

SensorType

Represents a group of sensors of the same type.
FIELDS
sensorColumns: [String]
sensorIds: [Int!]
sensorNumbers: [String]
sensorTypeName: String

GenericObject

Encapsulate information of each requested sensor, which tables are located and the data periods they span.
FIELDS
data: [DataObjectOfDecimal]
endDate: DateTime!
startDate: DateTime!
table: String
time: [DateTime!]

DataObjectOfDecimal

Encapsulated data of a requested sensor, interval between points, start time of the time series and a name to identify it by.
FIELDS
data: [Decimal!]
interval: Long!
name: String
startTime: Long!

GenericTimeType

A generic object used to define a time frame.
FIELDS
from: DateTime!
to: DateTime

Figure A.4: API type objects.

Type objects

MetadataType

Object containing metadata of a sensor.

FIELDS

actualDisposal: DateTime
altitude: Int
cableLength: Float
checkOnInspectionRound: Boolean
company: String
coordinate: String
createdAt: DateTime!
department: String
identificator: String
inspectionRound: String
lending: Boolean
lendingPrice: Float
locationDescription: String
locationID: Int
measureArea: String
metadataID: Int!
modelNumber: String
name: String
nextService: DateTime
number: String!
outdatedFrom: DateTime
ownerID: String
picture: String
plannedDisposal: DateTime
purchaseDate: DateTime
sensorID: Int!
serialNumber: String
servicePartner: String
signal: String
tag1: String
tag2: String
tag3: String
timeless: Boolean
tollerance: Boolean
updatedAt: DateTime!
voltage: String
warrantyDate: DateTime
website: String

Figure A.5: Metadata type object.

Input type objects

TimeSeriesPeriodeInput

Object used for defining a time period.

FIELDS

sensorId: int!
tableName: String!

TimeSeriesRequestInput

An object containing parameters of a request for data of a given sensor in a given time frame.

FIELDS

from: DateTime!
sensors: [Int!]
specifiedTimePeriod: Boolean!
ticksBackwards: Long
to: DateTime!

MetadataInput

It encapsules all parameters of a new metadata. It inherits from the same model as Metadata type but lacks system generated and maintained fields like createdAt, updatedAt and outdatedAt. For field description, see Metadata Type object.

CellInput

Cell data and options to add in the database.

cellId: Int
dashboardId: Int!
input: CellGraphInput!
options: CellOptionsInput!

CellGraphInput

Information regarding a new cell like which sensors it contains and the time frame if specified.

from: String!
sensors: [Int!]
specifiedTimePeriod: Boolean!
to: DateTime!

CellOptionsInput

Options regarding a new cell.

lYAxis: String
rYAxis: String
title: String

DashboardInput

A new dashboard to add/modify into the database.

acessLevel: String
dashboardID: Int
description: String
name: String

UploadDataInput

Object containing data and configuration encoded in Base64 string.

encodedConfig: String!
encodedData: String!

Figure A.6: API input type objects.

Root query

FIELDS

cell(cellId: Int!, dashboardId: Int!): Cell
: Returns a specific cell of a specific dashboard the current user has access to. It requires a unique cellId and a unique dashboardId. The resulting Cell is encapsulated in a Cell type object.

cells(dashboardId: Int!): [Cell]
: Returns all cells related to a dashboard the current user has access to. It requires the unique dashboardID and it returns a list of Cell objects.

dashboard(dashboardId: Int!): Dashboard
: Returns a single private dashboard owned by the logged user. It requires the specific dashboard id and it returns a Dashboard object.

dashboards: [Dashboard]
: Returns a list of private dashboards owned by the logged user, encapsulated as a Dashboard object.

metadata(onlyLast: Boolean, sensorID: Int, sensorNumber: String): [MetadataType]
: Highly configurable endpoint for retrieving metadata. It returns all the existing metadata in the database if no arguments are passed.
If “onlyLast” is set to True, it returns the most recent metadata of each sensor. The “sensorID” and “sensorNumber” arguments are used to narrow the search. If “onlyLast” is set to True while either of the sensor filters are used, the last metadata of the wanted sensor is returned.

sensors: [SensorType]
: Returns a list of SensorType objects, each containing a list of IDs and Numbers of all sensors of the same type.

sharedUserDashboards: [Dashboard]
: Returns a list of dashboards shared with the current logged user, each dashboard encapsulated in a Dashboard type object.

timeSeries(input: TimeSeriesRequestInput!): GenericObject
: TimeSeriesRequestInput object is required. Returns all data of the wanted sensors in the given time period.

timeSeriesPeriode(input: TimeSeriesPeriodeInput): GenericTimeType
: It accepts an TimeSeriesPeriodeInput object containing sensorID and number and returns a GenericTimeType object containing the time frame sensor data exists in the database. This endpoint is not being utilized in the front end in this proof of concept.

Figure A.7: API root queries.

Mutations

FIELDS

addMetadata(newMetadata: MetadataInput!): MetadataType
: It requires a MetadataInput object as input. It adds new metadata to the system and stamps previous metadata as outdated if it exists. It returns the added metadata as a MetadataType object.

deleteCell(cellId: Int!, dashboardId: Int!): Boolean!
: Deletes a specific cell in the system.

deleteDashboard(dashboardId: Int!): Boolean!
: Deletes a specific dashboard in the system and all the connected cells.

updateCell(input: CellInput): Int!
: It requires a CellInput type object as input. If the CellInput object contains a valid cellID, it updates the relative cell with the new information. Else it adds a new cell in the system with the new information, connects it to the relative dashboard and returns the new cellId.

updateDashboard(input: DashboardInput): Int!
: It requires a DashboardInput type as input. If the DashboardInput object contains a valid dashboardID, it updates the relative dashboard with the new information. Else it adds a new dashboard in the system with the new information and returns the new dashboardId.

uploadData(input: UploadDataInput!): Boolean!
: Endpoint for uploading new sensor data to the system. It requires an UploadDataInput object as input containing both data and configuration as a Base64 encoded string. Returns true if the data is valid, if data already exists in the database, it throws an error. This mutation also triggers an event causing the “onData” subscription to send data to the subscribed clients.

Subscriptions

FIELDS

onData(sensorId: Int!): GenericObject
: Start a subscription on a given sensor: when new data regarding the interested sensor is stored into the database, the API server encapsulates it in a GenericObject type and sends it to the subscribed client.

Figure A.8: API mutations and subscriptions.

A.2.5 Parser Service

Data is passed from the front end to the parser service via the uploadData mutation described in the API documentation shown in Figure A.8. To parse the data file, the parser needs a configuration file that includes information about how the data file is formatted, e.g. header size and the corresponding data types. A data file may contain data from different sensors, therefore, sensors correspondence to columns must be defined in the configuration file. The uploadData mutation requires both the data- and configuration file encoded as Base64Strings, which are decoded on the server side.

The parser creates a record data structure with the column names and their relative data. A submodule of the parser service "PrepareWritingDataToDB" is then called. The module processes all the column names, replacing characters that are not supported in the database. Due to the project being a prototype with limited access to different file formats, assumptions have been made. An assumption is that the data files are on the CSV format, with a unique timestamp column and separate sensor columns. The column name is used as the sensor name, and creates a new table in the database if the sensor does not already exist.

The "PrepareWritingDataToDB" module generates a SQL query for writing data in batches to the aforementioned tables. If the tables do not exist, the module creates queries for creating the tables and converting them to a hypertable. Documentation on how hypertable conversion is performed in TimescaleDB is found in the official TimescaleDB documentation: `https://docs.timescale.com/latest/getting-started/creating-hypertables`. The parser calls the database controller "WriteToDB.WriteData" which opens the connection with the database, and executes the previously generated SQL queries for writing data.

Note that the group's implementation of the parser contradicts the shared-data pattern from the architectural design shown in chapter 7. The group does not recommend implementing the parser in this way, regarding database and parser connection, but since this is a proof-of-concept, there was not allocated extra time for such best practice improvements. One can utilize the shared-data pattern by prohibiting any direct communication between the parser and the database.

A.2.6 Publish-Subscribe Pattern

As the project is aimed to be a proof-of-concept, the main goal is to find whether an implementation is feasible, often through a functional prototype evidencing the implementational possibility of a concept. In the case of the possibility for external entities to subscribe to certain data flows from specific sensors, the explored possibilities were a proprietary Publish-Subscribe(PubSub) architectural pattern, a Publish-Subscribe architectural pattern implemented through the GraphQL-subscriptions module, and a Data Distribution Service(DDS).

Publish-Subscribe Selection

The publishers of the data flow, the sensors, have no need for information regarding the entities which subscribe to the data flows. Likewise, the subscribers express interest in the data flows necessary and have no further need for information regarding the origin of the data. This loose coupling is a primary component of the publish-subscribe messaging pattern, where publishers and subscribers have no direct interest in knowledge of each other, only in the messages passed. In the context of the subscription part of the project, a publish-subscribe messaging pattern would mean that the sensors classify the data flow, and the external entities subscribe to the specific data flow classes which are of their interest. The sensors have no need to know who the subscribers are, or if there are any, and the external entities have no interest in the specific sensors, only in the data flows. From the messaging pattern follows scalability because of the lack of explicit

messaging between the publishers and subscribers. The publishers classify its data onto endpoints and the subscribers receive from the classes in which it has expressed interest. This means that new sensors can be added without large overhead by setting them as contributors to existing classes. New classes can also be made to accommodate new data flows from the new sensors. As no new coupling from a sensor to all external entities are necessary, such as is necessary with some other messaging patterns, these operations are scalable and will not add much overhead to an increasingly larger system. From the subscriber-side, adding new external entities and new class subscriptions for existing subscribers is as simple as allowing for a new connection onto the class endpoint.

Proprietary PubSub

A proprietary implementation of PubSub was assessed to require much more work to create and implement than the GraphQL-native implementation. Such a solution was therefore deemed an inefficient use of the group's time and was therefore discarded.

GraphQL PubSub

For this project, PubSub was implemented through the use of GraphQL subscriptions and WebSockets. This implementation was selected as it was the one already supported by the middleware chosen for the group's API, GraphQL. The selection of this version was based on ease of implementation, as the GraphQL version was functionally sufficient. The selection was based on the project being a broader proof-of-concept research, meaning to implement enough to prove feasibility and describing future possibility of further implementation to enhance scalability and performance.

DDS

During a sprint planning meeting, the customer representative specifically mentioned the DDS implementation of the publish-subscribe pattern, as it is the implementation already in use internally to the customer. As DDS is an implementation of the publish-subscribe pattern, it can be seamlessly implemented in place of the solution implemented in the proof-of-concept project. DDS has the reliability of the data stream as one of its primary goals and is therefore popular within applications that are mission- or business-critical, such as air-traffic controls. The fully-developed system is aimed to be used by seafood producers, meaning the data on the health of the fish may be business and mission-critical. Hence, the reliability of the time-sensitive data stream is of high priority and implementing DDS to the project in place of the GraphQL PubSub would be an effective upgrade. The scope of the project is variable with time and in the short-term, the system is to be used primarily for research, and later a possibility for commercial application. Therefore, scalability is also a concern. Scalability is already a feature of PubSub, but is further addressed by DDS. The further upgrade to DDS from GraphQL PubSub would allow for enhanced scalability for future data- and user-growth.

Conclusion

For the purposes of this project, a publish-subscribe messaging pattern was implemented through the GraphQL-subscriptions module. The implementation used is functional and allows for subscription to specific data flows, and therefore proves the feasibility of subscriptions in an open source capacity. For a large scale implementation, it is recommended to implement DDS in its place. As these are different implementations of the same concept, the DDS can slot into the place of the current implementation with little alteration needed. With scalability being a chief attribute of the envisioned final system, inserting DDS should be considered.

A.3 Front End Client

This section provides a detailed description of the design & implementation details of the front end client in the application. Throughout the implementation phase of the project, the group followed the architecture design created during the pre-study phase. The client is a web application built with the JavaScript library React at its core. The client communicates with the back end through the API described in Section A.2. The main open source software (OSS) and the commercial off-the-shelf (COTS) products used are described in subsection A.3.1 followed by the design prototype that the client is based on in subsection A.3.2.

A.3.1 OSS and COTS

React

React is an open source JavaScript library for building user interfaces. The documentation for React can be found here: `https://reactjs.org/docs/getting-started.html`. React code is composed of entities called components that can be rendered in the DOM (Document Object Model). The client uses functional components with React Hooks to manage components' life cycles and the state within the functional components. React Router is used to enable navigation between different pages. The package diagram in Figure D.15 shows how the components and their styles are organized.

Redux

Redux is an open source JavaScript library for managing application state. The documentation for Redux can be found here: `https://redux.js.org/introduction/getting-started`. The client uses Redux to manage the global state that needs to be shared between components in React. The class diagram in Figure 7.1 shows how Redux is organized inside the *globalState* folder.

Apollo Client

Apollo Client is an open source JavaScript GraphQL client for executing queries, mutations, and subscriptions on data graphs. The documentation for Apollo Client can be found here: `https://www.apollographql.com/docs/react/`. The front end client uses Apollo Client to query and mutate data in the database through the API provided by the back end. The class diagram in Figure 7.1 shows how the code for using Apollo Client is organized inside the *requests* folder and the package diagram in Figure 7.4 shows how it is connected to Apollo Client.

Microsoft Azure Active Directory

Microsoft Azure Active Directory is an identification service that provides sign in possibility by registered Microsoft accounts with multi-factor authentication, and is used for both authentication and authorization. The documentation for Azure Active Directory (AAD) can be found here: `https://docs.microsoft.com/en-us/azure/active-directory`. For the project, the group created a new directory at the `https://aad.portal.azure.com/`, defined the engineer, researcher, and customer roles, and gave the group members access to the directory. In the directory, two app registrations were made: FishFarmData and FishFarmDataAPI. This was done to enable verification of users from the API and the web. In the client, the communication with AAD is done through the react-aad-msal package [44], by using the AzureAD component and an authentication provider. The id token is stored in the Redux store, and the access token is fetched from AAD or cache and sent to back end in the header.

Highcharts

Highcharts is a COTS JavaScript library for creating interactive charts on a web page. The documentation for Highcharts can be found here: `https://www.highcharts.com/docs/`. The client uses Highcharts to visualize time series data as graphs. Highcharts provides a lot of different graph types and has an intuitive user interface. Figure A.12 and Figure A.11 show how Highcharts is used in the client.

A.3.2 Design Prototype

The group decided to create a design prototype for two main reasons. The first reason was to figure out how to create a user interface that solves the majority of the functional requirements listed in Table 6.1. The second reason was to create a system with high usability. Creating a design prototype helps to identify and formulate the main trajectory of the design and saves time. The design prototype is visualized in the Figures A.9 through A.16, and shows how the user interface in the client works.

This paragraph shows the most important pages of the design prototype. The full prototype can be viewed and downloaded from here: `http://folk.ntnu.no/ivarnm/TDT4290_Customer_Driven_Project/FishFarmData_prototype.pdf`. The user guide for how to use the product is located in Section A.1. Figure A.9 shows the start page that the user sees when not signed in to the application. When signed in, the user is redirected to the dashboards page in Figure A.10 where the user gets an overview of their private and shared dashboards. Figure A.11 shows how a dashboard with cells looks like and Figure A.12 shows the user interface for adding a cell to the dashboard. In Figure A.13 a modal shows the metadata for the different sensors in a cell. If the access role of the user is Engineer, they have access to the admin page. Figures A.14, A.15 and A.16 respectively show how to upload sensor data, edit metadata and manage sensor access in the admin page.

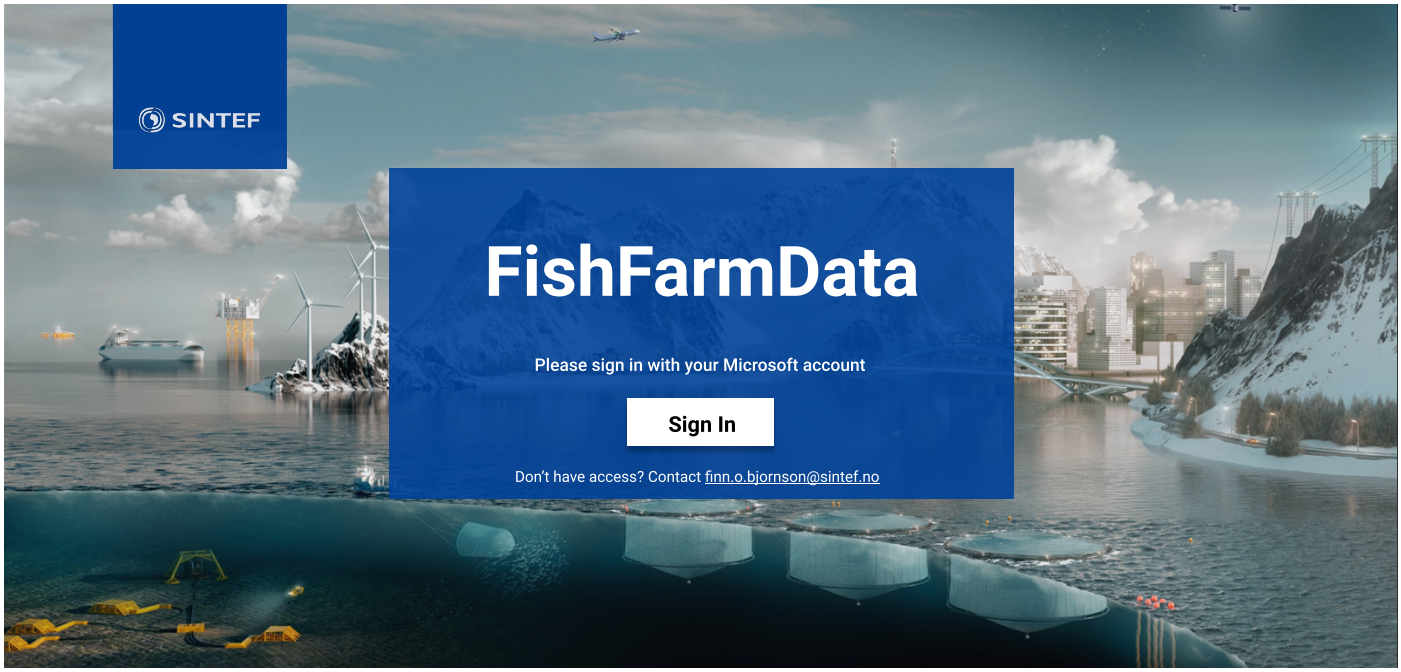


Figure A.9: Landing page when not signed in

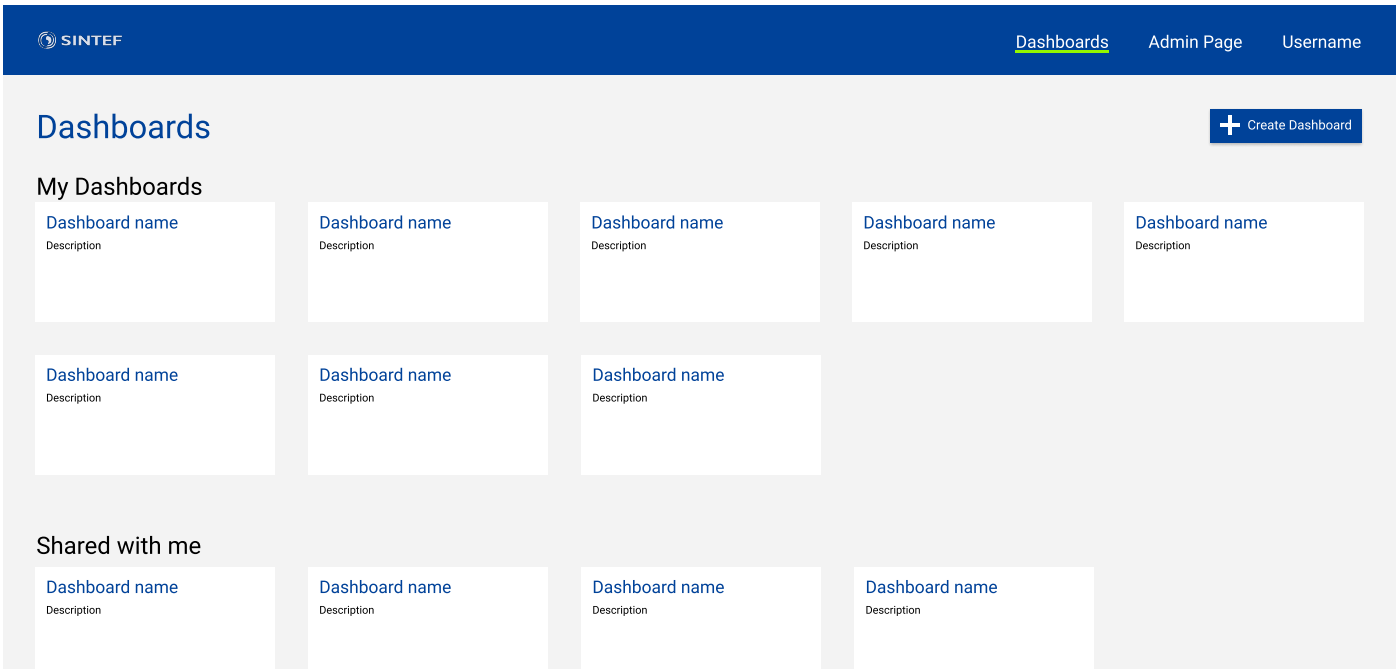


Figure A.10: Overview of your private and shared dashboards

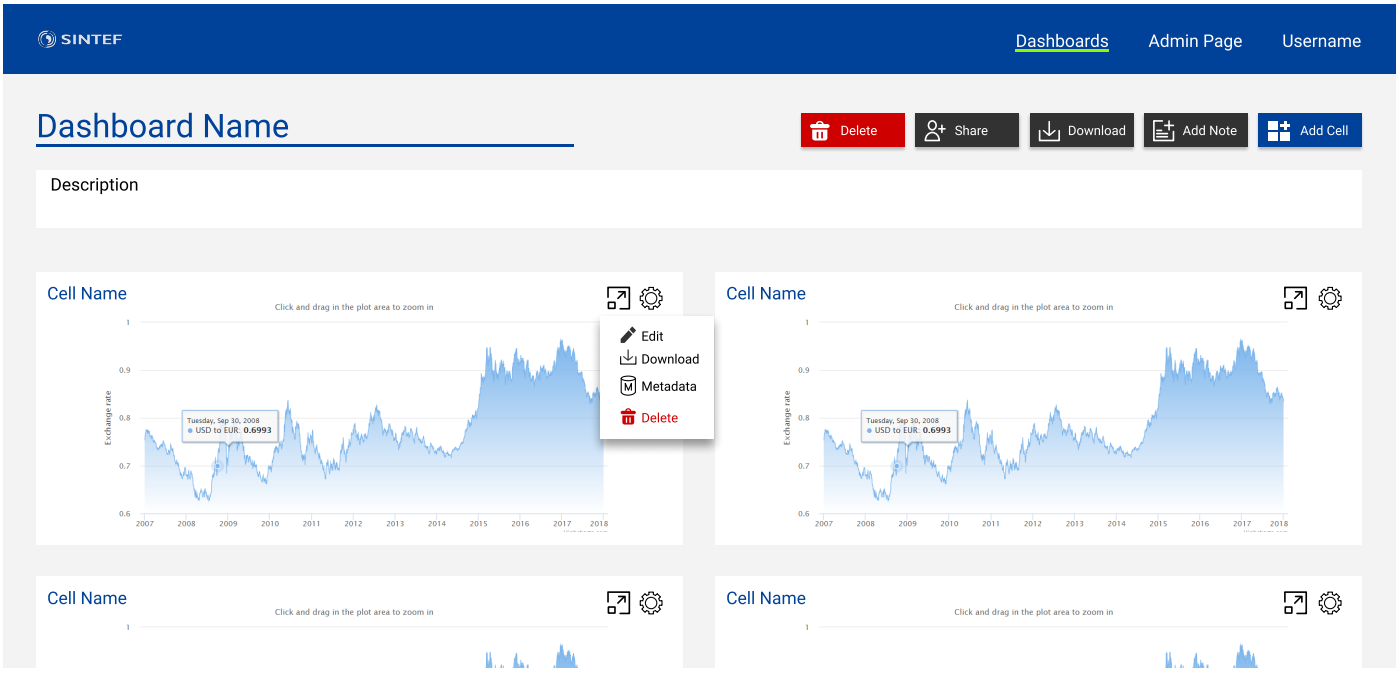


Figure A.11: Overview of a specific dashboard

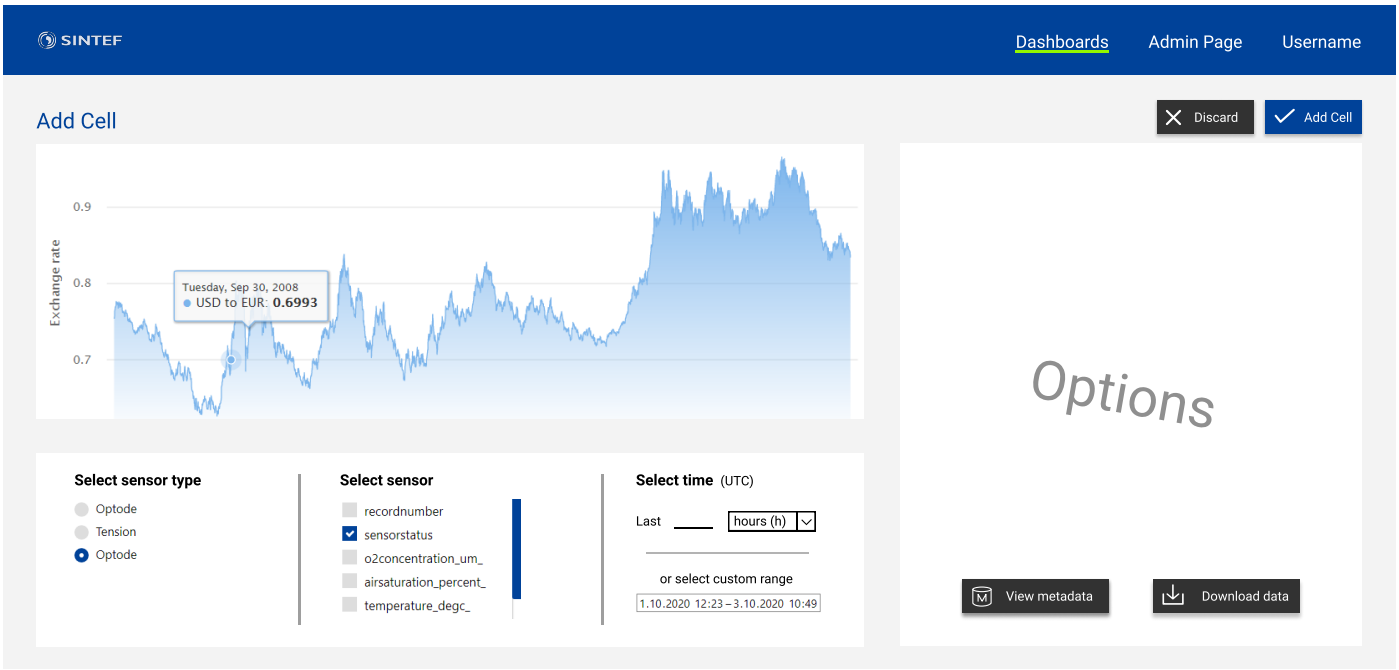


Figure A.12: Page for adding a cell to a dashboard

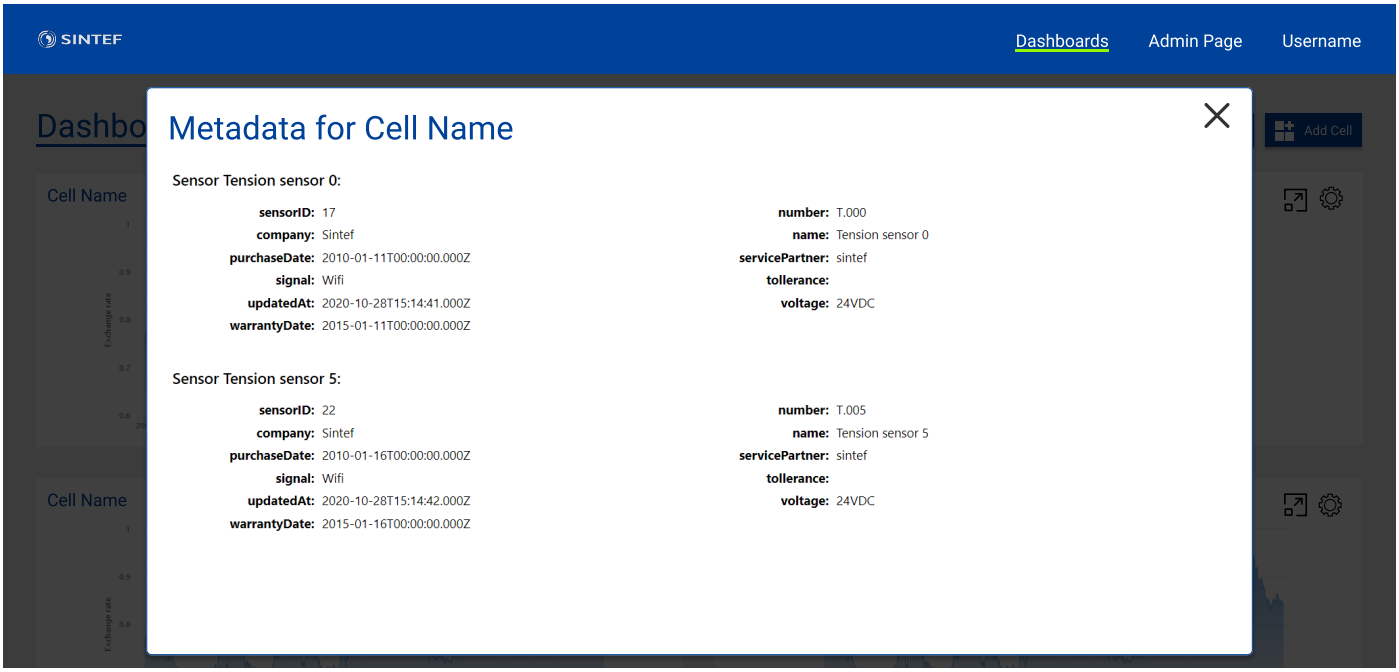


Figure A.13: Modal showing a cell's metadata

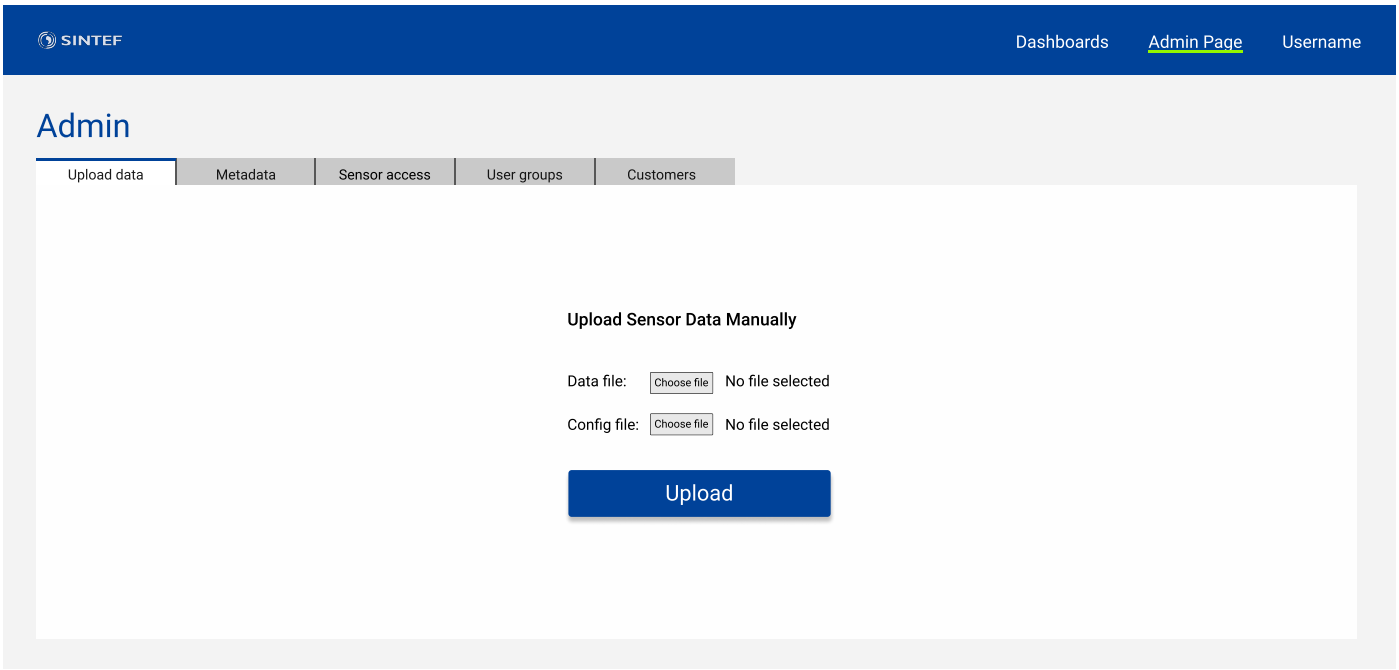


Figure A.14: Page for manually uploading sensor data

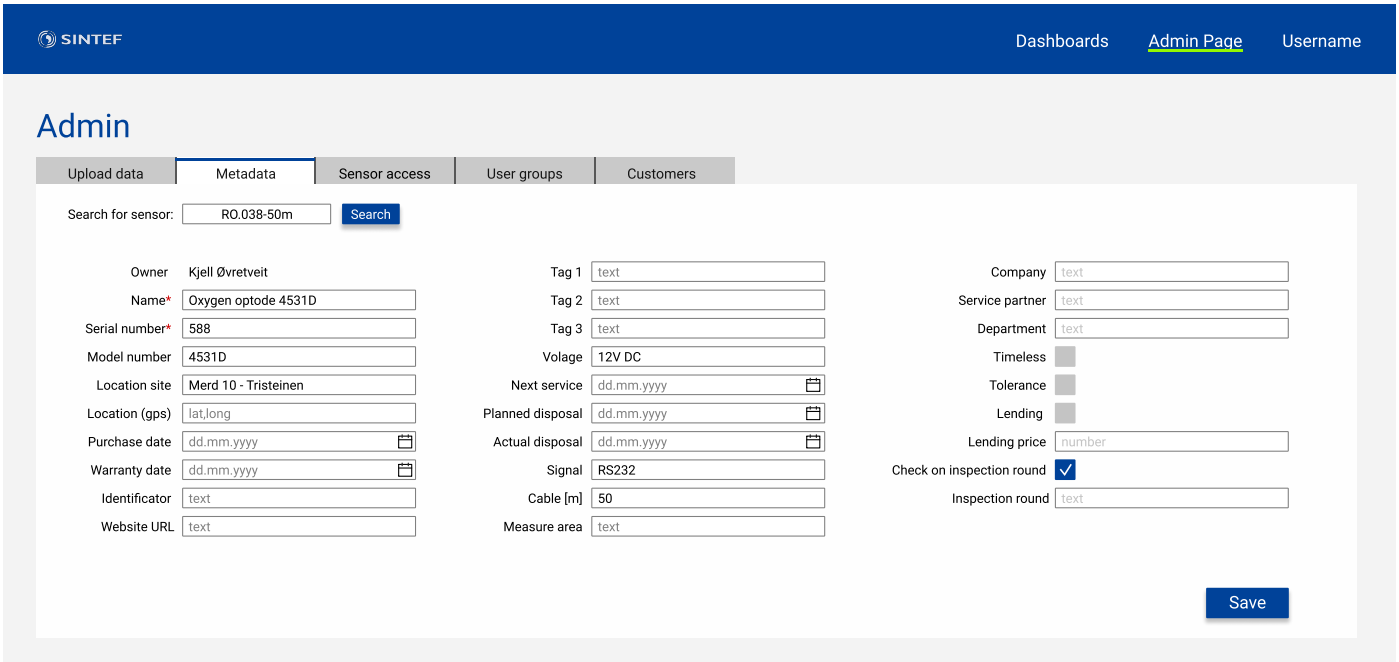


Figure A.15: Page for editing metadata of a sensor

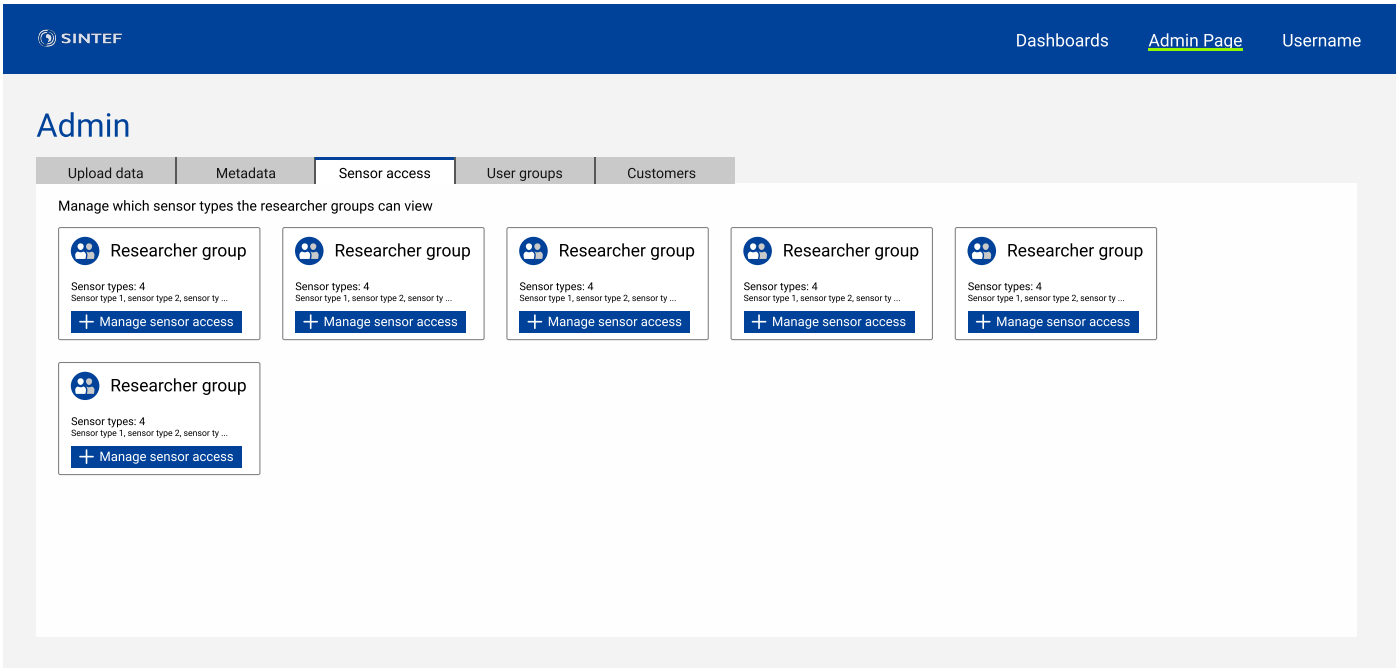


Figure A.16: Page for managing sensor access

B. Git Conventions

To keep the repository neat and organized, it is important that everyone follows the same git conventions. The repository have two protected branches, main and dev. This means it is needed pull request to merge code into these branches. This is an important quality assurance, and ensure that the code being merged to dev and main is inspected by someone else.

B.1 Git Branching Convention

Git branches should be named following this format:

branchtype/branchname

This are the branchtypes:

- feat: New feature
- enh: Enhancement on exisiting feature
- bug: bugfixes
- test: New test
- hot: Hotfixes (Urgent bug fixes deployed directly to main/dev)
- documentation: Improvements or additions to documentation
- duplicate: This issue or pull request already exists
- The branchname should be an indication of which issue you are working on.

B.2 Commit Message Structure

The commit message should follow this message structure:

JIRA-ID: commit message

Commit message should describe what the code does, and not what you have done.

C. Test Results

C.1 Functional requirements

FR1: As an engineer, I want to be able to add new data with different formats

Executor	Sebastian Aas
Date	28.10.2020
Time used	5 minutes
Evaluation	Success
Comment	To add new data, you need to write a new configuration file in the correct format. This is used for parsing the data format of the new sensor type.

FR2: As an engineer, I want to be able to save metadata connected to time series

Executor	Sebastian Aas
Date	28.11.2020
Time used	30 seconds
Evaluation	Success
Comment	When adding metadata for a new sensor in the admin panel, a new sensor is automatically created, connected to the metadata.

FR3: As a researcher/user, I want to be able to view data

Executor	Sebastian Aas
Date	28.10.2020
Time used	60 seconds
Evaluation	Success
Comment	First, one needs to create a dashboard, then create cells with the wanted data. After this, one can see it on the dashboard.

FR4: As a user, I want to be able to access an API to retrieve data

Executor	Sebastian Aas
Date	28.10.2020
Time used	3 minutes
Evaluation	Success
Comment	The API used for retrieving data for the front end is accessible to retrieve data. Publish-Subscribe functionality for retrieving data is implemented using GraphQL subscriptions and WebSockets.

C.2 Quality Attributes Scenarios Testing

FR5: As an engineer, I want to be able to add new sensors

Executor	Sebastian Aas
Date	28.10.2020
Time used	30 seconds
Evaluation	Success
Comment	When adding metadata for a new sensor in the admin panel, a new sensor is automatically created.

FR6: As an engineer, I want to be able to upload data manually

Executor	Sebastian Aas
Date	28.10.2020
Time used	90 seconds
Evaluation	Success
Comment	Given that the needed sensors are created, and that the configuration file is created, the new data is uploaded via the admin panel "Upload data".

FR7: As an engineer, I want to be able to modify metadata for a time series

Executor	Sebastian Aas
Date	28.10.2020
Time used	30 seconds
Evaluation	Success
Comment	In the admin panel, metadata connected to a sensor can be updated and saved to the database. Also possible through the API.

FR8: As a researcher/engineer, I want to be able to store large amounts of data

Executor	Sebastian Aas
Date	28.10.2020
Time used	N/A
Evaluation	Success
Comment	The time series database is capable of retaining large amounts of data, and with the group's testing it was able to retain and retrieve the uploaded data.

FR10: As a researcher/user, I want to be able to visualize data easily

Executor	Sebastian Aas
Date	28.10.2020
Time used	50 seconds
Evaluation	Success
Comment	In a dashboard, one can create cells that visualize data.

FR11: As a user, I want to be able to view the data on the web

Executor	Sebastian Aas
Date	28.10.2020
Time used	10 seconds
Evaluation	Success
Comment	The product is available on the web.

FR16: As a user, I want to view data on a static page

Executor	Sebastian Aas
Date	28.10.2020
Time used	30 seconds
Evaluation	Partial success
Comment	A customer user of the system with a dashboard, can view the dashboard on a static page. Only the static page is implemented, not the access control of dashboards.

FR23: As a user, I want to be able to interact with graphs

Executor	Sebastian Aas
Date	28.10.2020
Time used	15 seconds
Evaluation	Success
Comment	It is possible to zoom and drag the graphs inside a cell.

M1: Add new request

Executor	Sander Kilen
Date	01.10.2020
Environment	Design time
Stimuli	Add a new request to the API do be able to fetch data.
Expected response measure	Request added and accessible within 4 hours.
Observed response measure	Request added and accessible within 2 hours
Evaluation	Success. Adding a new request requires adding business logic, database connection, and the query or mutation.
Comment	If the business logic is already established, creating a new request is of low complexity and not the most time-consuming.

M2: Add data from a new file format

Executor	Bjørn Andre Aaslund
Date	03.10.2020
Environment	Design time
Stimuli	Create a driver to decode the new data format.
Expected response measure	Component created and connected within 1 work day.
Observed response measure	Component created and connected within 7 hours.
Evaluation	Success. Add a new driver for parsing new data files, encoding this for reading values for writing to the database.
Comment	If business logic.

U1: Create dashboard for measurements

Executor	Ivar Myrstad and external test subject.
Date	14.10.2020
Environment	Runtime
Stimuli	Create a new dashboard
Expected response measure	Understand how to create a dashboard with a cell within 3 minutes.
Observed response measure	A dashboard with a cell created in 2 minutes.
Evaluation	Success. A researcher can create a dashboard, and then add a cell to the newly created dashboard.
Comment	The user completed the task without any problems.

U2: Modify metadata

Executor	Ivar Myrstad and external test subject.
Date	14.10.2020
Environment	Runtime
Stimuli	Modify metadata for a sensor
Expected response measure	Understand how to change and save metadata for a sensor within 3 minutes.
Observed response measure	Metadata for sensor was changed and saved in 2 minutes.
Evaluation	Success. A engineer can select a sensor and edit its metadata.
Comment	The user completed the task without any problems.

S1: Admin page access

Executor	Turid Dahl
Date	02.10.2020
Environment	Normal operations
Stimuli	Attempts to enter admin page
Expected response measure	Admin page is protected from unauthorized access immediately.
Observed response measure	Admin page was protected from unauthorized access immediately.
Evaluation	Success. If an unauthorized user tries to access the admin page, an "Access Denied" page is shown instead.
Comment	The "Access Denied" page is shown on any page attempted access by an unauthorized user.

S2: Dashboard access

Executor	Sander Klien
Date	14.10.2020
Environment	Normal operations
Stimuli	Attempts to fetch dashboards
Expected response measure	Data is not accessible without valid credentials immediately.
Observed response measure	Data is not accessible without valid credentials immediately.
Evaluation	Success. With valid credentials, the data is fetched. With invalid or missing credentials, an access denied error is returned.
Comment	A valid JWT bearer token in the request header is needed to access the secured requests in the API.

D. Additional Figures

D.1 Planning

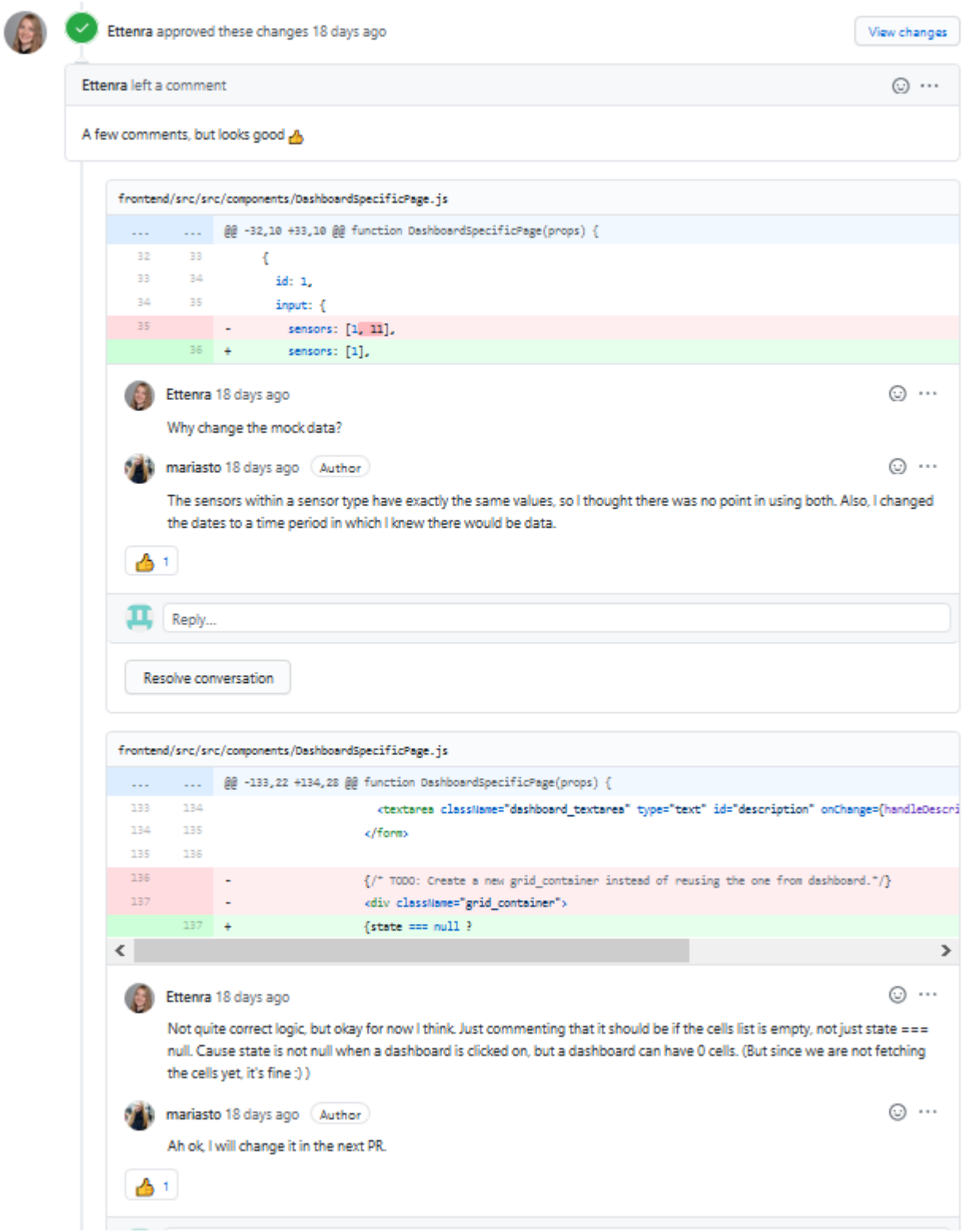


Figure D.1: An example of a code review.

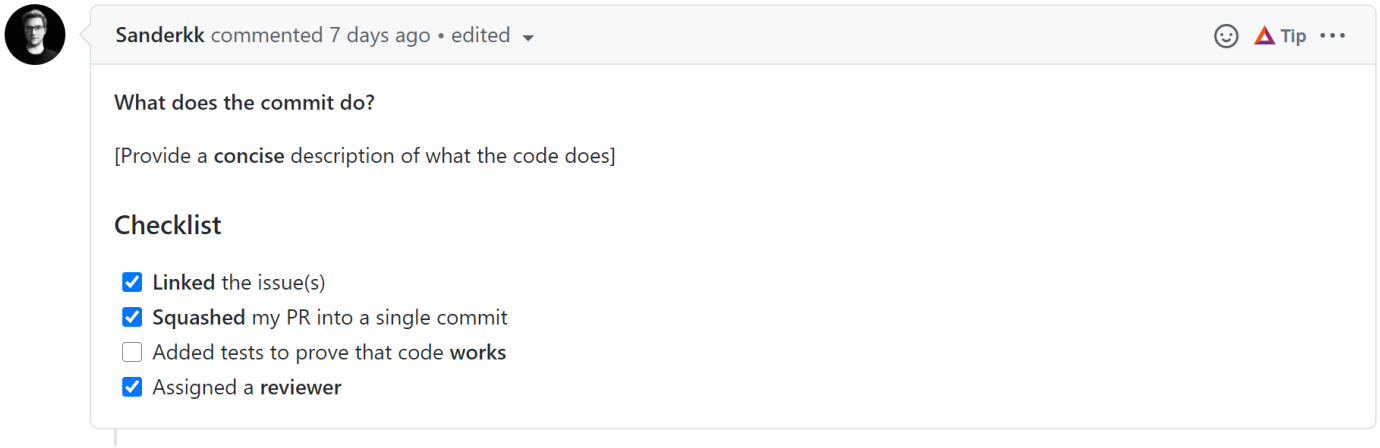


Figure D.2: Pull request template. The checklist is quite simple, but can make a huge improvement for the quality assurance.

Dato: DD.MM

Gruppemøte Kundestyrt Prosjekt Gruppe 6

1. **Innsjekk**
   (maks 1-2min pr pers)
2. **Daily scrum**
   (Skal være nokså kort, maks 20 min)
   "- What did I do yesterday that helped our team meet the sprint goal?
   - What will I do today to help our team meet the sprint goal?
   - Do I see any impediments that prevent me or our team from meeting the sprint goal? " - (tatt fra Scrum-boka fra PU)
3. **Punkter man ønsker å ta opp/Problemer man har støtt på**

Tema	Konklusjon

4. **Hva man skal jobbe med til neste gang (?)**

Figure D.3: To improve the consistency of project, there was used templates for different meetings. This is the template for a weekly group meeting.

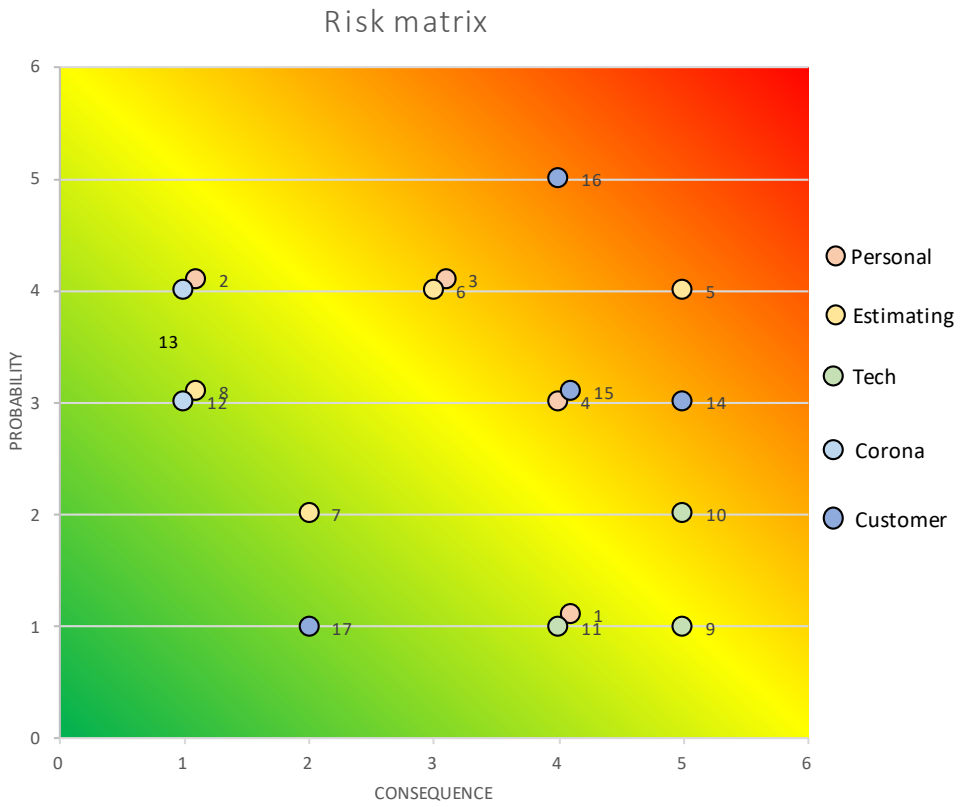


Figure D.4: Risk matrix for the identified risks.

D.1.1 Effort Registration

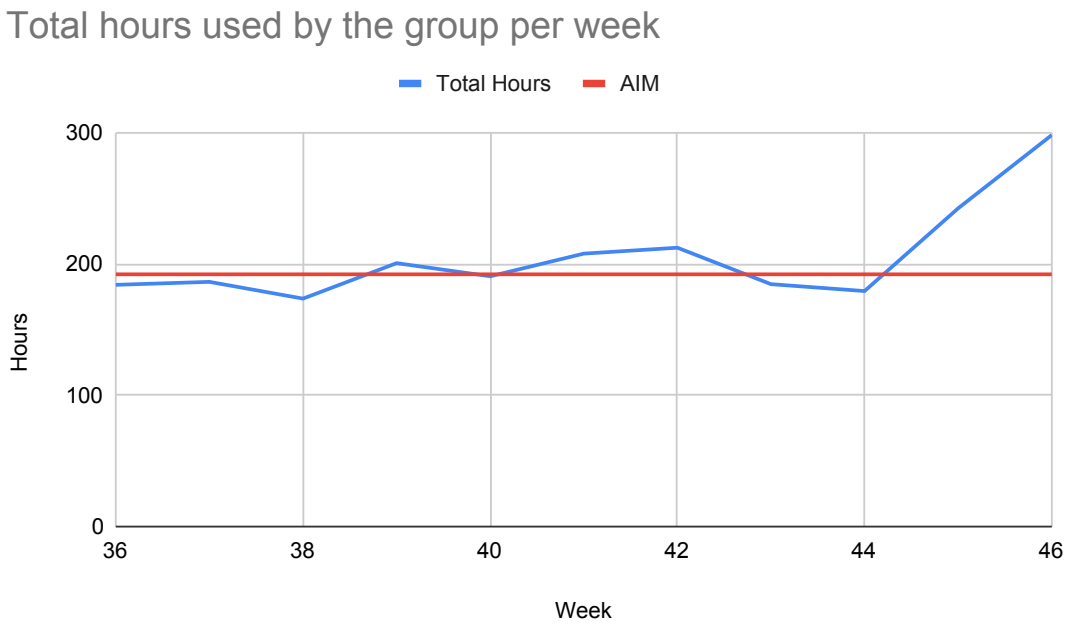


Figure D.5: Line graph showing the total hours of work the group has done per week.

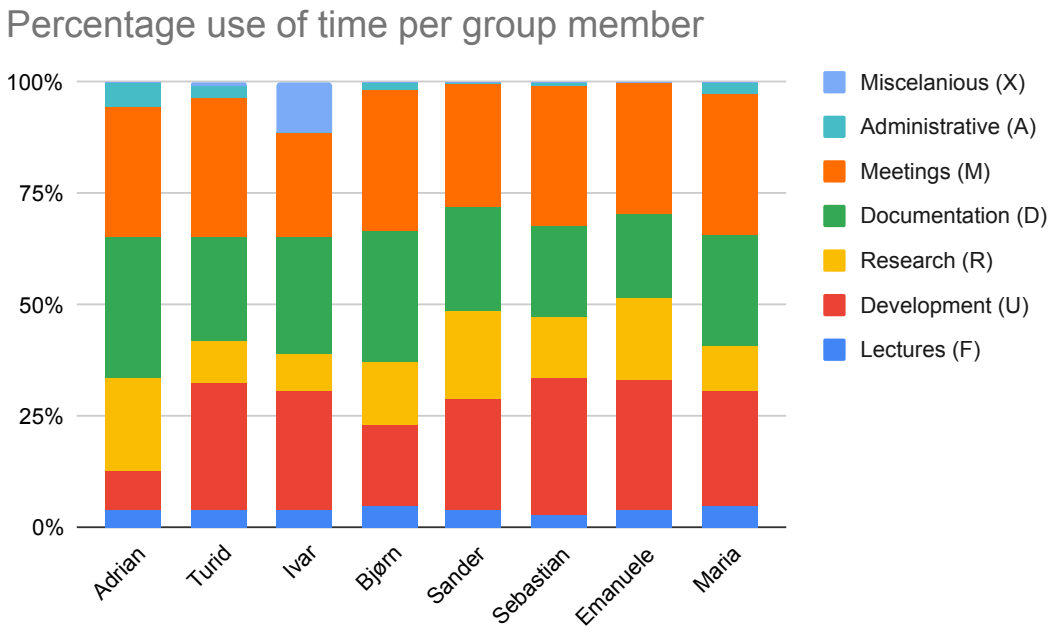


Figure D.6: Stacked bar chart, showing percentage use of time per group member throughout the whole semester.

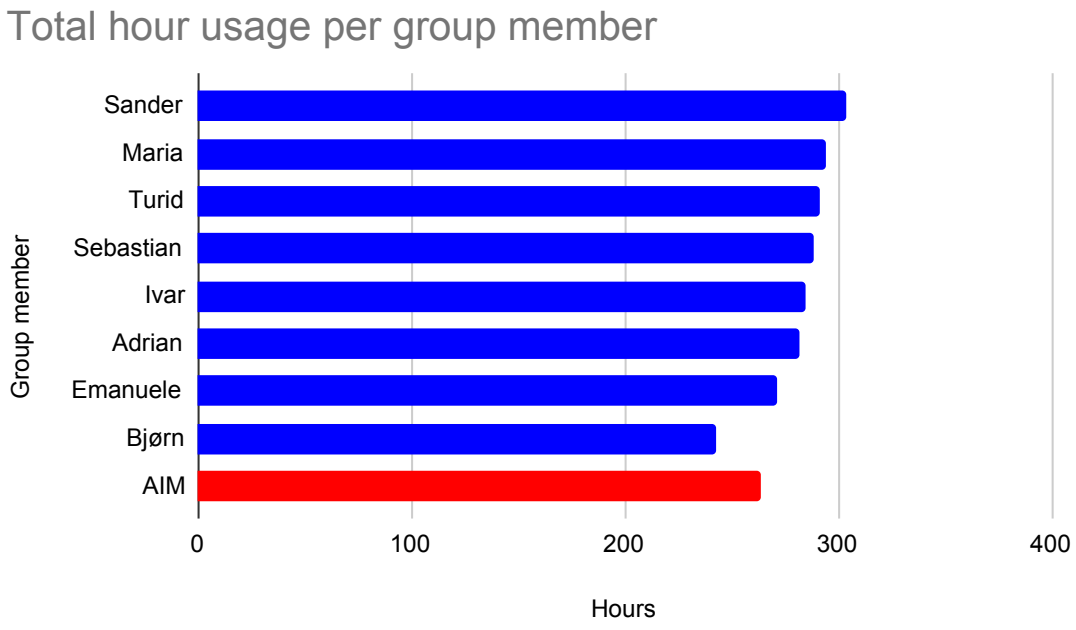


Figure D.7: Bar chart showing how many hours each group member has used on the course.

D.2 Pre-Study

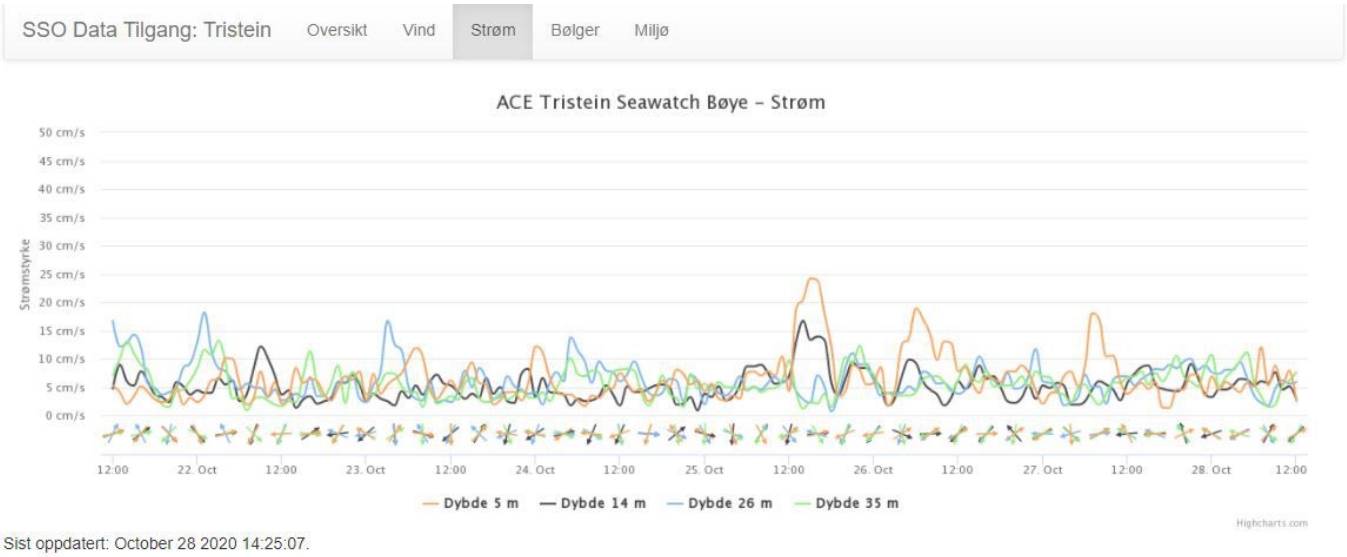


Figure D.8: Example graph for visualizing data in the customer's wanted solution

D.3 Development Methodology

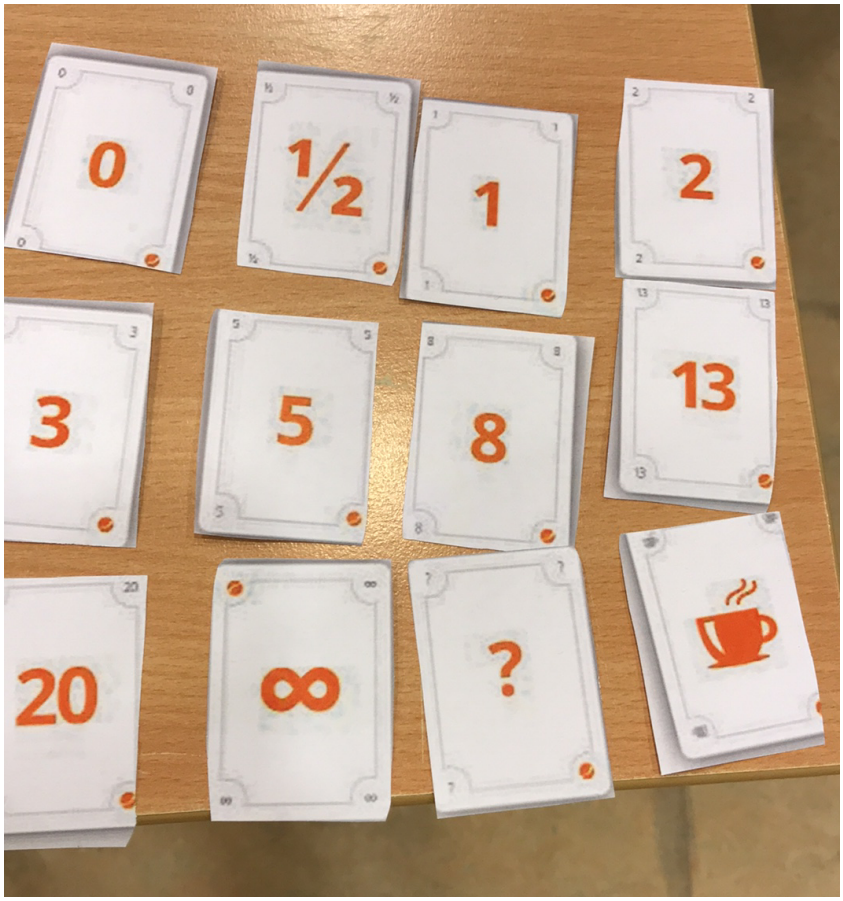


Figure D.9: The paper printed cards used for Planning Poker.

D.4 Architecture

D.4.1 First Designed Architecture

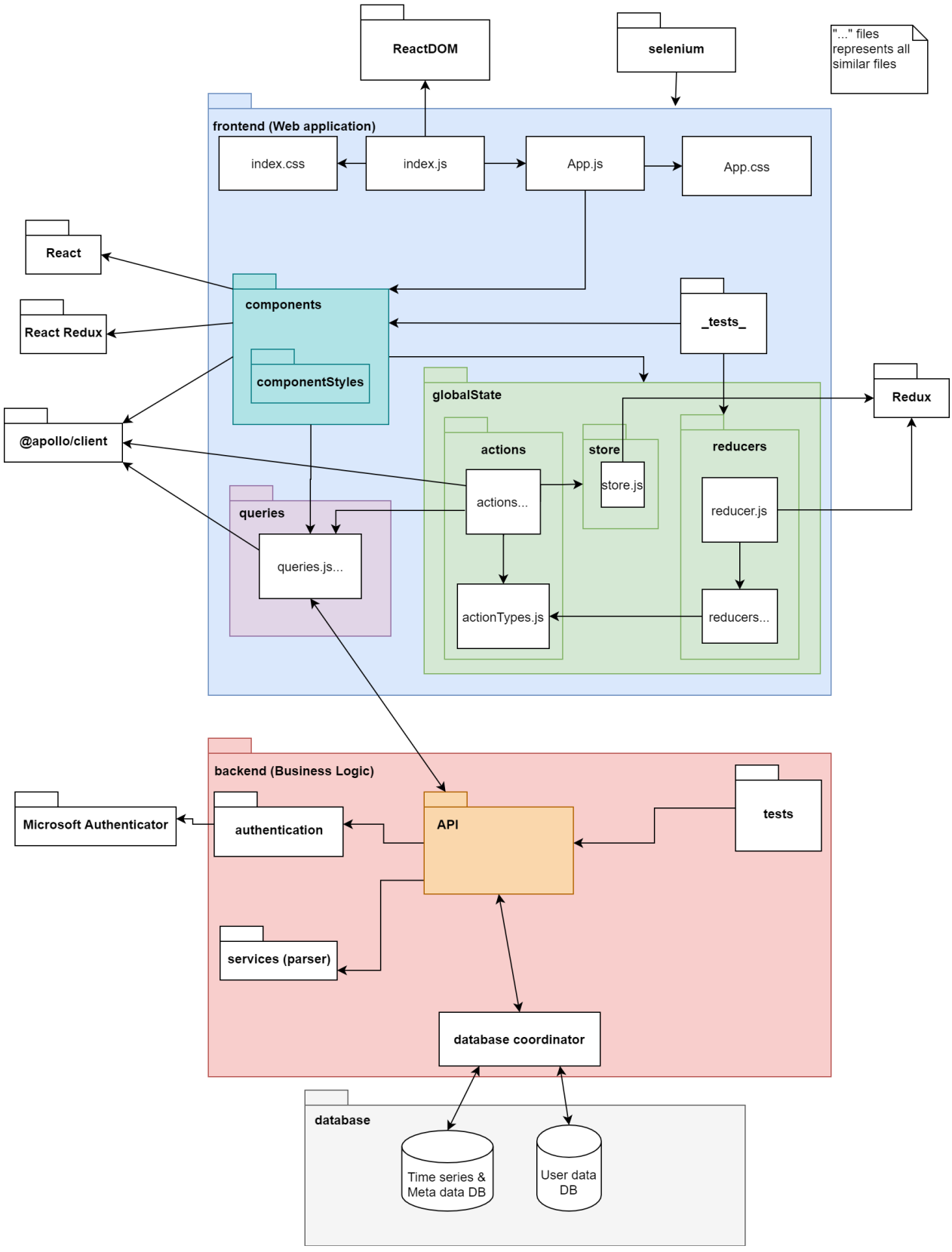


Figure D.10: First designed class diagram representing the logical view

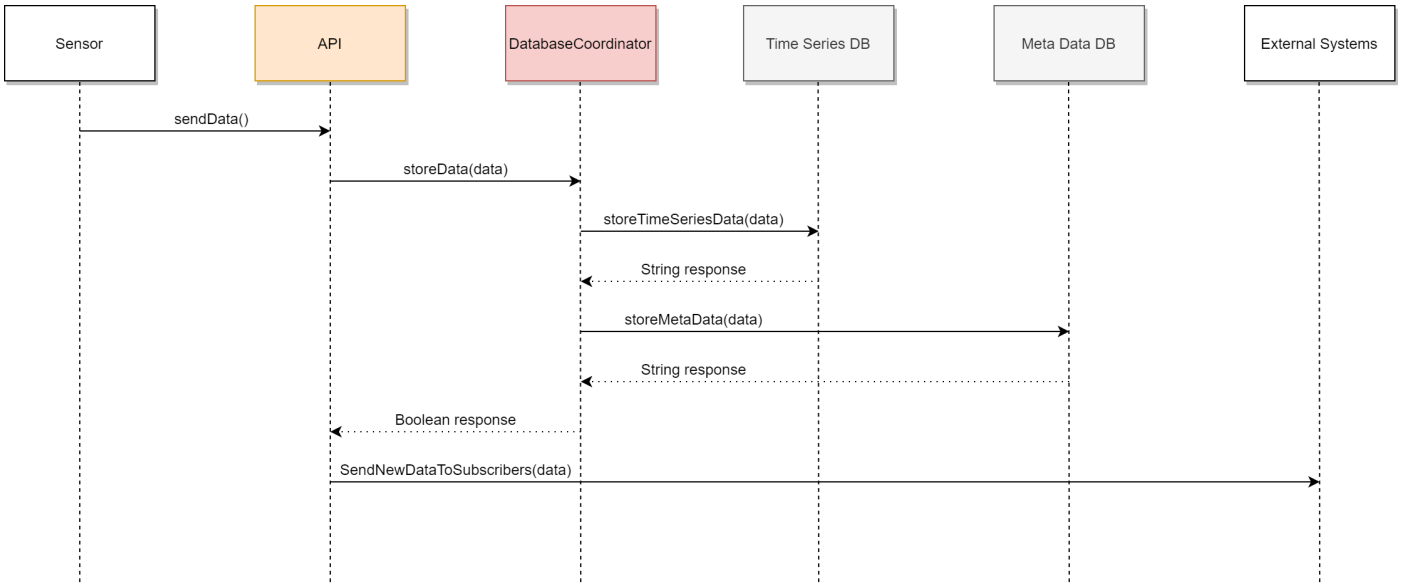


Figure D.11: First designed sequence diagram representing the process view

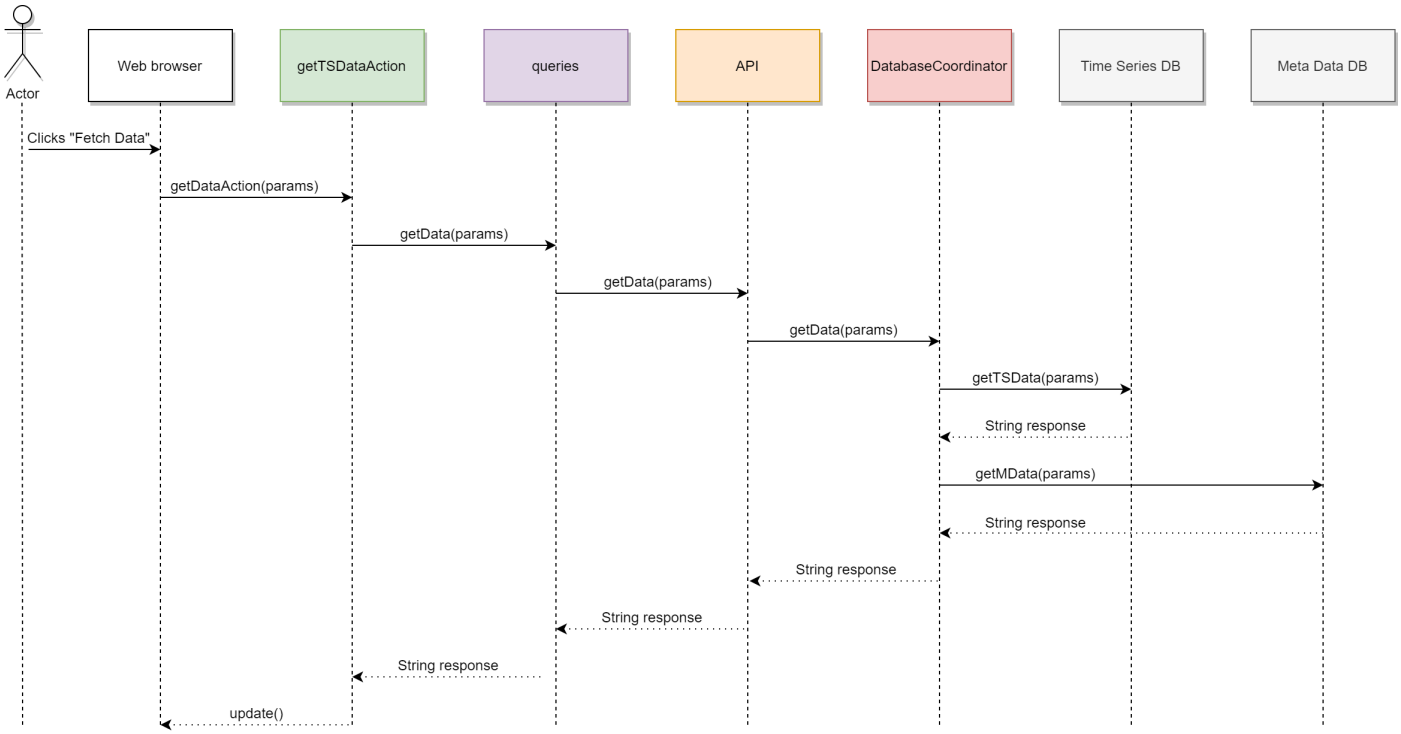


Figure D.12: First designed sequence diagram representing the process view

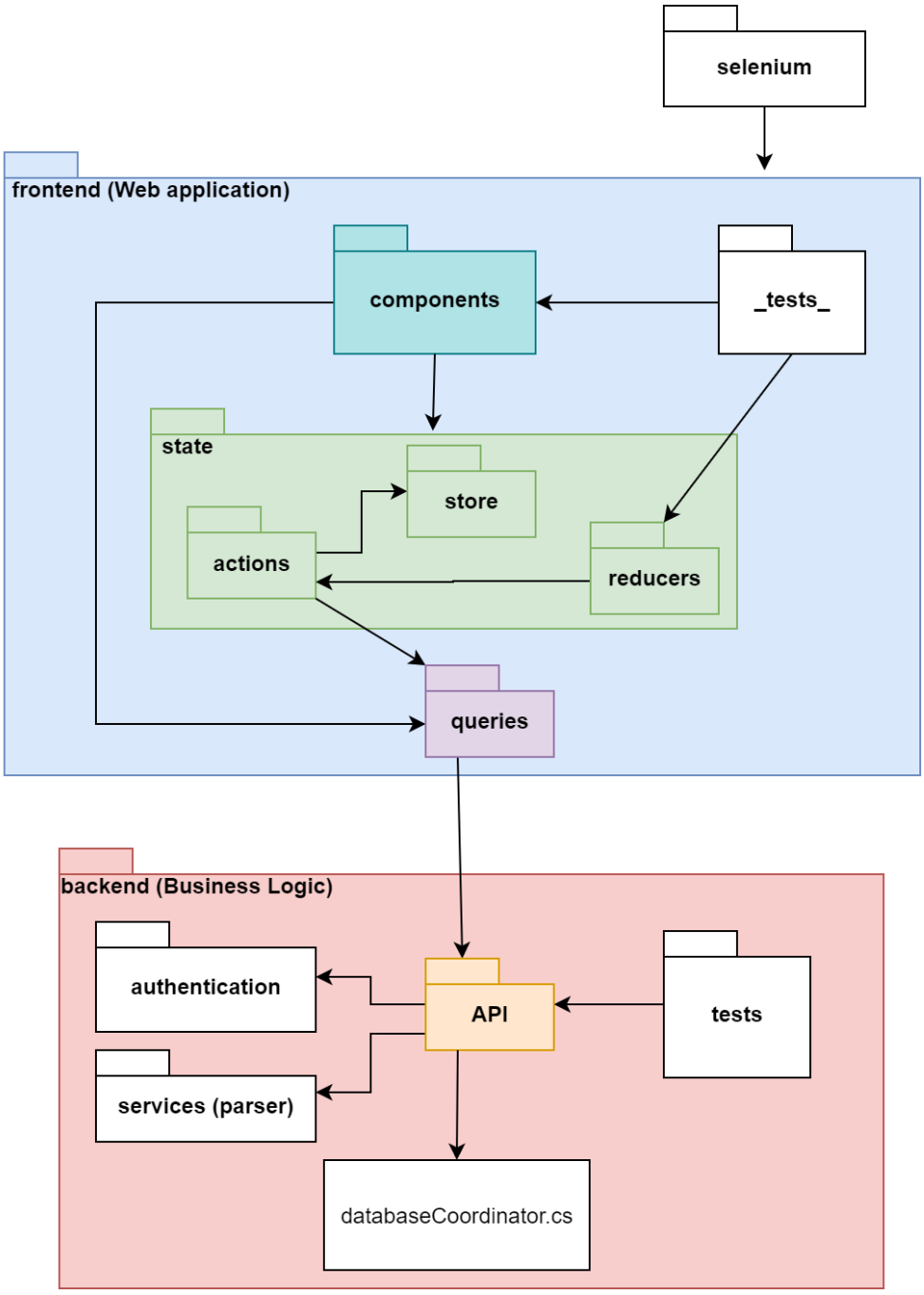


Figure D.13: First designed package diagram representing the development view

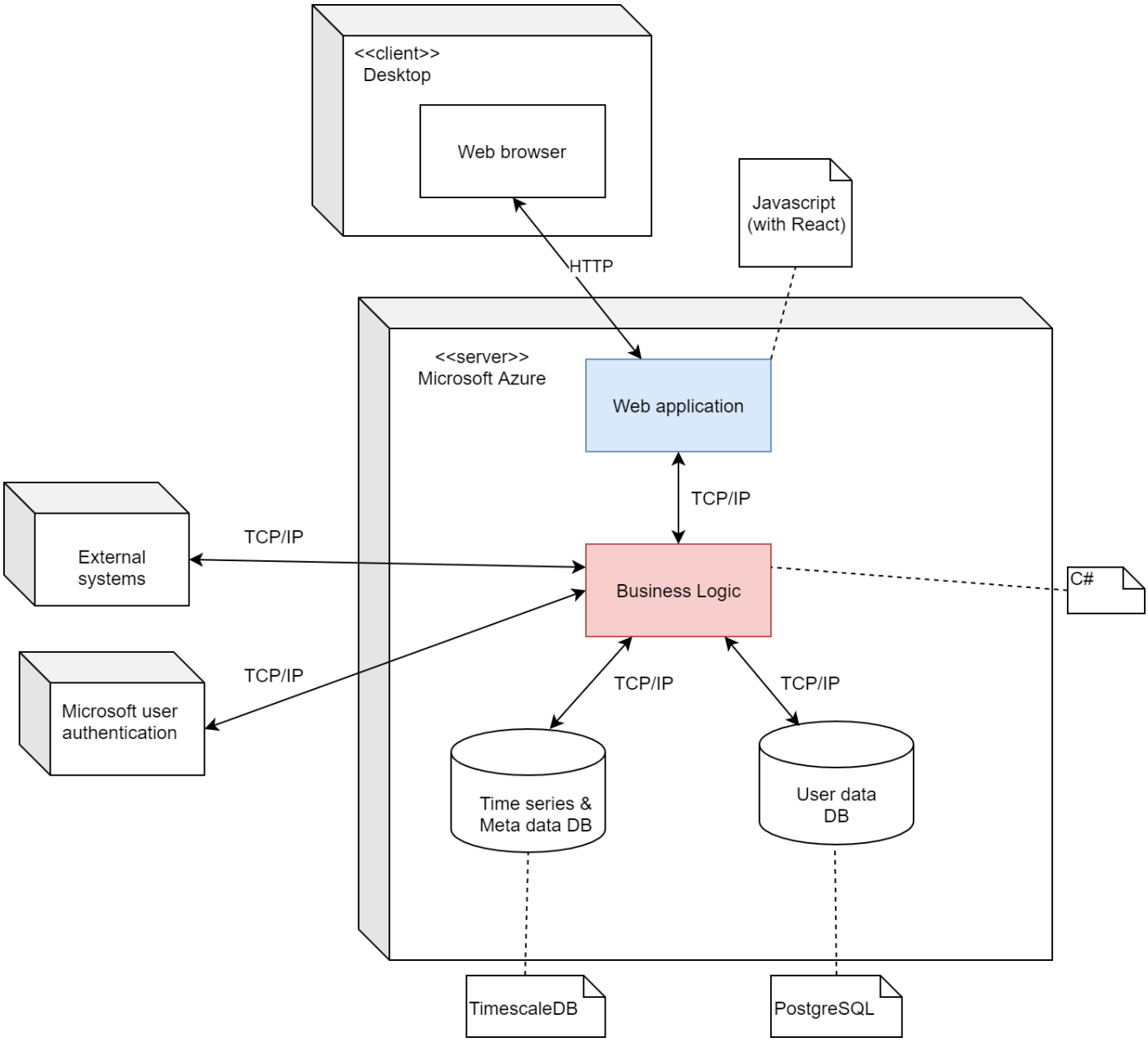


Figure D.14: First designed deployment diagram representing the physical view

D.4.2 Architecture

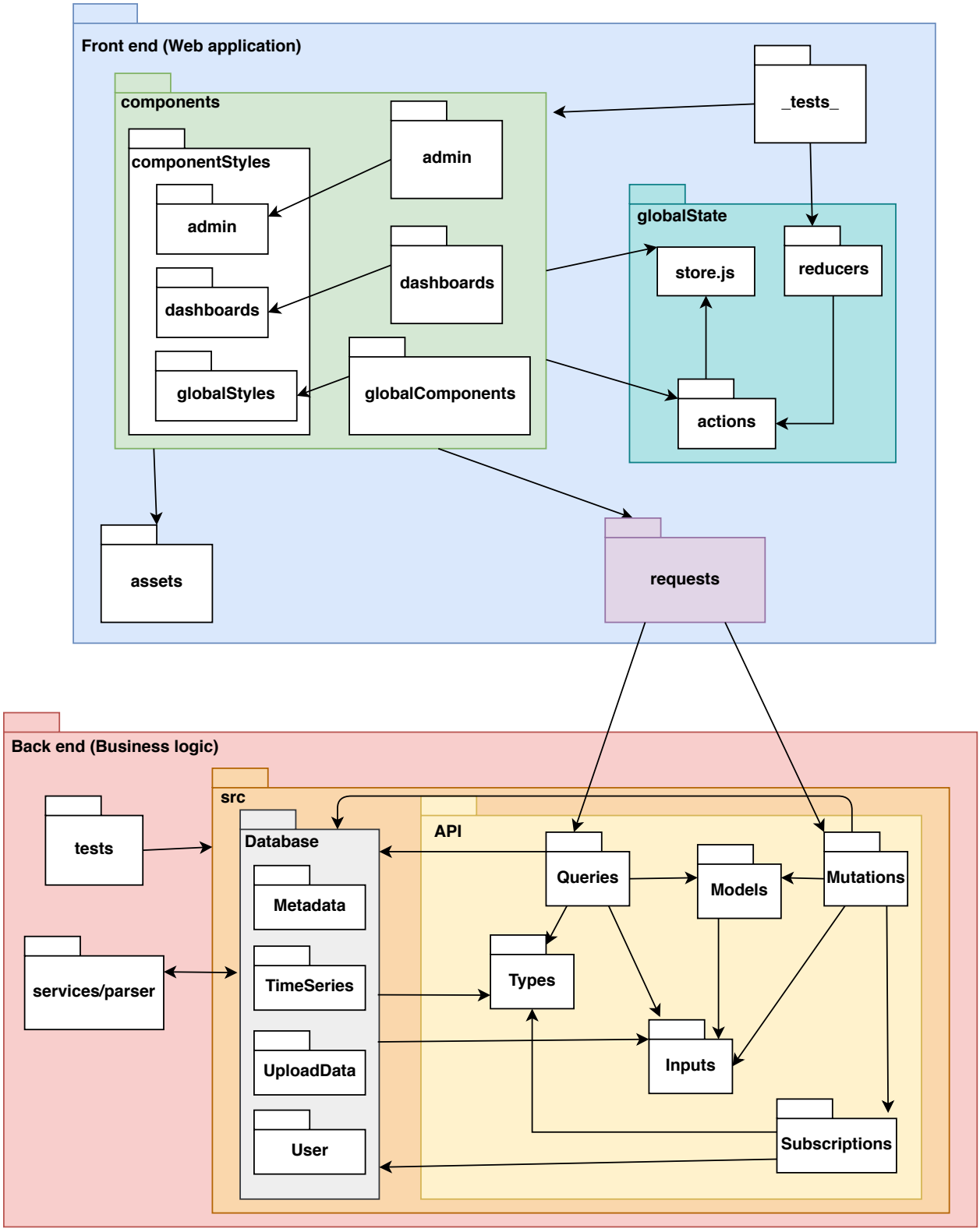


Figure D.15: A more detailed package diagram of the architecture, without external packages

D.5 Implementation

D.5.1 Sprint 1

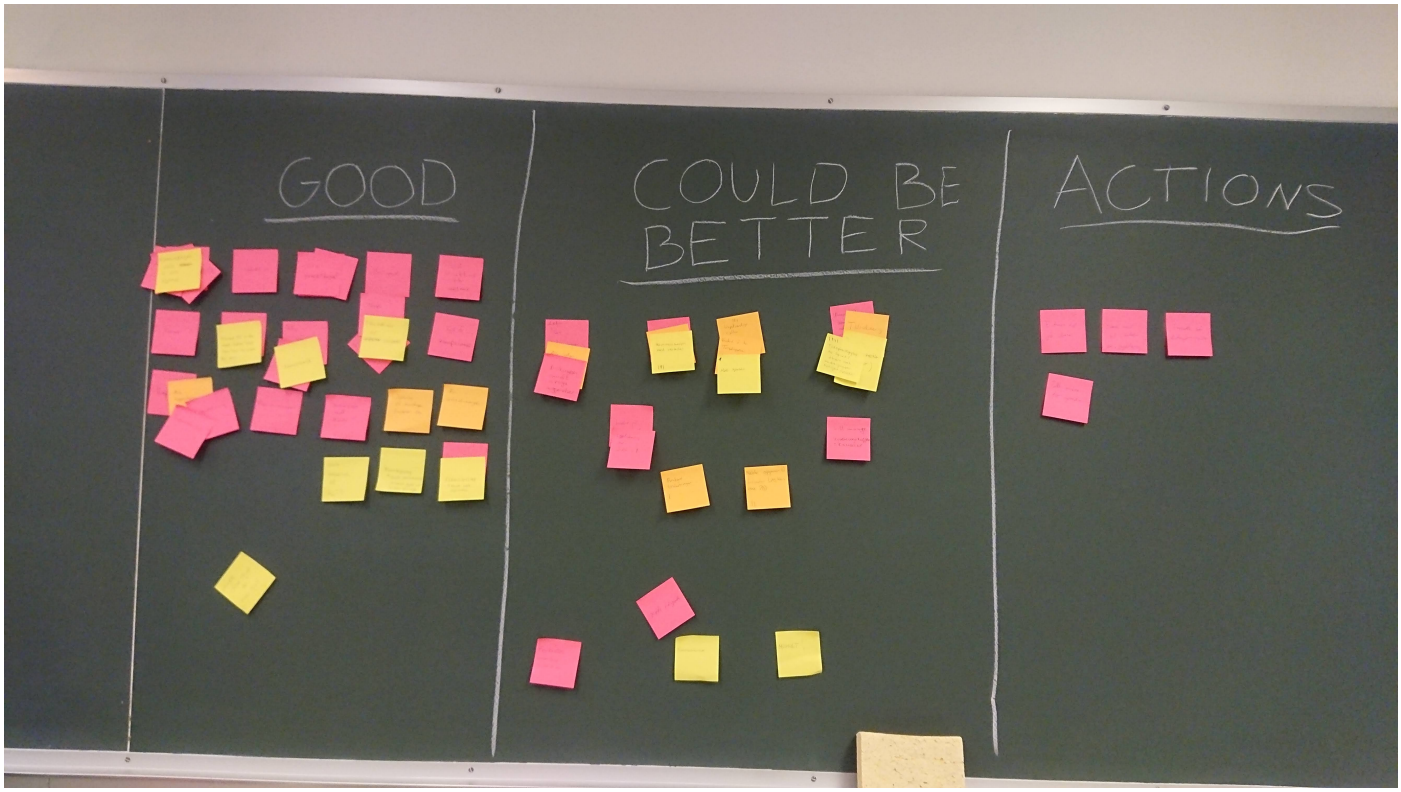


Figure D.16: A photo of all notes and their placement during sprint 1 retrospective

D.5.2 Sprint 2

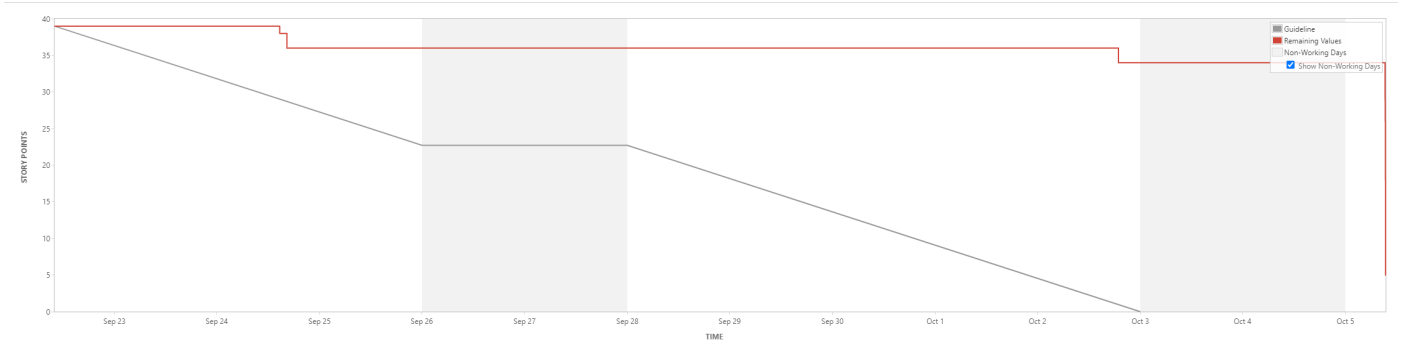


Figure D.17: Burndown chart showing Story Points completed throughout Sprint 2. Ended October 5 instead of October 3 because of when the Sprint Review took place.

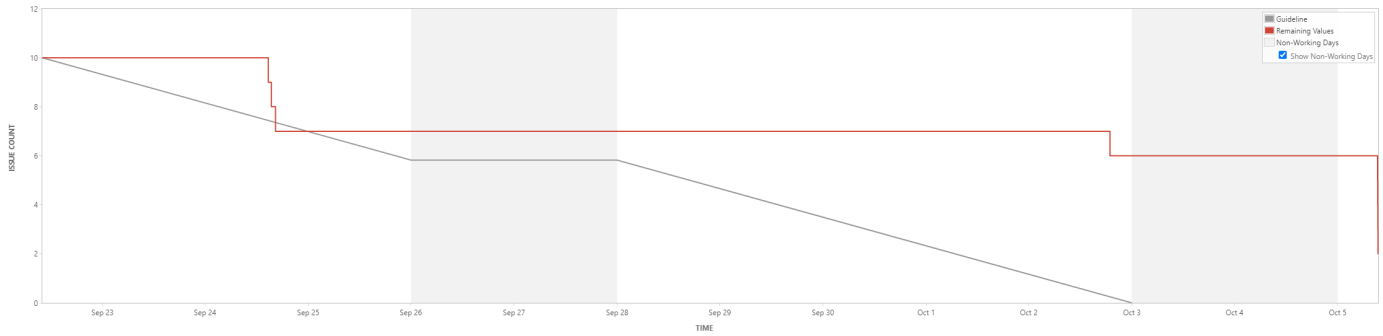


Figure D.18: Burndown chart showing issues completed throughout Sprint 2. Ended October 5 instead of October 3 because of when the Sprint Review took place.

D.5.3 Sprint 3

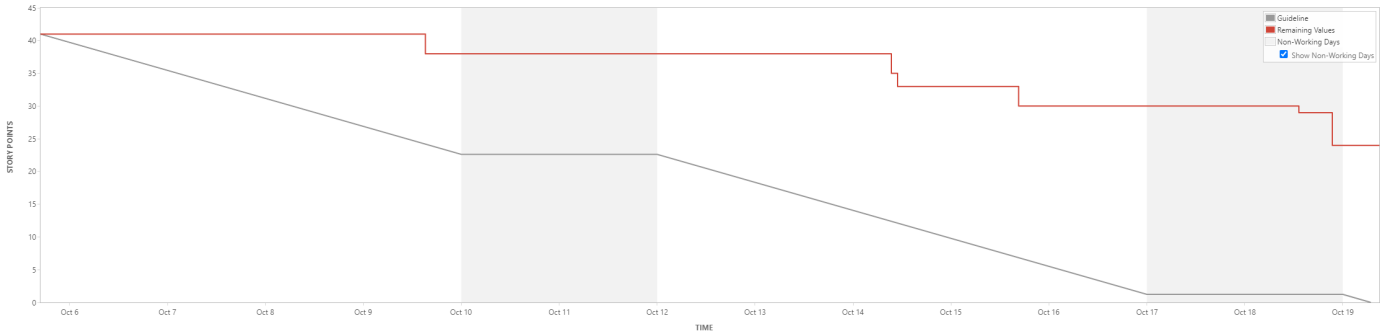


Figure D.19: Burndown chart showing Story Points completed throughout Sprint 3

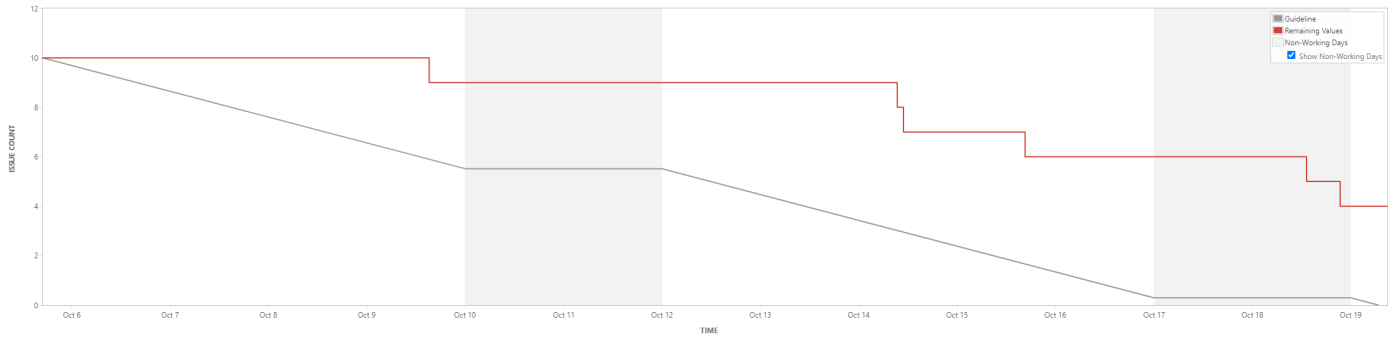


Figure D.20: Burndown chart showing issues completed throughout Sprint 3

D.5.4 Sprint 4

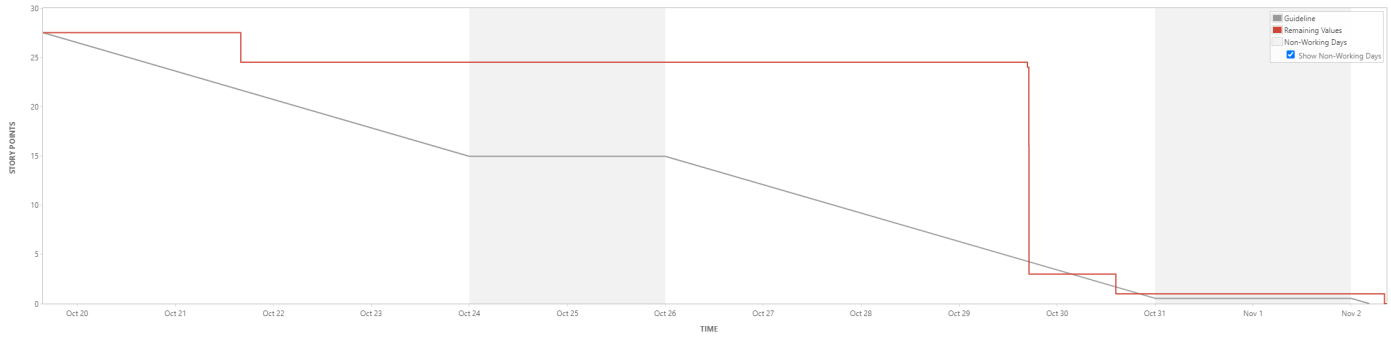


Figure D.21: Burndown chart showing Story Points completed throughout Sprint 4

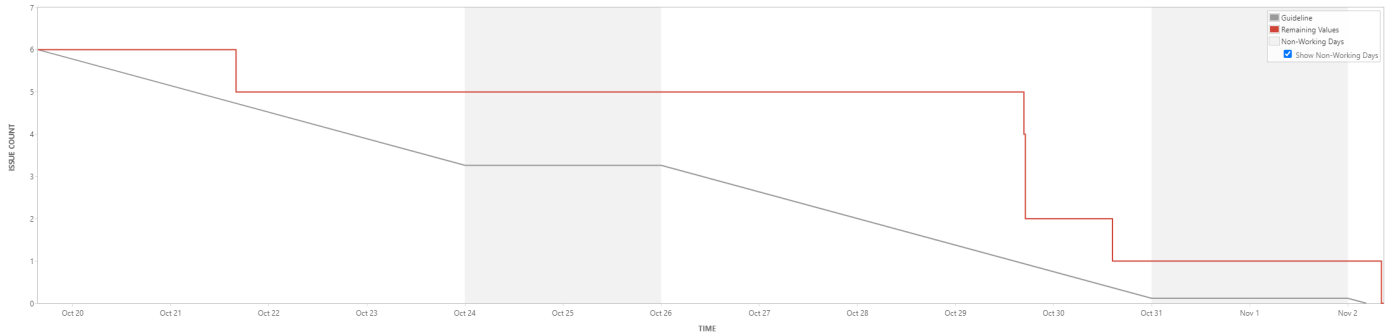


Figure D.22: Burndown chart showing issues completed throughout Sprint 4

E. Contracts and Original Documents

E.1 Original Requirement Specification

Assignment for Customer driven project

Title: FishFarmData
Customer (Company): SINTEF Ocean
Address: Sintef Sealab, Brattørkaia 17C, Trondheim

Assignment text:
In SINTEF Ocean we have been granted a research aquaculture license in order to support research and development for fish farming in realistic conditions. Currently we have four operational fish farms running off the coast of Trøndelag. Each site gathers data during experiments, which is then transmitted to our data servers at SINTEF Sealab.

The construction of the Ocean Space Center will lead to more infrastructure and sensors being added to our sites as part of the Fjord-laboratory. These sensors will be transmitting data continuously and not just during experiments to establish baseline conditions for each of the sites.

With the increase in sensors and data we see a need to restructure our dataservers. Currently, data is saved both in relational databases and large filestructures on the servers, making it challenging to find and compare old data. We are currently looking at transitioning to timeseries databases and have been investigating both commercial and open source databases with associated data ecosystems for recording, processing, and presenting data.

In this project we would like you to investigate the suitability of open source timeseries databases with associated data ecosystems, to save, process and retrieve, large amounts of sensor data. Several aspects of data value chain will be interesting and important to explore:

- Solutions and interfaces for gathering different sensor data
- Data structures in the database for different sensor types
- Integration with metadata
- Components for automatic and manual data processing
- Frameworks for data presentation.

A resulting demonstrator will need to demonstrate an overall scalable and modular architecture for a working value chain of sensordata. From the data is recorded by sensors at fish farm sites, preprocessing and saving it in appropriate data structures, automatic processing of data to detect anomalies or loss of data, as well as retrieving and presenting the data in a meaningful format.

Contact details:

Name: Finn Olav Bjørnson
Mobile: 97726490
E-mail: finn.o.bjornson@sintef.no

Figure E.1: Assignment from Sintef Ocean

E.2 Group Contract

Dato: 26.08.2020
Til stede: Alle

Gruppemedlemmer:

- Turid Cecilie Dahl
- Ivar Nordvik Myrstad
- Sander Kjetilson Kilen
- Bjørn Andre Aaslund
- Sebastian Aas
- Emanuele Caprioli
- Maria Storødegård

Regler:

- Starter alle møter med innsjekk.
    - Ikke ha PCen oppe (eller annet under innsjekk)
- Følg gruppekontrakten
- Møter har akademisk kvarter
- 15 min pause
- Si ifra i god tid om du ikke får til å fullføre oppgavene dine.
- Si ifra innen dagen før et møte om du kan komme eller ikke
    - (Med mindre det er god grunn)
- Sjekke teams 1 gang om dagen
- Utsjekk/oppsummering på mandagsmøte

Møtetider:

- Veiledningsmøte: tirsdag 12.00-13 A4-132
- Mandager 08-14 (JB10 moholt 08-16), Turid er borte 10-12
- Daily scrum på teams onsdag kl 09-10
- Torsdag 16-18
    - Diskusjon om digitalt eller fysisk taas på onsdagen før

Brudd på regler/straff:

- Kakestraff
    - Er for sent 1 gang så ta med pakke kjeks til neste møte

Avgjørelser:

1. Diskutering
2. Spør veileder
3. Flertallsavstemning

Målsetning/forventninger:

- lære hele prosessen
- lære prosess + forhold til kunde. Jobber
- Vil gjerne lære mer om backend (spennende fag)
- Fornøyd gruppe og kunde, lære så mye som mulig
- Jobber, hektisk. Håper å lære, eks om devops og server
- Se hele prosessen. Spent på å ha kunde
- gruppearbeid
- Jobber. prosessen

Innsats/tidsbruk:

24 timer i uka. Om ferdig med oppgavene sine tidlig så jobb med annet som trengs.

Roller:

Leder: Sander
- Dra på møte med referansegruppe
- Passe på booking av rom
- Sjekke at alle har noe å gjøre, eventuelt fordele nye oppgaver

Nestleder: Ivar
- Passe på booking av rom
- Hvis leder er syk/borte tar nestleder over oppgavene til leder

SCRUM Master Turid
- Organize and lead Scrum-team meetings:
    - Daily standup
    - Sprint planning
    - Sprint retrospect
    - Sprint demo
- Help the scrum-team with external resources
- Keep the sprint backlog in a good state
- Follow up on actions from retrospectives to see if they are being implemented

Kontaktperson: Adrian
- Organisere møter med kunde
- Videreformidle spørsmål og informasjon fra og til kunden

Rapportansvarlig: Maria
- Sette opp overleaf-dokumentet
- Oversikt over alt som skal gjøres i dokumentet
- Passe på at alle deler blir gjort
- Sende purremeldinger
- Få oversikt over alt som skal gjøres

Sekretær: Bjørn

- Lage møteagenda
- Skrive referat fra møter

Quality Manager: Sebastian

- Vedlikehold av kode
- Passe på at koden blir sjekket
- Konvensjoner og best practice

Test Master: Emanuele

- Sette opp framework
- Sjekke at test coverage er bra
- CI, kan være vanskelig

E.3 Group Meeting Notes - Daily Scrum

Group meeting, Customer driven project Group 6

1. **Checkin**
   (maks 1-2min pr pers)
2. **Daily scrum**
   (Skal være nokså kort, maks 20 min)
   "- What did I do yesterday that helped our team meet the sprint goal?
   - What will I do today to help our team meet the sprint goal?
   - Do I see any impediments that prevent me or our team from meeting the sprint goal? " - (tatt fra Scrum-boka fra PU)
3. **Points/Problems that need to be addressed**

Problem	**Conclusion**
Sensor diskusjon	når metadata lagres og sensor nummer ikke sendes så lages ny sensor
Opplastning av data	Laster opp data fil og config fil også fikser backend resten (antar at de har config fil lokalt på PCen sin) Kanskje skrive litt om i rapporten hvordan config fil løsningen ville vært
Hvor(i rapporten) skal jeg sette inn DDS-greiene kunden ba om. Det ble litt for langt og han ba spesifikt om en rapport på det. Kan vi dytte det i appendix, eller skal vi sende den direkte til han og legge en ekstremt kortere versjon i rapporten.	Oppsummer i further work, dytt selve rapport i appendix.
Hvor mye styr er det å endre input og options til JSON? (taas opp litt senere, og ikke relevant for alle)	Sebastian og Turid finner ut av det
Sikkerhet (Protection poker). Er det et krav å gjennomføre?	Neste mandag

4. **Agenda**

// TODO:	**Conclusion**
Jobbe :)))	

E.4 Group Meeting Notes - Review, Retrospecrive, Demo and Planning

Group meeting, Customer driven project Group 6

1. **Checkin**
   (maks 1-2min pr pers)
2. **Daily scrum**
   (Skal være nokså kort, maks 20 min)
   "- What did I do yesterday that helped our team meet the sprint goal?
   - What will I do today to help our team meet the sprint goal?
   - Do I see any impediments that prevent me or our team from meeting the sprint goal? " - (tatt fra Scrum-boka fra PU)
3. **Points/Problems that need to be addressed**

Problem	**Conclusion**
Rapport utgjør mye av karakteren	Bruke mer tid på rapport den neste sprinten

4. **Agenda:**
   - Kanskje vi burde skrive litt på rapporten denne sprinten? Tror det kan være lurt å investere en god del tid i den, siden det virker som om den utgjør mesteparten av karakteren.

Sprint Review

Tema	**Konklusjon**
Ta en runde hvor folk sier alt de har gjort/blitt ferdig med og ikke **ferdig med i sprinten + demonstrerer** det som de har gjort (3min ca pr pers)	X
Gå gjennom stories/issues i **Active Sprints** i Jira	X
Oppdater product backlog	X

Retrospektiv

Tema	**Konklusjon**
The Scrum master shows the sprint backlog and, with help from the team, **summarizes the sprint. Important events and decisions**, etc	X
What went well?	• Forstått mer av veileder • **Bra oppmøte (4)**

	• **God stemning (4)** • Oppdatering av jira • Bra å få mer detaljer fra kunden • **Fått gjort mye (2)** • Good structure on the meetings • god oversikt av løsningen • **god insats/arbeismoral (7)** • forståelse for at ting tar tid • **enkelt å få hjelp (2)** • good communication • **gode code reviews (4)** • effektivt arbeid • kjeks • tydeligere issues • getting into coding • **good workfloow/fleksibilitet (5)**

What needs improvement?	• **kommunikasjon/si ifra om du jobber med stuff som påvirker andre (4)**
	  ○ 6
	• **mer effektive møter (2)**
	  ○ 4
	• Dårlig stemning på møtene
	  ○ 7
	• **ventetid på pr (2)**
	  ○ 2
	• tatt på oss for mye
	  ○ 2
	• **better estimering (6)**
	  ○ 1
	• lite mulighet til å diskutere løsninger
	  ○ 1
	• kode modifiability
	  ○ 1
	• corona
	• **forsentkomming (3)**
	• **fysisk oppmøte (mandag) (2)**

Diskuter forslag til **konkrete ting for å gjøre neste sprint bedre** (It is important not to get overambitious here. Focus on just a few improvements per sprint.)	• Før tema droppes, spør om relevant for alle. Hvis "ja" så droppes det ikke • Send melding før du starter på issue som der er påvirkning på andre • Være mer tålmodige • Oppsummere kundemøtet for resten senest torsdag • Skriv opp diskusjonspunkter på agenda

Sprint 4 Planning meeting

(Endre sprint navn roe -> smolt -> salmon)

Tema	Konklusjon
Sette **sprint goal** (“Why are we doing this sprint?”)	“Make the product pretty and presentable and limit functionality to only the most important parts. The functionality that is there should work well.”
Sette **hvor lenge** sprinten skal vare	2 uker (til og med fredag 30.10)
Sette **demo tidspunkt**	man 2. nov (spør om ok for kunde)
Se på de med høyest importance og estimer story points, **Planning poker**	X
Team **selects stories** to be included in sprint. Do velocity calculations as a reality check.	X
Further breakdown of stories into **sub tasks**.	X
Fordel stories til hvem som har ansvar/skal gjennomføre dem	X

Demo

Tema	Konklusjon
1. Gi kort oppsummering av hvordan sprinten gikk	Ferdig med sprint nummer 3 (utviklingssprint 2) Ikke alt som har blitt ferdig.
2. Vis hvilke stories som ble ferdig og hvilke ble ikke ferdig	6 issuer ble fullført ● Ny metadata ● Autentisering i backend ○ På query-nivå (for hvert endepunkt) ● Visuell prototype ● Brukertesting ● Optimalisering av grafer (få bort lag, og lage grafene finere) ● Microsoft log in 4 issuer ble ikke fullført (små mangler) ● Lagre og endre metadata (mangler kobling mellom frontend og backend) ● Lage dashboards og cells ● Subscribe ○ Går over til graphql fra DDS ● Laste opp filer manuelt (mangler

	kobling mellom frontend og backend)
3. Gå gjennom demo og demonstrer det som er resultatet av de ferdige bruker	Turid og Sander viste frem det vi har gjort
4. Gjennomgang av prototype	Viste frem det nye **Kunde**: En celle der man skriver tekst er ikke så viktig så lenge man har en description av dashboardet. ● For enklere brukeropplevelse ● description knyttet til en spesifikk celle, som vises ved siden av cellen i dashboardet. Må huske å legge til muligheten for forsøk Endre sensortypes til sensor ID Share - dashboards -> groups?
5. Gjennomgang av neste sprint	Neste sprint er godkjent ● Implementere ferdig det som er påbegynt ● Appendiks som forklarer ulike løsninger beskrevet under.
6. Spør om tanker fra kunden a. Fornøyd? b. Andre kommentarer?	Fornøyd Ingen andre kommentarer
7. Neste demo	Mandag 02.11, kl 12:00
Nevn at vi fokuserer mer på rapport denne sprinten også	
Spør om eksisterende løsning til pre-study kapittelet	Neste demo skal Finn gå gjennom eksisterende løsning.

Kommentarer fra kunden:
Hvor mange timer tar det å utvikle fullverdig brukerstyring?
- Overslag i en av de kommende sprintene
- Estimat av hvor lang tid det tar å implementere hele kravspecen? Estimert timesverk for en junior utvikler (konsulent). Fra det utgangspunktet vi har kommet til

Skriver i rapporten om DDS, og hvorfor vi gikk over fra DDS
Kjør det lokalt hvis det funker bedre for oss. Trenger ikke å bruke tid på å få det til å funke i prod.
Kunden er interessert i dette. Ha med i Appendix eller noe sånt..?
- hvordan løser dere selve tidsserien kobla mot metadataene i den nye databasestrukturen? databasevalg: hvorfor og hvordan
- problematikk rundt DDS, hva vi ender på på websockets

- autentisering, brukerhåndtering (hvis vi har tid). Hvordan ser vi for oss veien fremover til et fullverdig autentiseringssystem? Koble det til databasevalget.
- generell dokumentasjon av hvordan strukturen er laget for presentasjonslaget (dashboards)

Han nevnte også at (hvis de får finansiering for det) kan det være interessant med deltidsjobb på våren / sommerjobb for en liknende løsning.

5. **Until next time (?)**

Adrian -
Sander -
Bjørn -
Sebastian -
Ivar -
Emanuele -
Maria -
Turid -

E.5 Supervisor Meeting

Supervisor meeting Customer Driven Project Group 6

Group members present:
(Write a X if you are present)

Name	Present
Sander	x
Ivar	x
Turid	x
Adrian	x
Maria	x
Bjørn	x
Sebastian	x
Emanuele	x

1. **Work done since last meeting**
    a. Finished a draft of the report
2. **Problems encountered**
    a. In Appendix A - Internal and External Documentation, we have a sub section for back end documentation. This sub section is becoming rather large. Should we create an own appendix for back end documentation or should we keep it in appendix a?
        i. Whatever pages you have is ok. It is also possible to have a link to the web drive where we store these pages. It is ok to keep the 20 pages here.
    b. Where should we include the link to our github repo?
        i. Put it in the introduction part. In the first one or two paragraphs. (Or in the paragraph that you introduce the task)
    c. How short do you think Architectural Rationale should be?
        i. (We removed a paragraph) - good
    d. Should we include some charts showing how much time we have used on the different sprints in the implementation chapter?
        i. Time/effort registration: put it in the appendix and refer to it in the planning section.
        ii. Burndown charts should be referred to from the implementation chapter. The graph can be in the appendix
    e. Our report is currently 52 pages. Is this ok since we have a lot of figures?
3. **Planning of work for the next period**
    a. Finishing the report
4. **Other issues (if there are any)**

1

Mail til veileder:

- suggestion for the presentation: prepare for questions such as which difficulties we faced and how we solved them.

The report seems good, but these things should be fixed:

- font size should be changed from 10 to 11.
- If we need to decrease the page count, we should decrease spacing
- use longtables for the tables to make them cross pages.
- the gantt graph should be in the planning section, not in the appendix.
- the reviewers will not look carefully at the appendix.
- We should add examples of the notes from customer meetings, group meetings and weekly status report to the appendix. Refer to these documents in the section 3.3 Meeting Plans where the meetings are described.
- the back end documentation can remain in appendix A, as it is now.
- The report should not be longer than 62 pages. This includes the evaluation chapter, but not the references.
- The risk management figure needs to be wider to improve readability
- The tables in section 6.2 should not be wider than the text.
- The link to our code should be in the introduction chapter of our task. (chapter 2.2)
- The length of the architectural rationale is good.
- Burndown charts should be added to the appendix and referred to from the implementation chapter.

Svar fra veileder:

Yujie Xing
to me

Yes that is correct! Burndown charts: if you have extra spaces, you can include it in the implementation chapter.

2

E.6 Customer Meeting Notes

Group meeting, Customer driven project Group 6

Sprint 3 Planning meeting

(Endre sprint navn roe -> smolt -> salmon)

Tema	**Konklusjon**
Sette **sprint goal** (“Why are we doing this sprint?”)	Få gjennomført hovedtrekkene av produktet angående data flow.
Sette **hvor lenge** sprinten skal vare	2 uker
Sette **demo tidspunkt**	(2 uker) mandag 19. okt
Se på de med høyest importance og estimer story points, **Planning poker**	X
Team **selects stories** to be included in sprint. Do velocity calculations as a reality check.	X
Further breakdown of stories into **sub tasks**.	X
Fordel stories til hvem som har ansvar/skal gjennomføre dem	

Demo

Tema	**Konklusjon**
1. Gi kort oppsummering av hvordan sprinten gikk	• Går gjennom sprint 2 på Jira
2. Vis hvilke stories som ble ferdig og hvilke ble ikke ferdig	• Noe har blitt forskjøvet til neste sprint
3. Gå gjennom dem og demonstrer det som er resultatet av de ferdige bruker	• Innlogging • 2 grupper -> researcher og engineer • Viser data som graf ○ Data hentes fra sql kall • Kan legge til flere grafer • Frontend delen av å laste inn filer • Backend delen som har blitt gjort vises implisitt i gjennomgang av frontend • Skal også kunne legge til konkrete

	e-poster ○ Eks: SINTEF brukere

4. Gjennomgang av prototype	• Ikke så viktig om vi sender ut mail
	• Kan poppe opp en e-post adresse til finn, så man har mulighet til å sende en mail
	• Private dashboard?
	  ○ Vil være private i utgangspunktet, og skal kunne dele det etterpå
	  ○ Forskere vil være mest interessert i private
	  ○ Ikke så viktig å rangere dashboard
	  ○ Hvis de private blir delt, så havner det i delt
	• Add cell delen ser bra ut
	• Endre størrelse på celler?
	  ○ Nice to have, but not a must have
	• Hva skal lastes ned? (data/grafer)
	  ○ Underliggende data er viktigere enn grafen
	  ○ Fra en serie
	  ○ Skal kunne laste ned alle dataene fra dashboard som blir vist. Men det viktigste er å kunne laste ned en og en graf
	  ○ Zippe alt sammen? Kan prøve
	  ○ csv-format
	• Skal kunne legge til tekstboks som en celle
	  ○ En tekstboks som forklarer grafen/cellen
	• Add cell
	  ○ Select time -> mulighet til å si fra nå og hvor lang tid tilbake i tid
	  ○ Eks: nåværende dato og 3 dager tilbake i tid
	  ○ Skal kunne velge ned på klokkeslett, kanskje også minutt
	  ○ Filtrere select fil: slik at man bare har relevante felter her
	• Kan ta utgangspunkt i de filene vil har
	• Hvis man har tilgang til optode serien har man tilgang til alle seriene i filen
	• Gjesteforskere vil ha tilgang til en optode serie i et bestemt tidsrom
	  ○ Eks: skal bare ha optode dataen i juni
	  ○ Viktigst å skrive om det i rapporten
	• Mer kompleks data -> mulig å legge til andre grafer, for eksempel heatmap
	• Admin side
	  ○ Ta inn data manuelt
	  ○ De forskerne som trenger å legge inn data kan få ingeniør rolle i tillegg

	• Kan ha en personlig gruppe for sensorer som man legger til en sensor • Skal det være forskjellig type tilgang ○ Begrense det til ingeniørrollen, og gi forskere ingeniørrollen hvis de trenger å legge til sensorer i sensorgrupper • Legge de fleste adminoppgavene i ingeniørrollen • Select measurement -> select o2 i stedet for • Select fields -> select sensor • Søker på metadata før man kommer til add cell siden • Eks: Trondheim -> mære 2 -> optode 1 og 5 •

5. Gjennomgang av neste sprint	• Hovedmål: Få gjennomført hovedtrekkene av produktet angående data flow
	• FFD-87: fra forrige sprint
	• FFD-37: web socket som åpnes hos clienten -> finnes i graphql
	  ○ Sjekke ut data distribution services -> standard fra OMG
	  ○ Sjekke ut hvor mye vanskeligere det er å implementere dette
	• Legge til en task på å søke på metadata
	• Begrense at ikke alle kan hente data med Microsoft

6. Gjennomgang av metadata	• Location
	  ○ Plassert på oppdrettsanlegg med navn med gps koordinat (kanskje link til bilde)
	  ○ På hvilken del av anlegget
	  ○ På hvilken dybde
	  ○ Location må kanskje deles opp mer
	  ○ Ha det på koordinat og dybdenivå, og kanskje tid
	  ■ En lokasjon kan endre seg over tid
	  ○ Skal kunne gjøre et romlig søk
	  ■ Hvilke sensorer er innenfor et område rundt en sensor
	  ○ Gjør en forenkling!
	• Kan ha en link mellom sensorer som var med på det samme forsøket
	• Log -> er greit å ha med
	  ○ Eks: tatt opp og renset for 1 måned siden
	• Flere personer kan gå inn og oppdatere metadata (dropp sensor_responsable tabell, alle ingeniører kan det)

	• Beholde ER-diagrammet som det er, men bruk den tilgangsstyringen som vi diskuterte • Forenkle metadata feltene (companies og relaterte, department) ○ Må ha id og navn

7. Spør om tanker fra kunden
  a. Fornøyd?
  b. Andre kommentarer?
	• Ser bra ut, bare å jobbe på i neste sprint!

8. Neste demo	• Mandag 19. okt kl 14:30 • Sender mail hvis det ikke passer
Hvordan vil kunden at siden hvor man legger inn ny meta data og ny sensor se ut?	Sensor = 1 oksigen måler for eks ikke 1 fil = 1 sensor (Men kommer an på fil til fil) (optode = samling med sensorer) • Forsøk = dashboard? Sannsynligvis ett eller flere dashboard pr forsøk (kan ha lyst på sensorer på tvers av forsøk også) Begrense på forsøksnivå (?) • Hva skal være på upload data manually? hvilke filer config av filene, eks 3 temp, 1 oksygen ect (Velg spesifikk sensor fra dropdown meny om eksisterer) (Timestamp som første automatisk?) (Anta at du har sensor) (Kunne lagre oppsett, np kommer fil som er helt lik den forgje jeg la inn, er nice to have not a must) (Heller ha kun konkret sensorid? O2noe) Konfigurasjonsfil valg! • Hva skal være på new sensor Nei • Hva skal være på new metadata/update metadata? navn, number er et must. sensor dannes når man lager metadata. Dashboard tilgang vil si tilgang til sensordataene på den? Svar: Ja. Her kan de få tilgang