Group Name: /*TODO*/	
Surname	First Name
Haraldseth	Ole
Meltveit	Fredrik
Mossestad	Daniel
Larsen	Aleksander

Course code	IS-304
Course name	Bachelor Thesis in Information Systems
Course responsible	Hallgeir Nilsen
Supervisor	Janis Gailis
Deadline	May 14th, 2021
Number of words (whole document)	16581
Project/product title	Wizards - a Simplified Case Management Application

We confirm that we do not cite or otherwise use other people's work without this being stated and that all references are listed.	Yes X	No

Can the report be used for teaching purposes?	Yes X	No

We confirm that everyone in the group has contributed to the report	Yes X	No

Report IS-304

Wizards - a Simplified Case Management Application

University of Agder
Institute of Information Technology
Group 5, /*TODO*/

Preface

This report will present, discuss, and reflect on the group project in course IS-304 Bachelor Thesis in Information Systems. The course is the final project for the team's bachelor's degree.

Special thanks to Fredrik Werpen (product owner), Marius Holen, and Merethe Sjøberg for their support and commitment to this project. Many thanks to the team at Sikri for allowing us to do the project with them and for superior guidance throughout the project.

Thanks to Janis Gailis for honest and direct feedback and input on the final report.

Kristiansand, University of Agder, Spring of 2021

The members of the team and the authors of this report are:

Ole Haraldseth: ole.haraldseth@gmail.com
Aleksander Larsen: aleksanderbl60@gmail.com
Fredrik Meltveit: melt.fred@gmail.com
Daniel Mossestad: daniel-mossestad@hotmail.com

Abstract

The project was offered by Sikri on the basis of an earlier research report. This report had documented some of the already known problems with their main application Elements and how they could be solved. Elements is a case management program used to handle case management in the public sector.

In the report, we learned that many of the users of Elements are uncomfortable with the design weaknesses in Elements. There are too many features for infrequent users, and they struggle to manage basic case management. This had resulted in the need for a lot of guidance and frequent training for these users. From the report it was concluded that Sikri needed a lightweight alternative with a simple and modern UI for these users. This lightweight alternative should be based on a step-by-step process, a wizard, to guide the users through a case. This would mitigate the need to spend extra time on training and support by administrators.

The bachelor group was tasked with creating this new lightweight application. The aim of the project was to create the basic design and functionality for the application, called Wizards.

The project was managed using the Scrum framework in combination with Software Development Life Cycle for development of the application. The project consisted of nine sprints divided into a two week cyclus.

The most central technologies used to construct the Wizards application was React with TypeScript for the Frontend, Redux for state management, Material UI and Material Design for the design of the application, and the Backend is managed with the help of OData API for connection with the NCore server.

The result of the bachelor project was a finished application with connection to the same Backend as Elements, with the basic functionality needed to process a case in the application. The application was also deployed at the end of the project and available for everyone involved to test and use. Sikri will continue to develop the Wizards application and is aiming to offer the application to its customers in the fall.

Table of Contents

1 - Introduction

The group /*TODO*/ has, during the spring semester 2021, from January to mid-May, written their bachelor project for Sikri AS. The group consists of four members.

Sikri was formed in January 2020, after it was separated from the IT company EVRY. The company has more than 100 employees. Elements is a case management system developed, used, and sold by Sikri. It is primarily used in the public sector in Norway.

The team was given the task of designing and developing a web application with the goal of simplifying Elements. The application will first and foremost be targeted against infrequent users who do not work with case management on a daily basis.

The project is managed through the use of the Scrum framework, and we have chosen to use an agile methodology to be ready for unforeseen events. A document with an analysis of the weaknesses of Elements was given to us at the start of the project and worked as guidance for how the Wizards application would end up. Since the analysis was already done by Sikri, our project was focusing on design, Frontend development, and connection to the Backend core of Elements.

The objective of this report is to document the process of developing the Wizards application and all the different steps made to ensure an end product of high quality. It also covers the experiences and learning outcomes gained by the group throughout the project.

1.1 Definitions

Elements: The case processing and archiving solution built by Sikri.
https://www.sikri.no/elements-sak-arkiv

NCore: The connection point to Elements, the NCore server is a backend environment made with the .NET Core framework, which is typically written in C#.

Workflow: An hierarchical collection of tasks defined in Elements.

Wizard: The specific user interface for workflow-based case processing (Tech Terms, n.d.) (As opposed to Wizards, the project).

Wizards A, Wizards B, Wizards C: The state of the application at the end of sprints 1-3, 4-6, and 7-9, respectively.

2 - Product

The product is a web application for simplified case processing. Sikri develops and delivers a product with many excellent and advanced functions that can be used for most case handling tasks that are in demand in today's market. The product is called Elements, and through a customer survey, Sikri has found that a lot of their users do not use Elements on a daily basis. This results in the users being in need of help and guidance when they use the system, which increases the workload of superusers because they have to take time out of their day to support these types of users. Through the survey, Sikri found that there is a need for a more simple way to handle these kinds of users. This need creates the basis for our bachelor project, which is to create a product with a simple user interface that makes case processing easy, seamless, and intuitive. The idea behind the simplified application was to create a wizard for case management: a step-by-step interface to guide less experienced or advanced users through case processing. This led to the working project name "Wizards."

This chapter will go into depth on what the Wizards product is and what its current state is.

2.1 Product Description

The Wizards web application is a supplement to Elements, connected to the same backend. Wizards is intended as a simplified alternative for case processing of cases that have a defined workflow. Workflows are definable in Elements and can be connected onto an individual case or a case type in order to provide a set of tasks for the completion of case processing.

All cases belonging to a caseworker can be displayed in the Wizards application. However, the Wizards only allows case processing through a wizard, and the wizard

interface is built from defined workflows, so cases that are not reducible to a workflow can not be processed. Because of this, the wizard does not have to display functions, features, and buttons that are not relevant to the task at hand.

The web application is divided into two primary sites: a "cases page" page for browsing cases and a "wizard page" for processing a selected case. The pages can be accessed after a caseworker logs in using their Elements account.

The cases page renders a list of all cases connected to the logged-in user in Elements. A search bar allows the user to filter cases based on search queries. The list is sortable based on all relevant case fields, and the number of cases viewed at once can be modified between 5-20. Once a case is selected from the list by the user, the application renders a view for case details to the right of the list. The application automatically resizes components to fit the number of components displayed at once. In the case details view, the user can select the different journal entries of the case for further inspection. After selecting a journal entry, the user can view the entry details, preview linked documents, and see the tasks required to process the case.

The wizard page allows the user to begin processing work for the selected case. The page replaces the case list and search bar with a Wizard interface while keeping the details view of the selected case. The details view allows the user access to all relevant case details and documents while working through the tasks for the wizard.

2.2 Product Demo

This link redirects to a video of how Elements and Wizards works and looks:
https://drive.google.com/drive/folders/1RgZGtG8OlWFl4luWOtMgAW-vTEl6L32b?usp=sharing
This link redirects to an early clickable version of the Wizards (version A) application from sprint 3:
http://wizards-env.eba-z9wcmrpk.us-east-2.elasticbeanstalk.com/Dashboard

3 - Project Decisions

In software development projects, "managers are saddled with the responsibility of leading their organizations to achieve objectives and stated goals" (Abubakar et al., 2019). This applies both in terms of how the team works internally and also in terms of the external collaboration with another party. Further in this chapter, we will describe and discuss central decisions throughout the project.

3.1 Project Management

In order for a project to take place in a smooth manner, it is important to make a decision on what the workflow will look like. In the initial phase of the project, there were many discussions regarding the choice of tools and working methods.

3.1.1 Project Framework - Scrum

We chose Scrum as our project framework because; it was the project framework we had the most experience with, and it was recommended by Sikri. The stakeholders in Sikri use this type of framework in their own software development projects and would therefore be able to offer better assistance.

The Scrum Team consists of the Product Owner, Scrum Master, and the development team. The Product Owner has the main responsibility for how the product will be at the end of the project. It is the Product Owner's responsibility to make the central decisions, while the Scrum Master must ensure that the demands made by the Product Owner are fulfilled. There is a lot of organization in the work tasks of the Scrum Master. Sprint Planning meetings, Sprint Retrospective meetings, and Sprint Review meetings must be organized and led. It will also be the role of the Scrum Master to lead stand-up meetings at the beginning of each working day. "The Product Backlog is an emergent, ordered list of what is needed to improve the product." This list is the only source of work to be done by the Scrum Team. The Sprint Backlog consists of a list of Sprint Backlog items. These items each contain a Sprint Goal, a Product Backlog item, and an actionable plan for delivering the Increment. (Scrum.org, n.d.).

The roles of the Scrum was delegated in the following way:

- Product Owner - Fredrik Werpen
- Scrum Master - Ole Haraldseth
- Development Team - Ole Haraldseth, Aleksander Larsen, Fredrik Meltveit and Daniel Mossestad

3.1.1.1 Agile Software Development

To be agile was important throughout the project and was adopted early. According to The Manifesto for Agile Software Development, there are twelve important agile principles in software development, some of these are: deliver working software frequently, welcome changes in requirements, frequent face to face conversations with customers and end-users, reflecting on how to improve and become more efficient (Beck et al., 2001).

It was decided to keep the sprints short and to have frequent contact with stakeholders. This was decided so the stakeholders could affect and influence the work as much as possible as well as come with inputs on potential changes to the scope. This way, potential misunderstandings or complications could be solved early.

3.1.2 Project Management Tools & Version Control

To ensure a successful project, it is important to keep track of all the tasks to be done, but also to keep track of what each individual should do. This chapter will describe which tools we chose to use, as well as a short brief about why we chose to use them.

3.1.2.1 Azure DevOps

The product management software we chose to use was Azure DevOps. Sikri uses it in its daily operations, and it was highly recommended by them.

The program was a great tool to help organize the Scrum and keep track of the Product Backlog (appendix 1), Sprint Backlog (appendix 2), and tasks (appendix 3). The tasks were organized in sprints and could be carried over to the next sprint if not all the planned tasks were completed. This means that you can keep a good overview of the tasks, and you can also link the various tasks to the person who

should do a given task. Another great advantage of DevOps is that you can write a time estimate on the various tasks.

3.1.2.2 Version Control - Git

After some discussion and advice from Sikri, the group decided to use Git (Git, n.d.) for version control. Git is something the group members have worked with before, and it is most likely the most widely used version control tool in software development (RhodeCode, n.d.). In addition, Azure DevOps has a close integration with Git. These factors combined made us agree that Git was the ideal tool for our project.

The way we used Git was that we defined user stories before we divided them into several smaller tasks. The various tasks should be short, preferably no longer than what can be completed in a day. Short tasks and proper use of Git will lead to high quality and easy to revert changes.

To assure high quality and proper use of Git, a document (appendix 4) was created for how Git should be used so that it was done correctly by all members. A positive consequence of using strict rules when working with Git is that there will be a standard for the programmers from Sikri when working with the code in the future.

3.1.3 System Development life Cycle (SDLC)

SDLC is a process that can be used when developing software and information systems. We decided to use this process along with Scrum to develop the product. This means that the development would happen in stages consisting of planning, analysis, design, testing, and implementation. These stages will be described further in chapter 5 and 6 (Wikipedia contributors, 2021a).

3.2 Technology Stack

This chapter will describe the technology stack we have used to create the product and some information about the choices we have made.

Technology Stack used:

- **JavaScript**, with **TypeScript** superset, as the programming language

- **React** as Frontend framework
- **Redux** for global state management
- **Material UI** for styling and design components
- **OData** for **API** connection to the **core** server

3.2.1 JavaScript, TypeScript & React

As Sikri has recently converted their entire software into React with TypeScript, it was a requirement from Sikri that this was also used in the new software so that it could easily be maintained by Sikri after the project is completed.

React is an open-source, Frontend **JavaScript** library for building user interfaces or user interface components. React can be used as a base in the development of single-page applications. It is maintained by Facebook and a community of individual developers and companies. Additional libraries for state management and routing are usually used together with React. Redux and React Router are examples of such libraries (Wikipedia contributors, 2021b).

TypeScript is an open-source language that builds on JavaScript. It adds static type definitions. Types provide a way to describe the shape of an object, providing better documentation, and allowing TypeScript to validate that your code is working properly (TypeScript, n.d.).

3.2.2 Redux

Redux is an open-source JavaScript library for managing application states. It is commonly used with libraries like React for building user interfaces (Wikipedia contributors, 2021c). It is a predictable state container for JavaScript applications. Redux helps with writing applications that behave consistently and run in different environments (client, server, and native). It works with any UI layer and has a large ecosystem of add-ons (Redux, n.d.).

The decision to use Redux was based on a desire to update and access the state from anywhere in the application. Keeping a global state for features such as multiple cases, a selected case, search queries, and selected registry entries. This allows a user to navigate anywhere in the application while maintaining a consistent state. Redux was chosen above other state management frameworks because (1)

Redux is the most developed framework, (2) Redux provides rigorous standards and documentation, and (3) Redux has a large number of optional resources and packages. As the project would at some point be connected to the Elements backend, the state would have to be updated based on API calls. In other frameworks, and in plain Redux, you can generally only call synchronous updates to the state. The thunk middleware package allowed for readily made standards and functions in regards to asynchronous updating of the state, which is often necessary functionality when making API calls.

Later in the project, the Redux Toolkit was added.

One drawback of Redux is that, if following Redux' TypeScript guidelines, the written code adds up to a lot of boilerplate while introducing coupling issues; a small change might require changes in up to four different files and functions. As this could build up to an annoyance, or a general issue in the Wizards project, we chose to add the Redux toolkit.

Redux Toolkit is a toolset that helps shorten Redux-related code and reduces code length and coupling issues. In addition to providing good defaults for store setup, Redux Toolkit includes the most commonly used Redux addons built-in (Abramov & Redux documentation authors, n.d.-a).

3.2.3 Material UI

For the design of the different components, the team decided to use Material UI. Material UI provides pre-made React components that can be used for faster and easier web development. Material UI is one of the most popular UI frameworks and is based on the "material design" standard developed by Google (Material-UI, n.d.). It provides more ready-made components than most other libraries, and some members of the group had used the framework in earlier projects. The Material UI components can easily be customized to fit a certain specific need for a specific scenario. This meant that it could easily be customized to fit the wizard and the design language the team was going for. That also meant that time was saved by not having to design components from scratch and helped the project progress faster.

3.2.3.1 Material Design

Material design is a design language developed by Google in 2014. Some of the main principles in material design are the use of grid-based layouts, responsive animations, padding, depth effects as lighting and shadows (Wikipedia contributors, 2021d).

Google describes Material Design as:
"Material is an adaptable system of guidelines, components, and tools that support the best practices of user interface design. Backed by open-source code, Material streamlines collaboration between designers and developers and helps teams quickly build beautiful products (Material Design, n.d.).

Material design is used by several of the biggest companies and social platforms in the world, such as Google and their entire ecosystem, as well as NASA, Netflix, Spotify, and Amazon. (Material-UI, n.d.-b)
The group chose to follow Google's material standards, as it would give users a familiar design as well as a modern, sleek and intuitive feel. Another important factor was that the whole team had previous experience with this library and had used it in an earlier project.

3.2.4 OData and API Connection

The OData protocol was used for communication between the NCore servers and the Wizards application. This was given to us by a resource from Sikri to help us with API connections. OData protocol is a standard way to use RESTful APIs in a simple and standard way (OData, n.d.). For connection purposes, we were provided with some custom-made services and schema files to easily integrate the Wizard application with the Elements ecosystem.

3.2.5 Code Standard and Folder Structures

It was decided early on that we wanted to have a standard on how code should be written and how the documents should be organized. Previous experiences from projects have given an increased focus on a common standard, making the work more agile and organized. It would be beneficial for the group members to have standard code language to understand each other, and it will be essential for Sikri that the code written follows a certain template. This will make the handover

smoother, and the developers who will continue to work with the product will have one standard to adhere to. This will help to strengthen the quality of the product that is delivered, which is a consistent focus area. The code standard can be found in appendix 5.

3.2.5.1 Project Folder Structure

For storing, organizing, and writing documents, Google Drive has been used as a platform. Here you can save files and documents as well as create and edit text documents, spreadsheets, and presentations from Google Docs directly in the storage medium. Another great advantage of Google Drive and Docs is that you can edit documents in real-time so that team members can see all the changes that are made to the same document. This makes the collaboration seamless when working from different geographical areas.

3.2.5.2 Code Folder Structure

The initial folder structure was based on the "grouping by file type" example from the React file structure FAQ (Abramov & Redux documentation authors, n.d.-b). Some folders were added on the initial project setup, for example, a "components" folder. Others were added later once the related code had begun development (e.g., the service and data folders). The parts of the file structure added and in use for the Wizards A application was:

- **/components**
  The components folder contains all react components that are not parent components for a screen (explained below).
- **/helpers**
  The helper folder contains files for generating data for the Redux store. All data shown in the initial application are generated through these files, allowing for a dynamic prototype with newly generated data upon startup.
- **/screen**
  The screen folder is used to hold parent components mapped to the different URLs of the application. For example, the "/Cases" URL linked to a CasesScreen component, and the "/Wizard" URL linked to a WizardScreen component.
- **/store**
  The store folder is used for all Redux-related code. It contains folders for each slice of the global case, with each containing files for types, reducers, and

actions, and helper files if necessary. This structure was based on the design patterns from Redux's "Usage with TypeScript" documentation.

In addition, there were folders for fonts, images, and styles.

In the period of sprints 4-6, in Wizard B, additional folders were added for services and OData. These folders were used for Backend connection and model schemas, as well as authentication of requests to the core layer.

Sprint 7-9 saw the project refactored, with Redux and component code reduced to a feature-type folder structure. In short, this grouped related components with one file for logic concerning the related store slice. A common folder was added to hold components that were reused throughout the application and not related to one specific feature (e.g., the sidebar). The components and store folders, as well as their subfolders, were cut in favor of the new structure.

3.3 Time Management

At the beginning of the project, a document was created and has been used to track how much all group members work every single day (appendix 6). In this way, it is ensured that everyone puts in the same amount of work that has been agreed on in the group contract found in appendix 7.

In addition to an overall time registration, we have used Azure DevOps as a time estimate in relation to the specific tasks in the coding part of the project. The reason for having an overall time registration outside of Azure DevOps was because large parts of this project included administrative tasks, and we did not want them to be mixed into the development tasks. The time management document can be found in appendix 6.

3.4 Work Location

It has been a very unusual semester because of Covid-19. Because of all the restrictions and needs to social-distance, it has been almost impossible to sit physically together to work on the project. This applies to both work within the group, but also physical cooperation with Sikri. For communication within the group,

we decided to use Discord throughout the project. All in-group communication took place on Discord, while all meetings with Sikri were conducted through Microsoft Teams. We were familiar with Discord from our previous school work, and we learned how to utilize Teams when working with Sikri.

4 - Sprints

In this chapter, we describe the different sprints and the sequence of events happening in each sprint. The project was divided into two-week sprints, which resulted in nine sprints. See the Gantt chart in appendix 8 for an overview of the sprints.

4.1 Pre-sprint

The pre-sprint was the period between our first meeting with Sikri until the start of our first sprint. We had multiple meetings with Sikri before starting the project properly, where we talked about Sikri's goals for the project. We learned that Sikri wanted a more user-friendly experience for none-frequent users. This was presented as difficult due to the application Elements had grown severely in features and time-to-learn. In this regard, the Wizards project could help, as the aim was to create a smaller and less difficult platform that could be picked up and used practically in a smaller amount of time, with an easier learning curve for new users. In addition to the project description and discussions, Sikri informed us about the technologies to be used in the project, such as React with TypeScript for Frontend development and .NET for backend development.
In the pre-sprint, we also set some goals for the early sprints: to develop user stories to define the MVP (minimum viable product) and to finish a first design draft. We decided on a sprint length of two weeks, as suggested by Sikri, as this would allow us to be more agile and to work closer with Sikri. We also hoped that changes in requirements would happen slower than every two weeks, and it gave the stakeholders the opportunity to wait and see what we had done to the end of the sprint before making any changes (Rawsthorne, 2020). For the early sprints, we decided to focus on the definition of user stories for an MVP, including the definition and development of acceptance tests, as well as on design.

4.2 Sprint 1-3 - Wizard A

In this period, we defined and redefined user stories multiple times, finished early sketching and design, and developed an application (Wizards A) backed by program-generated data. In this period, we were in daily dialogue with our project owner, as well as developers at Sikri, in an attempt to plan the eventual connection of the project to the NCore servers (the Elements Backend).

4.2.1 Sprint 1

The goals for sprint 1, was to begin the process of developing user stories, sketches, and design, to draft an MVP specification, to define a backlog, and to begin learning the Azure DevOps platform.
The sprint was used to develop and redevelop the first version of user stories, as well as an early design, which provided a foundation for further work. On the last day of the sprint, the programming project was initialized; to be developed according to the continually developed user stories and design.

4.2.2 Sprint 2

The goals for sprint 2 were generally to continue the development on all fronts: user stories, design, and to transform the user stories and design into a prototype or demo application. The project was also added to the Azure DevOps platform in order to connect the produced code to defined tasks and to manage Git branches. Work tasks would be divided between group members, such as project management and further work in defining user stories, design, and development. Meeting frequency with Sikri increased, and the first week of the sprint would include four separate meetings with the project management team, as well as developers in Sikri. A conversation was started in regards to how the project would, eventually, connect to the Elements NCore Service. A lot of discussions and ideas were created by both the group and the team at Sikri, some of which made it into future sprints and development. The decision that a case should be possible to process in both the Wizards system, as well as Elements, was made by the project management at Sikri.
An analysis of the structure of case processing in the Elements system was done, presenting the hierarchy and structure of cases, registry entries, and documents to

the group. It was decided together with Sikri to focus on a specific type of case to make sure that the project was focused on quality instead of quantity.
In this sprint, we also presented our design suggestion for the application, and the product owner decided that they preferred this design over the existing design in Elements and recommended that we moved forward with our suggestion.

4.2.3 Sprint 3

The Product Owner instructed us to reduce incoming application forms to schemas (skjema.no), putting aside other forms of cases involving other documents.
The goals for sprint 3 were to design and begin the development of the wizard page of the application, to be used in the processing of cases defined in incoming application forms. The incoming application forms were to be processed as defined schemas, following a standardized XML or JSON format.

Based on new information regarding cases to be processed and the structure of these incoming application forms, a wizard was designed with a JSON schema structure in mind. Folders were created for building any potential wizards or schemas by processing the incoming schemas and rendering the necessary components.

4.3 Sprint 4-6 - Wizard B

This period was the start of the API connections to Elements Backend. It was also the sprint where we focused on getting all the group members to get an understanding of the code written till this point. Pair programming was introduced here to help the members get an understanding of the different components and help with further development.

4.3.1 Sprint 4

In sprint 4, time was set off for development and for catching everyone up to speed in regards to how all the parts of the application worked. Because of the increased focus on programming and development, members who had focused on design and user stories in earlier sprints needed additional knowledge about the structure and development of the project in order to work efficiently. A decision was made to use

pair programming for the sprint and to increase the amount of time spent in dialogue.

Sprint 4 showed improvement in individual and group development work, which aided development in future sprints. The sprint began the plans for the connections to the NCore servers and the OData API and included follow-up meetings to the earlier meetings with developers at Sikri.

4.3.2 Sprint 5

Our goal for this sprint was to continue the development of the Frontend while starting preparations with Sikri for a connection to the NCore servers. Besides Frontend development and improvements, unit tests would be added for the project.

We were tasked with the challenge of presenting the project to the employees at Sikri. The presentation took priority for the first week of the sprint and proved an educational and inspiring experience. The feedback from the director and employees was very motivating as they were very interested in the project. Overall feedback was very positive and demonstrated that the company is likely to further develop the application once our work is completed.

In this sprint, we improved the UI of the Frontend, started the plans for server connections, and started the development of tests for existing code.

4.3.3 Sprint 6

The overall goal for sprint 6 was to connect the project to Elements and the NCore servers. This would include an increased amount of meetings with developers at Sikri to share knowledge and instruct the group.

This sprint was cut short because of Easter. Service files to handle connections to the NCore service were added and connected to the project case list, based on analysis and plans from sprints 4 and 5.

4.4 Sprint 7-9 - Wizard C

This period was used to finalize all necessary connections to the Elements server, refactor and redesign the application to work with and better suit the Elements/NCore environment, and end the project with a working application.

4.4.1 Sprint 7

Our goals for Sprint 7 were to refactor and redesign in order to ready the project for the remaining connections with the NCore servers. The structure of the wizard had to be redesigned, as our implementation was based on specific JSON schemas which did not reflect the models at the server. Problems in the Redux store, our state management, had begun emerging, as the structure and models of the store were implemented before any connections. Too much of the store, and application in general, was based on early assumptions regarding the structure of the data and would have to change in order to fit with actual data.

Sprint 7 taught us the importance of agility in project management, as large parts of our assumptions regarding the application structure were invalid in light of new information gained from our experimentation with Elements and the NCore servers.

The suggested schemas for incoming application forms (the forms which began a case) were not directly supported by the NCore servers or the Elements architecture. Because of this, we had to redefine the product; the view containing incoming application information would be a PDF viewer, not the customized schema viewer. The wizard had to be reconfigured as well, as we discovered that our standardized JSON schema "Wizards" were ill-suited for the task hierarchy structure of Elements.

4.4.2 Sprint 8

The goals for sprint 8 were to continue the refactoring work and to develop the PDF viewer and the new and task-based wizard. We spent some time redefining tasks and features to fit our understanding of the NCore servers and the Elements environment.

Because of the work done in Sprint 7 in refactoring the project, the adding of an Elements-connected Wizard and PDF proved successful. This sprint also showed

additional improvement in splitting and delegating work, and time was spent on tasks and meetings relevant to individual assignments. The program was now, for all purposes, finished, and only small bug fixes and structure improvements were added to the Sprint 9 plans.

4.4.3 Sprint 9

The goals for sprint 9 were to get the project ready for handover and to finish the project report. In finishing the project, some tasks related to bug fixes, structure improvement, and documentation were added.

5 – Project Execution

This chapter provides an additional description and further details for the activities performed by the team from beginning to end of the project.

5.1 Analysis

At the beginning of the project, we were given a report that was the result of a survey conducted amongst some of the Elements users, which provided background information about the challenge that the users were experiencing and a description of how Wizards could solve this challenge.

5.1.1 Product scope

The report served as the basis for the product suggestion from Sikri. The report contained clear wishes from users of Elements. With the basis in the report, we worked together with the Product Owner, Fredrik Werpen, and the stakeholders to develop the project scope. Werpen shared the role as stakeholder togheter with Merethe Sjøberg and Marius Holen at Sikri. The product scope is defined through user stories created and then narrowed down to an MVP.

5.1.2 Specification

A specification was delivered to us in the report during the pre-sprint by the product owner. This report included feedback from users of elements and suggestions to what could be included in the new system. It also contained a description of the intended user group of the application.

5.1.3 User Stories

Based on the previous report, user stories were created in close cooperation with the product owner and stakeholders. After the user stories had been made, they were prioritized with the Moscow method. The Moscow method is a popular prioritization technique used for managing system requirements (ProductPlan, 2020). This prioritization helped us distinguish between the most important features and the less important features. All of the user stories prioritized as "must-have" formed the MVP (appendix 9). These were the minimum and most vital features required for the application to work as intended. This served as the foundation for the rest of the project.

5.2 Design

The initial plan was to base the design of the application on a style document written by developers in Sikri, which was in line with the current Elements design. Unfortunately, we were not able to get hold of this document for a while, so we decided to create a temporary design and change it when we had the design document. But after the first demo with this temporary design, the group was given more freedom in regards to design choices; and was practically allowed to start from a blank canvas. Sikri proclaimed from the start that they did not necessarily want the system to be based upon the design choices of Elements, as the design choices of Elements were part of the problem Wizards was trying to solve.

5.2.1 Sketches and prototypes.

The normal design procedure is to start with sketches such as wireframes and then to create mockups. A wireframe is a skeletal blueprint that outlines the basic design functions of a user interface. A mockup is a more in-depth iteration of the wireframe that includes more stylistic UI details. The next step is typically to create a prototype. A prototype is typically a function iterative simulation that includes all the stylistic details intended for the final product (Lucid Software Inc., 2020).

Because of restrictions due to Covid-19, the team had to get creative to be able to work on a design together efficiently. In our case, we decided to use an application called Figma (Kopf, 2018). Figma makes it possible to work together as a team, designing in real-time from anywhere. In this way, we could hash out our ideas and

implement them fast. In Figma, we could also create high-fidelity wireframes and mockups easily. It also has the possibility to create clickable prototypes from the high-fidelity sketches created. See sketches and prototypes in appendix 10 - 11

By utilizing the early user stories we had made with the product owner, we could start with the design of the application. By showing the product owner the designs we had made in Figma during our meetings, we could spar together and make improvements based on our high fidelity sketches in Figma. Being able to compare and discuss sketches of the application also helped both the product owner and the group to evolve the design and the user stories, as well as figuring out important features to be implemented in the project. This was a highly iterative and agile process and allowed for continuous design improvements in the early sprints.

A few clickable prototypes were made in Figma to give both the product owner and the group a feel for how the application was seen from a user perspective. After completing the prototypes, the group received confirmation from the product owner that coding could begin. The product owner was particularly satisfied with this process and thoroughly pleased with the design at this point.

5.3 Technical Analysis

Our project can be divided into three different phases, or "programs," separated into the three different bulks of sprints; Wizards A, Wizards B, Wizards C.

5.3.1 System architecture

Wizard A

The first wizard was mainly supported by application-generated data, meaning data generated by the Frontend program itself. A helper folder contained helper files used for the generation of case data, as well as the generation of incoming application schemas from users to be processed by the wizard. The wizards were defined in JSON schema files, and component helper files were used to build the wizard and incoming application interface for the user. Sites, or screens; components belonging to an URL, were placed in a folder named Screens. All smaller bits of the surface, or components called upon inside screens, were placed in a components folder.

For state management, Wizards A had a store folder, which grouped state management for different parts of the state in folders containing a type definition file, and actions files, a reducer file, and a helper file. The Redux store initially managed the state of (1) the system: user (generated upon application start) and routes, (2) cases: a list of generated cases, the selected case id, and search query, (3) messages (scrapped before use), and (4) the wizard: holding the currently in use wizard and incoming application form schemas. See Wizards A in appendix 12.

Wizards B

In Wizards B, the components folder was further separated into smaller groups, grouping components that belonged to the same screen (e.g.,/components/cases contained the CaseList and CaseItem components). This program also finished the initial draft of the wizard, adding additional wizard schemas and additional incoming application schemas. After finishing a data-generation backed prototype, or demo, Wizards B began the process of connecting the application to the Elements/NCore Backend, adding a service folder containing services that held connection related logic for connecting to a Testing/Development environment of NCore, and OData folders containing model schemas for the OData objects used. Wizards B updated the case list to reflect actual cases and allowed a user to search the entire environment database for cases, filtering on case title, as well as connected user id. The connected user was hardcoded for all requests to the NCore service. See Wizards B in appendix 13

Wizards C

In the final implementation of Wizards, all generated data was replaced by actual data retrieved from the Elements/NCore servers, and an additional production environment was added. This means that the helper folders, which handled the generation of the data used by the system, could be removed. Authentication (login, cookie storage) was added for both the production and development environment, although different methods were used. Document reading support was also added to the case details. See Wizards C in appendix 14

The project was generally refactored for performance improvements, and the folder structure was redesigned for increased readability and to reduce coupling and

improved cohesion. The Redux store was redesigned to fit large updates in the design patterns in the official documentation.

Wizards C also introduced a redesign of the application: new animations, new icons, a new logo, some fixes in color scheme, a resizing of application surfaces, and an improved case detail container. A lot of the redesign was done to improve understanding for users already familiar with the Elements system while maintaining ease of use as well as speed-to-learn.

5.4 Risk Analysis

When starting the planning phase of a project, it is important to ask yourself: what can go wrong? There will always be room for things to fail in a project. This is why it is important to make an effort to predict what difficulties may lay ahead. This allows the team to be aware of what needs to be focused on to avoid a major crisis. In order to keep track of the potential risks, as well as their possible harm, the group decided to write down and structure the risks in risk analysis; a document used to track risks. This document was used to describe specific risks, their likelihood of occurring, and the potential damage which could be inflicted. After initially analyzing the potential risks, the group has continually documented occurring risks (Bridges, 2019).

5.4.1 Risk Register

The risk register contains a list of 25 identified risks that could have the potential to negatively impact the project. The risks are sorted and categorized by a defined score, which is calculated for each risk in the list. The risk score is a combination of a probability score of 1-5 (the likelihood of a risk to happen) and an impact score of 1-5 (how big of an impact risk has). A higher combination of these two scores results in a higher total risk score. The list also consists of a short explanation of how to avoid the risk before it happens and how to limit the impact of the risk if it happens.

A couple of examples of identified risks were: "Poor communication with the Product Owner," "Getting stuck," and "Illness." The list of risks was constantly monitored and modified. New risks had to be added as new things not earlier identified occurred. An example of this was a change in Sikri's team, and was added as "Change in the client team." The risk register can be viewed in appendix 15.

5.4.2 Issue Log

In the issue log, risks that occur are logged with details attached. The log consists of a list with what type of risk has occurred, risk score, open/close date as some risks lasted for longer periods, and a comment that gives more in-depth details about this particular incident and how it was solved.

Some of the risks in this list were recorded more than others. Technical problems (risk #17) were a bigger issue than anticipated and eventually had to be updated to reflect this. The complete issue log with recorded risks can be seen in appendix 16

5.5 Implementation/Development

The development process is very central to the project. This is where the vast majority of working hours are spent. It was therefore discussed at an early stage and decided what kind of work process should be followed. This chapter will describe how the work was carried out in the form of work routines and work distribution.

5.5.1 Work Routines

Well-established work routines are important in projects like this. Scrum can be an important tool and aid in achieving good work routines. There are, however, many different ways of implementing Scrum and work routines. The group's implementation is described below:

5.5.1.1 Sprint planning

Every weekday started at 09:00 with a daily stand-up, which lasted for about 15 minutes or more depending on the situation, the day typically ending at 15:00. These meetings were conducted via Discord from home because of Covid-19 restrictions. This worked very well, and the team actually never met physically during the entire project but collaborated closely digitally. Each sprint lasted for two weeks. The first Monday of the sprint started with sprint planning. This involved generating new tasks in Azure DevOps for the sprint, running a Planning Poker to estimate time for each task, and delegating tasks (appendix 17). Planning poker is a secure and fun way for agile teams to guide sprint planning and build accurate consensus estimates (Planning Poker®, 2020). The tasks were generated from features that were created from all the most important user stories. As tasks were created, they contained a

short and concrete description and acceptance criteria. It also contained a time estimate, time spent, and time remaining on the task.

5.5.1.2 Implementation

Tasks were delegated to individuals or pairs of two depending on the complexity of the tasks. Tasks were also created to be finished in a day or less, meaning that if a task took more than one day, it would be broken down into several tasks. A task was always connected to a feature, each feature having its own branch. When a task had met its acceptance criteria, and unit tests had been performed, it was pushed to the feature branch linked to the task. After all relevant tasks and unit tests connected to a feature were complete and pushed to its related branch, a pull/merge request towards the main branch was created. Two members of the team worked as reviewers in Azure DevOps through the whole project, which meant that no code could be pushed to the main branch before approval from these reviewers. These work routines ensured quality throughout the project.

5.5.1.3 Sprint Conclusion

The last Friday of each sprint was used to do a sprint retrospect, a reflection of how the sprint went, and to merge all branches not already merged into the main branch. This was followed by a sprint review and demo with Sikri highlighting the progress of the project. The last task of the day was to close the sprint in Azure DevOps after consulting with Sikri.

5.5.2 Work Distribution

At the beginning of the project, the group decided that all members should participate in all, or most, activities, in order to maximize learning outcomes for everyone. To some extent, it was possible to maintain this, although there was some further distribution of tasks closer towards the end of the project, as roles were more clearly defined. In the work phase of the project, all the members helped to organize the startup by reading up on project management, as well as the various components needed. Everyone attended the meetings with Sikri, and we made user stories together as a team.

When the programming started, we maintained an interest in everyone participating, in order for everyone to understand how the program and the overall code functions.

As mentioned, further towards the end of the project, the task division changed slightly, as the number of tasks increased, while the amount of remaining time did not. We found that it required an incredible amount of work for everyone to have an equal understanding of all the work, and therefore chose to divide the work focus a little differently. As the aim was still some equal contribution, we maintained some balance of activities, although everyone became more specialized. Two group members prioritized coding, while the other two group members prioritized project management and report writing, although everyone has contributed to all parts of the project to a certain degree.

6 - Quality and Testing

In this section, we will explain the steps the team has taken to assure quality throughout the entire project and how testing was done to achieve this.

6.1 Quality Assurance

Early in the project, we discussed how we could keep a high standard of quality in the project. We had a goal as a team to deliver a product of high quality and to learn as much as possible. In projects of this magnitude, it is important to keep track of everything that leads to a successful and high-quality product. To enforce this, as mentioned previously, we focused on setting up a proper folder structure to keep everything well organized. It was expected that this project would expand, and in the end, consist of many documents.

The group members had different experiences in regards to working on projects such as this. It was then decided to set up code standards and guidelines to support us with the project. These documents can be found in appendix 4, 5, 18, 19, 20, 21. Pair programming was also adopted to help with the disconnect in experience between the team members. Following guidelines and working in pairs of two would help us learn more as well as enforcing high quality. Ensuring that features were tested and the acceptance criteria were fulfilled was something we valued. This was to ensure the application performed as expected and to ensure the features fulfilled their requirement definitions. A standard for commenting and description was set up for the codebase in the hope that these standards can help ensure high quality and a smooth handover and transfer of the product to Sikri.

6.1.2 Acceptance Criteria

All user stories had their own acceptance criteria. Acceptance criteria are a set of predefined requirements that must be fulfilled to mark a user story complete. Acceptance criteria are also sometimes called the "definition of done." This is because they determine the scope and requirements that must be executed to consider a user story finished (ProductPlan, 2020). These criteria were approved by the stakeholders. For a user story to be set as complete, these criteria must be met, and the feature(s) correlated with the user story must be tested and confirmed by at least two members of the project team.

6.1.3 Code Standard and Guides

Guidelines and code standards were developed to support the team with the different tasks tied to the project. Git guide (appendix 4), Redux guide (appendix 20), React-components guide (appendix 19), branch guide (appendix 21), sprint review, and sprint guide (appendix 22) were guides we developed together to get a uniform understanding of how to work with the different technologies and how to get started and implement new features. They also assisted us in how to deem a task complete and guided us with the necessary steps to complete a merge to the main branch.

6.1.4 Pair Programming

Pair programming has been used frequently during the project. Pair programming is an agile software development technique where two programmers work together on a computer. In our case, this happened with screen sharing over Discord. One of the programmers is the "driver" (screen-sharer), and the other programmer is the "observer" (watching the video stream) (Wikipedia contributors, 2021e).
This was helpful in our case as it was easier for the observer to spot mistakes in the code while the driver was coding, also discussions on what methods, approaches, and sharing of ideas can be made on the fly.

6.1.5 TypeScript

TypeScript will help keep high quality and ensure the code is working properly by requiring types. A key difference between TypeScript and JavaScript is that TypeScript needs to be compiled while JavaScript code does not need to compile

(Rungta, 2021). In JavaScript, types are inferred in runtime and might run into issues or errors when running. With TypeScript, these errors will be caught in the compile step. This leads to higher quality and less time-consuming debugging. Having to set types for everything will help complement unit tests as there is no need to create unit tests that are testing the types. This is built-in for TypeScript.

6.1.6 Comments and Descriptions

As the project developed, the need for comments and descriptions for the codebase increased. A comment in computer programming is a readable explanation or annotation of a block of code in the source code. Their purpose is to make the source code easier to understand for humans (Wikipedia contributors, 2021f). This was especially important because it would make the handover process of the project and code to Sikri much smoother.

6.1.7 Refactor

Later in the project, after several changes to the Scope/MVP, we saw the need to tidy up the codebase. As some functionality was dropped and better solutions were found, we decided to make a proper refactor of the code before the handover to Sikri was going to happen. Code refactoring, in computer programming, is the process of restructuring existing code without changing its behavior. Refactoring is intended to improve the code while preserving the functionality. Advantages of refactoring may include improved code readability and reduced complexity. This will then help the source codes maintainability and sometimes improve performance (Wikipedia contributors, 2021g).

The Wizard applications' Redux store had been modeled on the design patterns documented in Redux's "Usage with TypeScript" documentation. These design patterns were redefined by Redux later on, once the Redux Toolkit was added to the recommended approach of how to use redux (Abramov & Redux documentation authors, n.d.-c). The previous design patterns had advocated a folder structure that split the code, defining the configurations, for state management between a large number of files, which proved time-consuming to work with. Any time the state of the application had to be redefined, changes had to be made to up to four files. It also made it more difficult to get a quick overview of the code. A decision was made to refactor the structure to fit the new design patterns. With the refactor, we

achieved a more readable source code as well as improved design and performance by reducing coupling and improved cohesion.

6.2 Testing

In software development projects, it is important to locate and deal with errors and flaws as early as possible. This is where testing comes in. It will help locate these flaws or errors early and prevent future errors that may lead to setbacks in development. After every new major feature is added, it is a good idea to conduct tests to ensure quality. This is to remove potential errors or flaws that may be an issue when implementing new features.

6.2.1 Unit Testing

Unit testing is a type of software testing where components or individual units of software are tested. The purpose is to validate that each unit of the code performs as expected (Rungta, 2021). Unit tests lead to a high-quality and robust software system. Unit tests were performed on state updates as well as data fetching functions and utility functions. User interface components were tested in the later part of the coding process when the components were decided to be included in the end product. An example of a unit test can be found in appendix 23.

6.2.2 User Test and Demos

At the end of every sprint, there was a demo of the application. This was to show the design and functionality to the Product Owner. In these demos, it was possible to pinpoint problems and necessary changes the Product Owner wanted. This helped the stakeholders to steer the project in the right direction and make changes to the scope of the project as obstacles were encountered. In the beginning, only the Product Owner and coordinator had access to these demos, so the feedback was limited. We brought this up, and they agreed to ask other employees in Sikri to see some of the demos.

Later in the project, we got the opportunity to test the application with real users of Elements. These tests were conducted by the Product Owner, and the feedback was given to us continuously over Teams meetings.

7 - Reflection

This chapter will address several different aspects of the project, including project management, the quality of the end product, and the risk management process.

7.1 Changes and Challenges

This section is about the changes and challenges linked to the product definition and MVP, code, resources, and risk management.

7.1.1 Product and MVP Definition

At the beginning of the project, the MVP seemed easier to define than it turned out to be, and it became increasingly difficult to contain the scope within the original definitions. The first challenge that arose was the lack of experience in defining an MVP within the group. The second challenge was the fact that no members had any experience with case processing, which made it more difficult to define an MVP for a case processing software. The third challenge was the unfamiliarity and lack of knowledge with the massive code base of Elements and the depth of technology. As all these challenges were mended incrementally, as the group gained experience, the originally defined project had to be refactored to better fit an increased understanding. The use of 14-day sprints allowed increased agility in this regard. Practically, this was done through a rigorous sprint retrospective and sprint planning, in which the group fixed and rethought feature and task definitions.

The main parts of the original MVP and project definitions that proved difficult were the shape of, or models for, the wizard and the incoming application form.

7.1.2 Code

The developed code has gone through several iterations throughout the project, and a sufficient connection between code, product definitions, and project management has shown itself important. Finding a good balance between strictness and flexibility proved difficult but became easier as the project became more developed.

7.1.2.1 Incoming Application

The incoming application is the part of the Wizard program that displays the incoming case details to the caseworker. While the Elements software provides incoming applications as PDF or DOC/DOCX files, the group was instructed in the early sprints to base incoming applications on forms (or schemas) structurally defined in an XML or JSON format. (For examples on such forms, see https://www.skjema.no/). The intention was for this to simplify the incoming application form in the application, as these schemas were transformed to a standardized format through the Elements Backend layers. Early development was done with this in mind, and the first finished demo program - Wizards A - built on the idea of these forms.

After the initial connections to the NCore servers were completed, and connections were to be made for the Wizard and application forms, it became apparent that these schemas and formats were not made available and could not be served through the API (without a large amount of additional work for the Sikri developers).

A decision was made to rebase the incoming applications on the PDF and DOC/DOCX files, and the original view for incoming applications was scrapped. An added benefit of this was the fact that Elements had a component for document viewing for PDF and DOC/DOCX files, which was added to the project in place of the previous schema-based preview.

Though not directly following the initial challenges mentioned, the challenges that arose in the development of the incoming application view could have been mitigated with sufficient knowledge of the Elements code base and environment. If inevitable, the example points out the importance of agility in project management and the value of a close dialogue with developers with relevant experience.

7.1.2.2 The "Wizard."

The wizard is the part of the application that the caseworker uses in processing a case. The initial wizard was developed for Wizards A at the end of sprint 3 when the group was still lacking in technical depth in regards to the Elements code base and environment. Inspired by the application viewer, the Wizards code served as a "Wizards Builder," rendering the wizard based on an incoming JSON schema. For

demonstration purposes, these schemas were hardcoded and connected to each of the three defined application schemas mentioned above. The Wizards A project made some assumptions in regard to the structure and models of the backend, which caused some time spent on refactoring and rebuilding. Although the Wizards A had been built mainly for demonstration purposes and design testing, it could have proven more effective to implement it "correctly" the first time. However, the early demo software increased discussion points and led to a better understanding of the intended product and its implementation. It also allowed for a larger amount of reiterations over design, which further improved the final product.

7.1.2.3 General

Besides the difficulties relating to fitting the code inside the project and MVP definitions and within project plans, there were some additional challenges. One challenge that proved more difficult than initially foreseen was the fact that the project had to be developed by the group as a team. Differences in experiences, methods, and knowledge made the requirement of rigorous standards and documentation is key to effective development. Increased discussion, and sharing of knowledge, became important in order to familiarize everyone with the frameworks and concepts the project was built on.

Although one came after the other, the initial progression speed of the project was made possible through simultaneous work on defining user stories, designing the program, and developing the program. This led to a lot of early code being written while some members were working on user stories and design. This made it more difficult for everyone to jump in and contribute immediately once the user stories and design stages were finished. In order for everyone to gain an understanding of the written code, the group chose to increase communication and implement the use of pair programming for development in Sprint 4.

7.1.3 Resources

In this chapter, we talk about the different resources that were available to the group during this project, both human and technical resources.

7.1.3.1 Human Resources

In a company like Sikri with over a hundred employees, there are many human resources (HR's) available, and Sikri made them available to us. This was positive because there was always someone that had an answer to the question at hand. However, we also discovered some of the negatives with this. One of these negative things was that even though the right HR exists somewhere within Sikri, finding the right HR can be difficult.

Many meetings with different HR's were held before finding the right one. This was time-consuming and not something we had accounted for in the planning phase of the project. However, when the right HR's were found, the answer was usually very helpful and satisfying. Another discovery we made was that the knowledge of the HR's within the company typically was specialized in a specific area. This meant that there were few, if any, with Fullstack competence. This required meetings with many HR's to get the complete picture of information needed to solve a specific problem.

7.1.3.2 Technical Resources

Connection with the Backend had some challenges. The test cases provided did not match real cases. Another issue was that a case had several posts tied to the case, and the workflow we needed to process the case could be tied to different posts. Because of this, there was confusion about how we wanted to implement the feature related to a case process. This was later solved in discussion with Sikri. We were to focus on the first post, and the workflow was tied to the first post of the case.

Another issue we encountered was that the development environment we were connected to went through some maintenance, and therefore work related to that came to a stall. This was temporarily solved when we got access to another environment Sikri used for testing. However, this was time-consuming, and the test cases were different from the original environment.

7.1.4 Risk Management

In the beginning, we struggled to see the positive outcome trying to predict what could possibly go wrong. We had done earlier school projects without it and had not encountered too many problems. However, it would turn out that risk analysis had its benefits after all as we encountered several of the risks listed.

Some risks were solved with temporary solutions like the technical problem with the development test environment being offline. The impact could have been mitigated if we had other tasks to work on, but in this case, we had a deadline tied to the task that relied on the environment is online. To limit the impact of this risk, we had to get access to another test environment. This worked out well, and we finished the task within the deadline. Other risks could only be mitigated and not resolved, like absence from team members due to illness (appendix 16).

Having done a risk analysis helped the group to know what to do when risks occurred. It also helped with general risk management and when new risks appeared. In retrospect, we have a good understanding of why risk management is important, as it can help a project to become successful.

7.1.5 Keeping the Project Agile

By having frequent meetings with the product owner and not being afraid of making changes to the requirements, even late in the project, the group was able to keep the development process agile. There were many changes throughout the project, as mentioned earlier, but by reflecting on what we had learned through the different sprints, we were able to adapt and overcome these challenges. This was especially important towards the end of the project when the deadline was getting close. Some big changes were made, and we had to utilize what we had learned to be as efficient as possible and to make that deadline, which we did by being agile.

7.2 Learning Outcomes

In this section, we talk about some of the experiences and things we have learned during this project.

7.2.1 Professional Evening

The team was asked by the product owner to present the project at a professional evening (PE) with the employees of Sikri. A PE is a forum where employees have the opportunity for personal development and to share knowledge. This specific PE was hosted via Microsoft Teams due to the ongoing pandemic. Here we held a 30-minute presentation of the project. About 35 employees attended this meeting,

including the CEO of Sikri. The meeting was also recorded and shown to many more employees and has been shown in several other events.

This experience was very exciting for the entire group, and the first time most of us had ever had a presentation resembling what would be more common in the real world. The presentation was also very positively received by the employees of Sikri and was, all in all, a very positive experience for the entire group.

In the beginning, we did not understand the impact this presentation had. But during later unrelated meetings, we found out that our project had been presented to the entire staff of Sikri in a morning meeting. This gave us the feeling of really working on something important and increased our motivation even further.

7.2.2 Project Management

Project management was a concept we were quite unfamiliar with. From knowing very little to being able to manage a whole project from start to finish required discipline. We have learned about different tools that help with project management, such as Gantt chart, Planning Poker, project management tools such as Azure DevOps, general project planning, and the value of having a plan to follow.

How to delegate tasks and time estimation of these tasks were skills we developed during the project and might have been the best experience from the project management side.

7.2.3 Technical Skills

The group developed several technical skills during the project. These skills were tightly tied to the technologies we used, like React, Redux, TypeScript, API/Odata, and Git. Some of us had experience with some of these technologies from past projects, and for some of us, it was a completely new experience. Anyhow, we all learned and evolved as developers and project managers during this project.

The main technologies used in this project were React and Redux, written in TypeScript. These technologies are the fundamentals of the project, and it was crucial every team member became familiar with them. We all got different levels of understanding of this, but all in all, this was a success. We learned that even though

TypeScript felt slower in the beginning to use, it rewarded us with fewer errors in the end.

As mentioned earlier, we decided to split the team to cover more ground, and therefore the skills absorbed were different for each group member. One part of the team focused on the API/OData part for connection to Sikris APIs. The other part focused on the project management part of the project. When it comes to the API part, we learned how this worked and how to look through the network tab to understand the flow of connections to different APIs.

How to use version control, like Git, properly was an important skill we developed. As we started with more features and developed several parallel features, we saw the need to keep control over the different versions. This is done through Git and their branch system.

7.2.4 Cooperation Skills

A large project like this requires good collaboration skills. The challenges are many, and it is important that everyone collaborates and pulls in the same direction. If there is poor coordination and many misunderstandings, it will be difficult to deliver a good product. Everyone was aware of this from previous collaborations, and this project has some extra factors that make it even more difficult.

7.2.4.1 Planning

In the pre-sprint and into the first sprint, the group made a number of documents to be able to keep track of everything that concerned the project. This choice has proven to increase our productivity since a lot of organization makes it easier for everyone involved to keep up with what is going to happen at any given time. Something we learned is to set up a Gantt chart from the very beginning. Our solution at the beginning of the project was to create an overall plan in Google Sheets from scratch (appendix 24). This worked well for a short period but quickly became confusing when there were many tasks to be completed. Fortunately, we were introduced to the Gantt chart and implemented this with great success. It gave both the client and us a good overview of when the various tasks were planned to be completed.

7.2.4.2 Meetings

None of us had much experience of communicating with clients in a professional context, but we did it many times throughout the project, and today we are left with a lot of experience in this area. In the beginning, we had a slightly sloppy structure in the meetings we were to present project status. We received feedback on this from the Product Owner in the form of constructive criticism. Of course, we took this into account and managed during the course of the project to tighten this up so that the meetings under our auspices were tight and concise. The collaboration we had with Sikri was very educational and fun. They were honest and fair and always gave us tips on how to improve and prepare for the future at work.

7.2.4.3 Delegate Work

In the beginning, the group members worked a lot together with all the tasks related to the project. There was a lot to learn, so the road was created as we walked it. It worked well in the beginning, but as the tasks became more complex and the deadline approached, we had a discussion about how we should cooperate further. This choice took place approximately in the middle of the project, and we then came to the conclusion that we had to delegate the tasks more to the members who were best at the respective tasks. This choice was well reflected, and we see in retrospect that it was absolutely necessary for the project to be completed with high quality and on time.

7.2.4.4 External Guidance

The complexity of the project was a bit uncertain when we started the development. It was not very specified by Sikri what challenges we were going to face. There was a certain understanding of everything that needed to be done, but as the project developed, we encountered many challenges. In the beginning, we spent a lot of time-solving these problems ourselves. It took a lot of time, and the motivation sometimes decreased with the feeling of not getting the job done. Sikri said early in the project that the resource persons in Sikri were ready to help if problems occurred. We became much better at using this in the last half of the project. We received invaluable help from developers and project managers. We, therefore, learned that it is sagacious to ask for help and that the client benefits when we solve the tasks efficiently and correctly.

7.2.5 Self Evaluation

This project was developed by the members of /*TODO*/. The members put in an equal amount of work, As mentioned in the report, there has always been a focus on learning, and every member has contributed in some way to all the big tasks. Anyway, as the project evolved, the tasks were more distributed to the members with the best knowledge on given tasks. Below, the group members will describe more details about personal contributions and thoughts about the project in general.

Aleksander Larsen

During the project, my goal was to gain experience in how project management and software development were done in a real project. During the project, my tasks have been diverse. I have been fortunate to be involved in some form or another in every part of the project, such as programming, design, writing meeting summaries from meetings, planning, and more. I have been able to use a lot of the knowledge gained in BACIT. Some of these are: IS112-Tjenestedesign og foretningsmodeller (design og applications), IS 104 Digital interaksjonsdesign (design) IS 202 Programmeringsprosjekt (programming), IS 200 Systemanalyse og systemutvikling (analysis) and IS 308 Internetteknologier (Creating docker image and deployment of application). I have been able to use this knowledge, see it in use in a real-world context and develop them further. Finally, it has been very interesting to see how it is to be working in teams in a real project and how important project management is to be in control of the project.

Ole Haraldseth

In this project, my main tasks have been project management. I have worked as a Scrum Master, and that role made it my responsibility to make the project management to work out smoothly. Tasks that have been in focus includes communication with Sikri, meeting planning, meeting management, document management and more. In addition, I have participated in the work that had to be done in the initial phase of creating user stories, designing elements in Figma, and creating and managing documents for time registration, Gantt chart, and standard documents. Coding has not been my most central work area, but in the first half of the project, I contributed a good deal of coding of various elements of the application. It has been a very instructive project, and the knowledge gained on the

Bachelor's program in IT has come in very handy. I would especially like to highlight the subjects related to project management. An example of a subject that has been central to my work areas in this project is "IS-200 System analysis and system development ". Here we learned about how Scrum can be used in a project, and it has made me much more confident in the role of Scrum Master. Finally, I would like to add that the collaboration within the group has been very good, and the lessons learned from working together closely as a group with an external part comes with great learning outcome.

Daniel Mossestad

My main focus in this project was to work with Fredrik with the code development part. This includes writing components, data fetching functions and help with the overall design in the application. Another big task was to support Ole with the project management part. This includes document management, taking notes, and writing summaries of meetings. In the early sprints, we all worked together with creating user stories, designing components in Figma, and creating and managing documents for standards, time registration, and Gantt chart. It has been a steep learning curve, especially in the start, in the project and I have learned a lot! Most of the knowledge gained from the bachelor's program in IT has come in handy when tackling a project like this. Fundamentals learned from the subjects like "IS-110 Objekorientert programmering" and "IS-211 Algoritmer og datastrukturer" was a big support to help further learn more about software development. Subjects like "IS-308 Internetteknologier" and "IS-214 Informasjonssystemsikkerhet" introduced important technologies and aspects about information systems that also helped a lot when understanding the new technologies that are available on the web. I also want to point out that the subject "IS-104 Digital interaksjonsdesign" introduced my passion for creating better user designs. In general, my experience from this project has been pleasant. The group has had a very good synergy and collaborations have been a breeze!

Fredrik Meltveit

I acted as the lead developer throughout the project. I divided and set tasks between group members in Azure and reviewed pull requests together with Daniel. I also set up the project and the initial tech stack, wrote the code standards to be used for the team, and sketched out the initial design of the application.

My design and development skills have improved a lot since I began my studies. I initially began learning Figma in "IS104-Digital interaksjonsdesign" as a design alternative and Spring (which is used to serve the deployed versions of the website) during "IS202-Programmeringsprosjekt(programming)".

7.3 Statement from Client

Statement from Sikri

Sikri is a software company, specializing in delivering essential digital solutions to the Public and Private sector. We were established in 2020 but our history dates back more than 20 years. We have a strong track record in delivering the market leading system for case management and archiving and are extending our reach into a wider range of software and solutions.

Sikri has had a close cooperation with The University of Agder (UiA) for several years. For Sikri this includes having students or student groups stay with us during their bachelor year. Through their stay, they work on their project while getting guidance and resources from Sikri. The projects are generally related to a business areas Sikri is involved in. For Sikri this has been a fruitful arrangement. It enables us to contribute to the important work UiA does in offering good education possibilities for people who wants to work in IT and at the same time giving us new insight and fresh perspectives from engaged students.

Spring 2021 the group consisted of Aleksander Bævre Larsen, Daniel Mossestad, Fredrik Meltveit and Ole Haraldseth. Their project was based on a report written by another student who stayed with us the fall of 2020. In short, the report indicated that it would be preferable for Sikri to add a super light web application on top of our feature rich Elements platform for handling specific tasks in a more simple and straightforward manner.

In their relative short stay, the student was able to develop a complete MVP which was well integrated with our Elements platform. The students worked impressively as a team, including handling delegation and division of roles. They also handled project management, sprints and reporting with excellence. We were really impressed by what the students were able to achieve in this short period of time with an elegant app design, developing of all necessary functions in React and setting up a relative complex back-end logic. The feat even more impressing taking into account that this was done during the Covid-pandemic with the students manly coordinating and cooperating virtually.

It has been a privilege to work with these students and I give them my warmest recommendations.

Best regards

Fredrik B. Werpen
Business developer and Supervisor/product owner for the project
Sikri AS

sikri

06.05.21

Sikri AS • Vollsveien 4, 1366 Lysaker
87 65 43 21 • post@sikri.no • www.sikri.no
Org.nr.: 922 308 055

8 - Conclusion

In collaboration with Sikri, we have created a product named Wizards, which is a light version of Elements. The main goal for Sikri was to develop a user-friendly, modern, and light alternative to Elements. We had a few limitations with the programming language and workflow. Other than those limitations, we had free rein. The freedom and trust Sikri gave us is much appreciated because it put our creativity and independence to the test. In addition, we discussed within the group so that it was agreed on which programs and technologies seemed relevant concerning the issue we had. We see our project as a great success. The client has given good feedback and views the product as finished and with very satisfactory quality. Sikri has announced that the product will be adapted to customers and sold for relevant cases already in the autumn.

The learning benefits of this project have been incredible. The group members have acquired large amounts of knowledge regarding programming and software development, but many of the lessons are about project management, group collaboration, and working closely with an external client. This new knowledge will come in handy in the future when entering the labor market.

In addition, all the group members have contributed significantly to this project ending up as a success. Different members have different strengths, and we have utilized and distributed this in a good way. The communication has been excellent, both within the group but also in connection with Sikri. There has been persistent meeting activity through Microsoft Teams with Sikri, which has led to the impeccable communication flow.

The product is currently deployed and retrieves real live data from Sikri's archive core. It connects to a test environment, but when the application is handed over to Sikri, they can easily connect it to the active archive core when they want to use the product for their customers. In other words, it is an application that is fully executable and ready for customers to use, something we are very proud to have achieved.

References

Abramov, D. A. & Redux documentation authors. (n.d.-a). *Redux Toolkit | Redux Toolkit. Redux Toolkit*. Retrieved April 28, 2021, from https://redux-toolkit.js.org/

Abramov, D. A. & Redux documentation authors. (n.d.-b). *Redux Toolkit | Redux Toolkit. Redux Toolkit*. Retrieved January 20, 2021, from https://web.archive.org/web/20210202221558/https://redux.js.org/recipes/usage-with-typescript/

Abramov, D. A. & Redux documentation authors. (n.d.-c). *Redux Toolkit | Redux Toolkit. Redux Toolkit*. Retrieved January 20, 2021, from https://redux.js.org/recipes/usage-with-typescript/

Abubakar, A. M., Elrehail, H., Alatailat, M. A., & Elçi, A. (2019). *Knowledge management, decision-making style and organizational performance. Journal of Innovation & Knowledge*, 4(2), 104–114. https://doi.org/10.1016/j.jik.2017.07.003

Beck, K. B., Beedle, M. B., Bennekum, A. V. B., Cockburn, A. C., Cunningham, W. C., Fowler, M. F., Grenning, J. G., Highsmith, J. H., Hunt, A. H., Jeffries, R. J., Kern, J. K., Marick, B. M., Martin, R. C. M., Mellor, S. M., Schwaber, K. S., Sutherland, J. S., & Thomas, D. T. (2001). Manifesto for Agile Software Development. Agilemanifesto. https://agilemanifesto.org/

Bridges, J. (2019, December 9). *Risk Analysis 101: How to Analyze Project Risk*. ProjectManager.Com. https://www.projectmanager.com/training/how-to-analyze-risks-project

Git. (n.d.). Git. *Git --Fast-Version-Control*. Retrieved March 25, 2021, from https://www.git-scm.com/

Hastie, T., & Tibshirani, R. (1998). *Classification by pairwise coupling. The Annals of Statistics*, 26(2), 1–4. https://doi.org/10.1214/aos/1028144844

Lucid Software Inc. (2020, February 27). *Wireframes vs mockups: Determining the right level of fidelity for your project*. Lucidchart. https://www.lucidchart.com/blog/wireframes-vs-mockups#:%7E:text=A%20mockup%20is%20a%20static,mockup%20is%20a%20visual%20mode

Kopf, B. (2018, July 31). *The Power of Figma as a Design Tool*. Toptal Design Blog. https://www.toptal.com/designers/ui/figma-design-tool

Material Design. (n.d.). *Material Design*. Retrieved February 2, 2021, from https://material.io/

Material-UI. (n.d.-a). *About Us - Material-UI*. Retrieved April 13, 2021, from https://material-ui.com/company/about/

Material-UI. (n.d.-b). *Material-UI: A popular React UI framework*. Retrieved April 14, 2021, from https://material-ui.com/

OData. (n.d.). *OData - the Best Way to REST*. OData – The Protocol for REST APIs. Retrieved May 7, 2021, from https://www.odata.org/

Planning Poker®. (2020, August 28). *PlanningPoker.com - Estimates Made Easy. Sprints Made Simple*. PlanningPoker.Com. https://www.planningpoker.com/

ProductPlan. (2020, October 19). *What is acceptance criteria? | Definition and Best Practices*. https://www.productplan.com/glossary/acceptance-criteria/

ProductPlan. (2021, February 18). *What is MoSCoW Prioritization? | Overview of the MoSCoW Method*. https://www.productplan.com/glossary/moscow-prioritization/

Redux. (n.d.). *Redux - A predictable state container for JavaScript apps*. Retrieved May 10, 2021, from https://redux.js.org/

RhodeCode. (n.d.). RhodeCode › Version Control Systems Popularity in 2016. Retrieved April 28, 2021, from https://rhodecode.com/insights/version-control-systems-2016#:%7E:text=The%20data%20from%20Stack%20Overflow,and%20Mercurial%20keep%20their%20niches.

Rawsthorne, D. (2020, November 22). Sprint Length: What's the Right Length? - 3Back Blog. 3Back. https://3back.com/scrum-tips/sprint-length-what-length-is-the-right-length/#:%7E:text=It%27s%20a%20rule%20of%20Scrum,Story%20and%20get%20it%20Done.

Rungta, K. (2021a, March 8). *Typescript vs JavaScript: What's the Difference?* Guru99. https://www.guru99.com/typescript-vs-javascript.html

Rungta, K. (2021b, May 9). *Unit Testing Tutorial: What is, Types, Tools & Test EXAMPLE*. Guru99. https://www.guru99.com/unit-testing-guide.html

Tech Terms. (n.d.). Wizard Definition. Retrieved May 12, 2021, from https://techterms.com/definition/wizard

TypeScript. (n.d.). *Typed JavaScript at Any Scale*. https://www.typescriptlang.org/

University of Massachusetts Dartmouth. (n.d.). *Decision-making process*. *UMass Dartmouth*. Retrieved January 20, 2021, from https://www.umassd.edu/fycm/decision-making/process/#:%7E:text=Decision%20making%20is%20the%20process,relevant%20information%20and%20defining%20alternatives.

Wikipedia contributors. (2021a, May 6). Systems development life cycle. Wikipedia. Retrieved May 10, 2021, from https://en.wikipedia.org/wiki/Systems_development_life_cycle

Wikipedia contributors. (2021b, May 8). *React (JavaScript library)*. Wikipedia. Retrieved May 11, 2021, from https://en.wikipedia.org/wiki/React_(JavaScript_library)

Wikipedia contributors. (2021c, May 12). Redux (JavaScript library). Wikipedia. Retrieved May 13, 2021, from https://en.wikipedia.org/wiki/Redux_(JavaScript_library)

Wikipedia contributors. (2021d, April 23). *Material Design*. Wikipedia. Retrieved May 6, 2021, from

https://en.wikipedia.org/wiki/Material_Design

Wikipedia contributors. (2021e, April 25). *Pair programming*. Wikipedia. Retrieved May 10, 2021, from https://en.wikipedia.org/wiki/Pair_programming

Wikipedia contributors. (2021f, April 15). *Comment (computer programming)*. Wikipedia. Retrieved May 11, 2021, from https://en.wikipedia.org/wiki/Comment_(computer_programming)

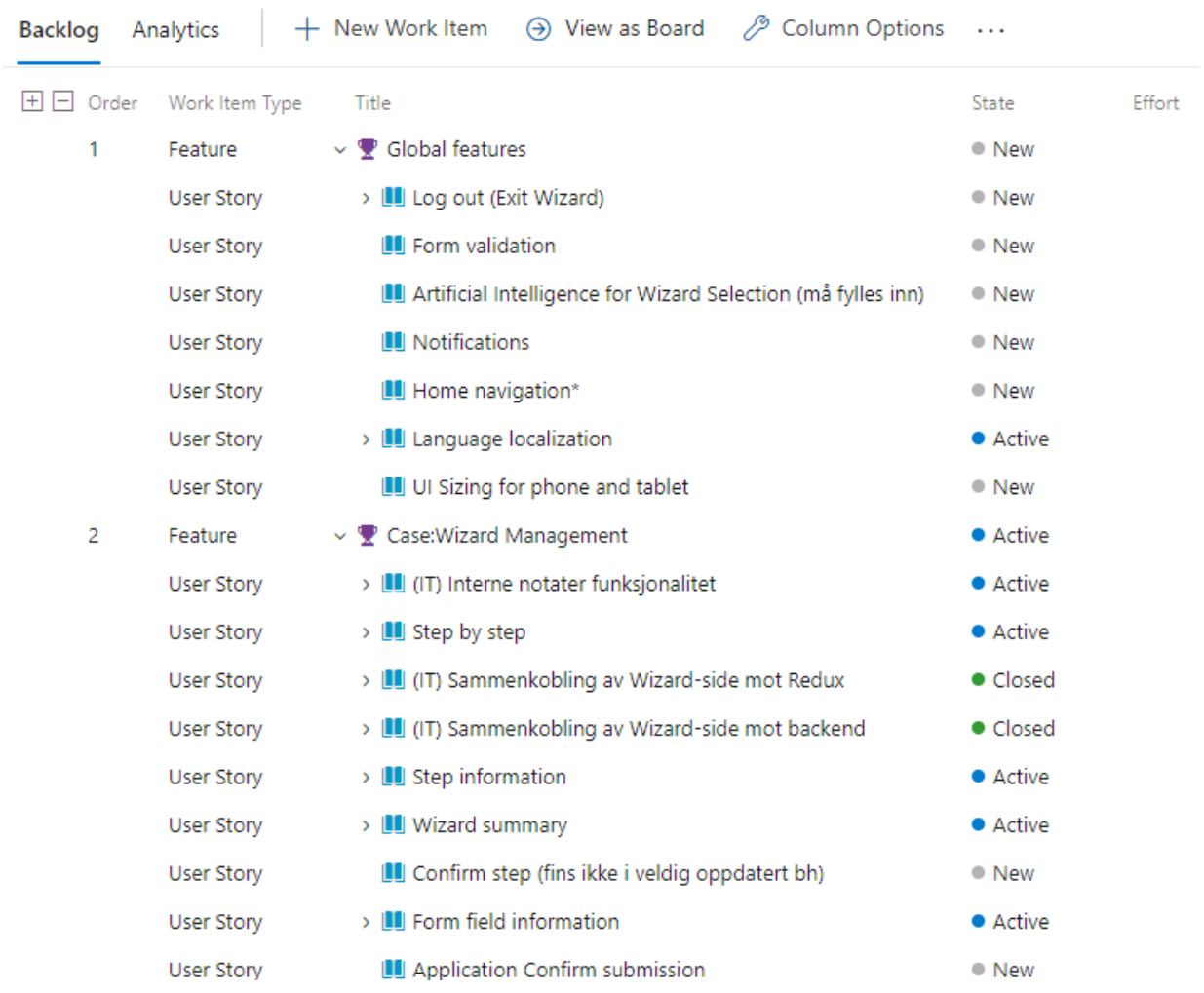
Wikipedia contributors. (2021g, April 7). *Code refactoring*. Wikipedia. Retrieved May 11, 2021, from https://en.wikipedia.org/wiki/Code_refactoring

Appendix

This section will illustrate central documents and figures used in this project. Some of the figures are excerpts of the whole document for demonstration and visualization purposes.

Appendix 1 – Product Backlog Azure DevOps

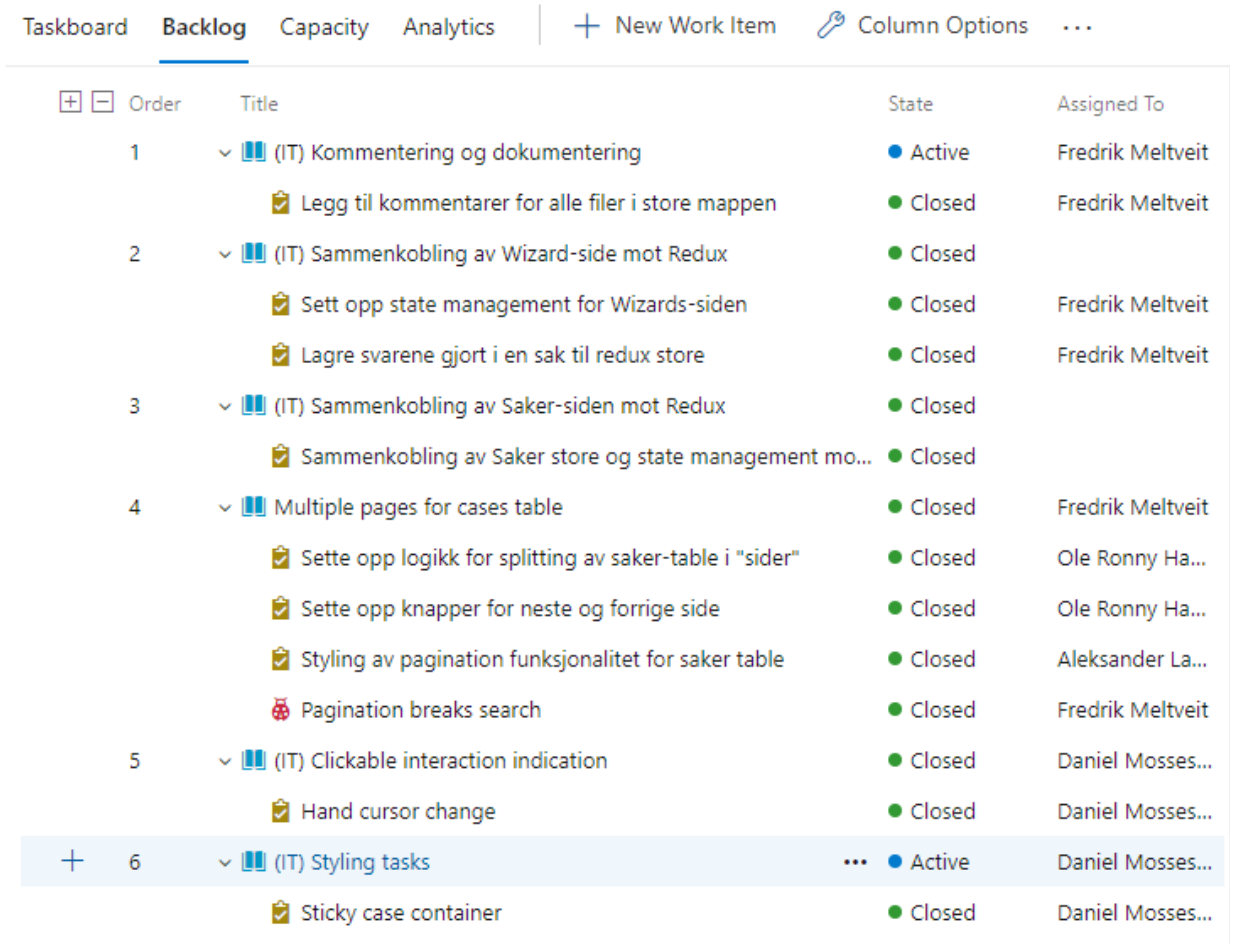
The list is longer, but this is a view of how it is built up.



	User Story	Navigate between steps	New
	User Story	Conditional wizard logic	Active
3	Feature	Inbox	New
	User Story	List display of incoming messages	New
	User Story	Filter incoming messages	New
	User Story	Message display	New
	User Story	Sort incoming messages	New
	User Story	Create Case from Message Display	New
4	Feature	Administrator Settings	New
	User Story	Create, edit, delete Wizard Template	New
	User Story	Create, edit, delete Step	New
	User Story	Create, edit, delete form field	New
5	Feature	Dashboard	Active
	User Story	Display cases with short deadline	Active
	User Story	Cases summary	Active
	User Story	Display cases that are edited by others	New
	User Story	Incomming summary	New
6	Feature	Case Display and Case Management	Active
	User Story	Table display of cases	Active
	User Story	Deadline field for cases table	Active
	User Story	Display case item	Active
	User Story	(IT) Sammenkobling av Saker-siden mot backend	Active
	User Story	(IT) Sammenkobling av Saker-siden mot Redux	Closed
	User Story	Sort cases table	Active
	User Story	Multiple pages for cases table	Closed
	User Story	Aktive / Arkiv select	Active
	User Story	Search cases table	Closed
	User Story	Status field for cases table	Active

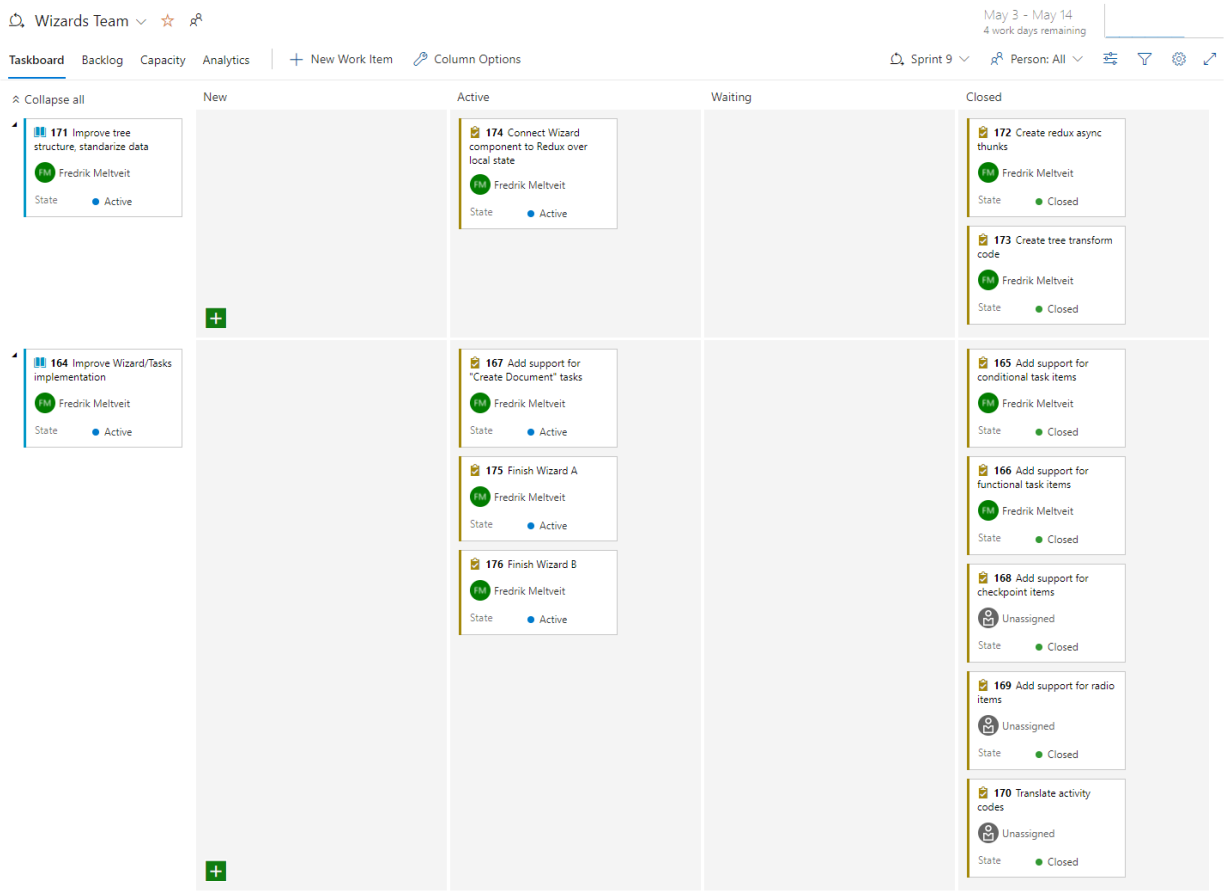
User Story	Wizard field for cases table	Active
User Story	Add, edit, delete Information	New
User Story	Select Wizard	New
User Story	Add, delete coworkers	New
User Story	Edit, delete Wizard	New
User Story	(IT) Clickable interaction indication	Closed
User Story	(IT) Styling tasks	Active

Appendix 2 - Sprint Backlog Azure DevOps



Order	Title	State	Assigned To
1	(IT) Kommentering og dokumentering	Active	Fredrik Meltveit
	Legg til kommentarer for alle filer i store mappen	Closed	Fredrik Meltveit
2	(IT) Sammenkobling av Wizard-side mot Redux	Closed	
	Sett opp state management for Wizards-siden	Closed	Fredrik Meltveit
	Lagre svarene gjort i en sak til redux store	Closed	Fredrik Meltveit
3	(IT) Sammenkobling av Saker-siden mot Redux	Closed	
	Sammenkobling av Saker store og state management mo...	Closed	
4	Multiple pages for cases table	Closed	Fredrik Meltveit
	Sette opp logikk for splitting av saker-table i "sider"	Closed	Ole Ronny Ha...
	Sette opp knapper for neste og forrige side	Closed	Ole Ronny Ha...
	Styling av pagination funksjonalitet for saker table	Closed	Aleksander La...
	Pagination breaks search	Closed	Fredrik Meltveit
5	(IT) Clickable interaction indication	Closed	Daniel Mosses...
	Hand cursor change	Closed	Daniel Mosses...
6	(IT) Styling tasks	Active	Daniel Mosses...
	Sticky case container	Closed	Daniel Mosses...

Appendix 3 - Taskboard Azure DevOps



Appendix 4 – Git Guide

Git

Prosess før man sender pull request og merger brancher. Dette vil gjøre at man håndterer conflicts før man merger. Dette vil utelukke feil/conflicts i main branchen.

1. Pull siste endringer i main og feature branch
    a. Git checkout main
    b. Git pull
2. Switch over til feature branch og rebase
    a. Git checkout <feature-branch>
    b. Git rebase main -i (-i blir brukt for å gjøre den interactive)
    c. Man åpner da Vim editor der man kan se commits som er gjort. Her trengs det som oftest bare å se over og lagre dette med:
        i. :wq
    d. Om det blir conflicts må man ordnet dette her og adde de filene man fikser conflict i
        i. Spør om hjelp, kan legge mer detaljert til her senere.
3. Til slutt lage pull request:
    a. Vi gjør dette via Azure devops
    b. Lag navn på PR og legg til kommentar
    c. Link relevante work items
    d. Legg til reviewers
    e. Create PR

Appendix 5 - Code Standard

Kodestandarder Wizards

Indentation (tab size): 2
Godkjente filtyper for screens og components: `.ts`, `.tsx`
Styling: `makeStyles`

- Styling gjøres gjennom `makeStyles()` funksjonaliteten til Material UI
- Spacing (Margin/Padding):
    - Spacing gjøres gjennom `theme.spacing` funksjonaliteten fra `makestyles(theme)`. Standard spacing bør være `theme.spacing(1)` eller `theme.spacing(2)`.

Komponenter: Functional

- Arrow: `const component = () => { }`
- Funksjon: `function component() { }`
- Bruk den du foretrekker
- // En konsekvens as functional components er at noen ting gjøres annerledes enn i Class based components. Kan skrive detaljer om dette etter hvert.

Kommentarer:

- Kommentar over hver unik komponent /** */
- Kommentar over hver unik screen /** */
- Kommentar til lengre funksjoner i screens og components
- Kommentar over kategori av imports
- !NB Les følgende:
    - https://react-styleguidist.js.org/docs/documenting/
- "!TODO"
    - Ting som må gjøres
- "!Deprecated"
    - Ting som skal fjernes ELLER revurderes fordi de ikke er definert i MVP-en

Validation:

- RegEx (Regular Expression) skal/kan bli brukt for form validering

Testing:

- https://github.com/testing-library/jest-dom

Alle farger er definert i en fil i stil-mappen. Denne bør importeres i alle komponenter og skjerm komponenter, og farger bør **aldri** settes manuelt (uten referanse til Colors).

TypeScript brukes for kvalitetssikring av komponenter.

Appendix 6 - Work Time Registration

Timeregistrering for Daniel

Total Prosjekt: **406**

		Admin	Design	Kode	Rapport	Møte	Forelesning			
Dag	Dato	157	18	98	72	50	11	Total Dag	Total Uke	Total Sprint
Mandag	19. April	2						2		Sprint 8
Tirsdag	20. April	1		4		1		6		
Onsdag	21. April	1		2				3		
Torsdag	22. April	2		4				6	22	
Fredag	23. April	1		3		1		5		
Lørdag	24. April							0		
Søndag	25. April							0		
Mandag	26. April	1		4		1		6		49
Tirsdag	27. April	1		4		1		6		
Onsdag	28. April	1		4		1		6		
Torsdag	29. April	1		3		1		5	27	
Fredag	30. April	1		2		1		4		
Lørdag	1. May							0		
Søndag	2. May							0		
Mandag	3. May	1			2			3		Sprint 9
Tirsdag	4. May				3			3		
Onsdag	5. May				3			3		
Torsdag	6. May	1			6			7	25	
Fredag	7. May				8	1		9		
Lørdag	8. May							0		
Søndag	9. May							0		
Mandag	10. May				8			8		68
Tirsdag	11. May				8			8		
Onsdag	12. May				12			12		
Torsdag	13. May				12			12	43	
Fredag	14. May				3			3		
Lørdag	15. May							0		
Søndag	16. May							0		

Appendix 7 - Group Contract

Group Contract - IS305

We, the group /*Todo*/, consists of the following members:

- Ole Haraldseth
- Aleksander Larsen
- Fredrik Meltveit
- Daniel Mossestad

We are to work together with the following project: “Bachelor Thesis in Information Systems”

We promise we will:

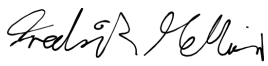
- Do assigned tasks
- Finish tasks within deadline
- Make sure work is divided equally between the group members
- Research necessary technologies and documentation needed to do assigned work

Conflicts will be resolved by discussion between the group members. If we can’t resolve the issue, the product owner will be contacted.

Signatures:

Ole Haraldseth

Aleksander Larsen



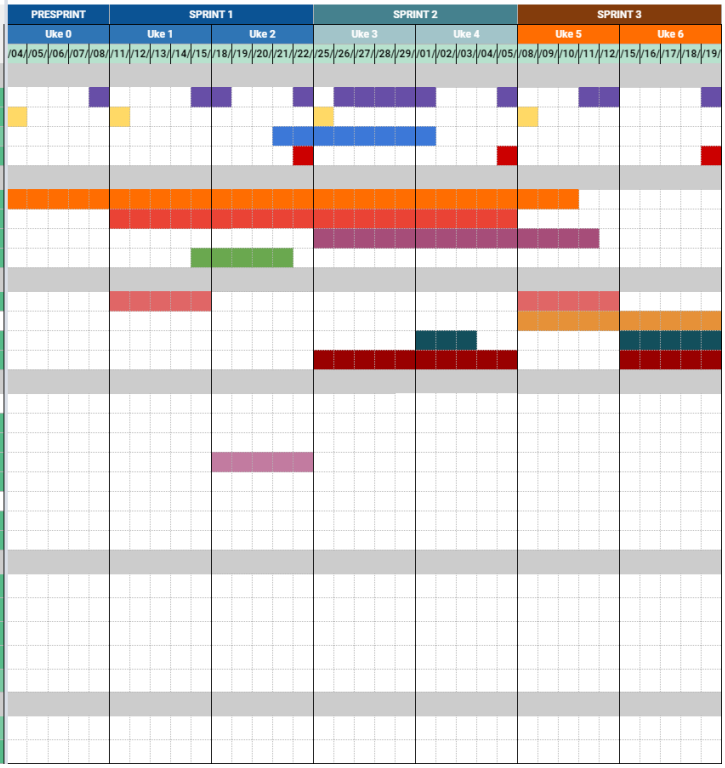
Fredrik Meltveit

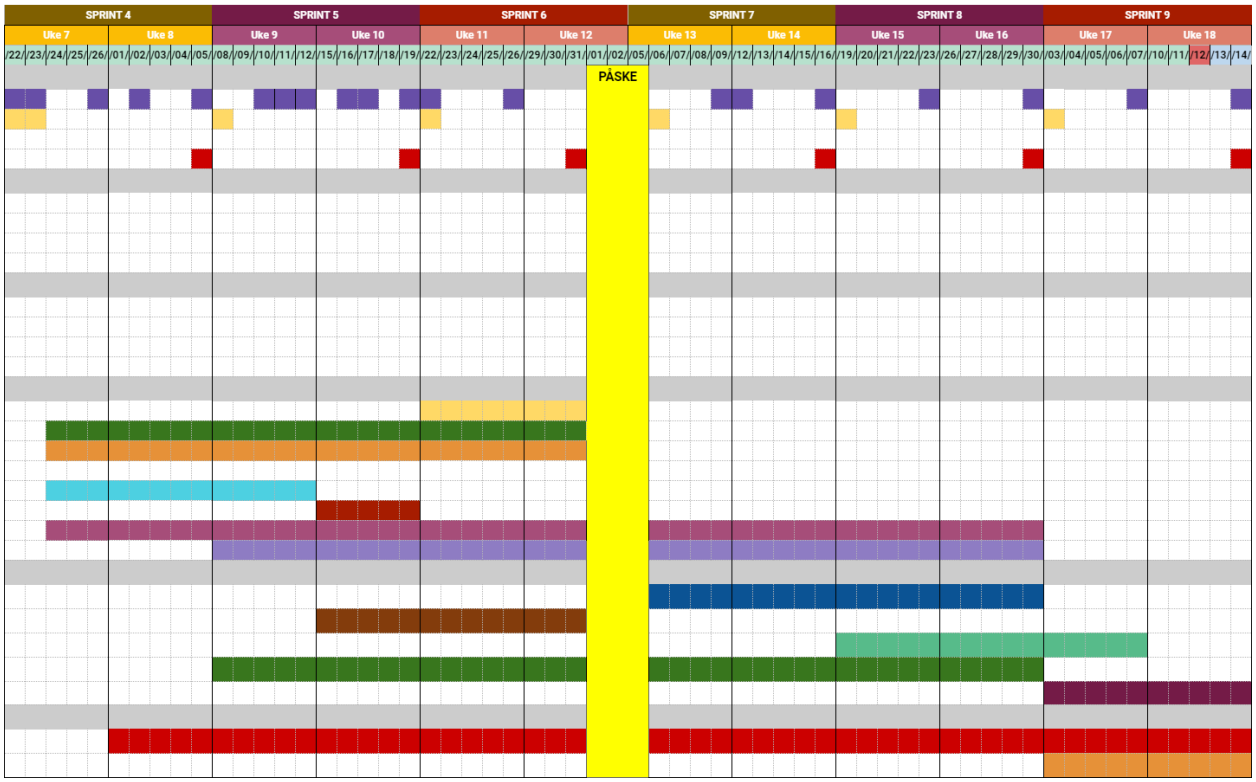
Daniel Mossestad

Appendix 8 - Gantt Chart

PROJECT TITLE	IS-304-1 21V Bacheloroppgave i informasjonssyste	**COMPANY NAME**	/*TODO*/
PROJECT MANAGER	/*TODO*/	**START DATE**	2/1/21

OPPGAVE	STARTDATO	SLUTTDATO	VARIGHET	% FULLFØRT
Planlegging				
Møte/planlegging med Sikri	1/11/21	5/14/21	123	100%
Planlegging av sprinter	1/19/21	05/20/21	121	100%
Rapportstandarder	1/21/21	2/1/21	10	80%
Sprint Retro	1/22/21	5/14/21	112	100%
Analyse				
Brukerhistorier	1/4/21	2/10/21	36	100%
Definere MVP	2/1/21	5/2/21	91	100%
Sette opp backlog / PBI (Azure)	01/25/21	2/11/21	16	100%
Risikoanalyse	1/15/21	1/21/21	6	100%
Design				
Tema	1/11/21	1/15/21	4	100%
Design av dashboard	2/8/21	2/12/21	Cancelled	0%
Design av Wizard management	2/1/21	2/19/21	18	100%
Design case display and case management	1/25/21	2/19/21	24	100%
Utvikling				
Koding dashboard	2/1/21	3/31/21	Cancelled	0%
Koding wizard management	2/24/21	3/31/21	37	100%
Koding ambulerende sjenkebevilling	1/21/21	3/31/21	70	100%
Sette opp prosjekt/rammeverk	1/18/21	1/22/21	4	100%
Koding case display and case managment	2/24/21	3/12/21	18	100%
Koding administrator settings	3/15/21	3/19/21	Cancelled	0%
Koding tilknytting med backend	2/24/21	4/30/21	66	100%
Koding global features	3/8/21	4/30/21	52	100%
Testing				
Akseptansetester	4/6/21	4/30/21	24	100%
Brukertesting design	3/15/21	3/31/21	16	100%
Brukertesting program	4/19/21	5/7/21	18	100%
Unit Tests	3/8/21	4/30/21	52	100%
Improve Report	5/3/21	5/14/21	11	80%
Leveranse				
Rapportskriving	3/1/21	5/14/21	73	80%
Presentasjon	5/3/21	5/14/21	11	100%

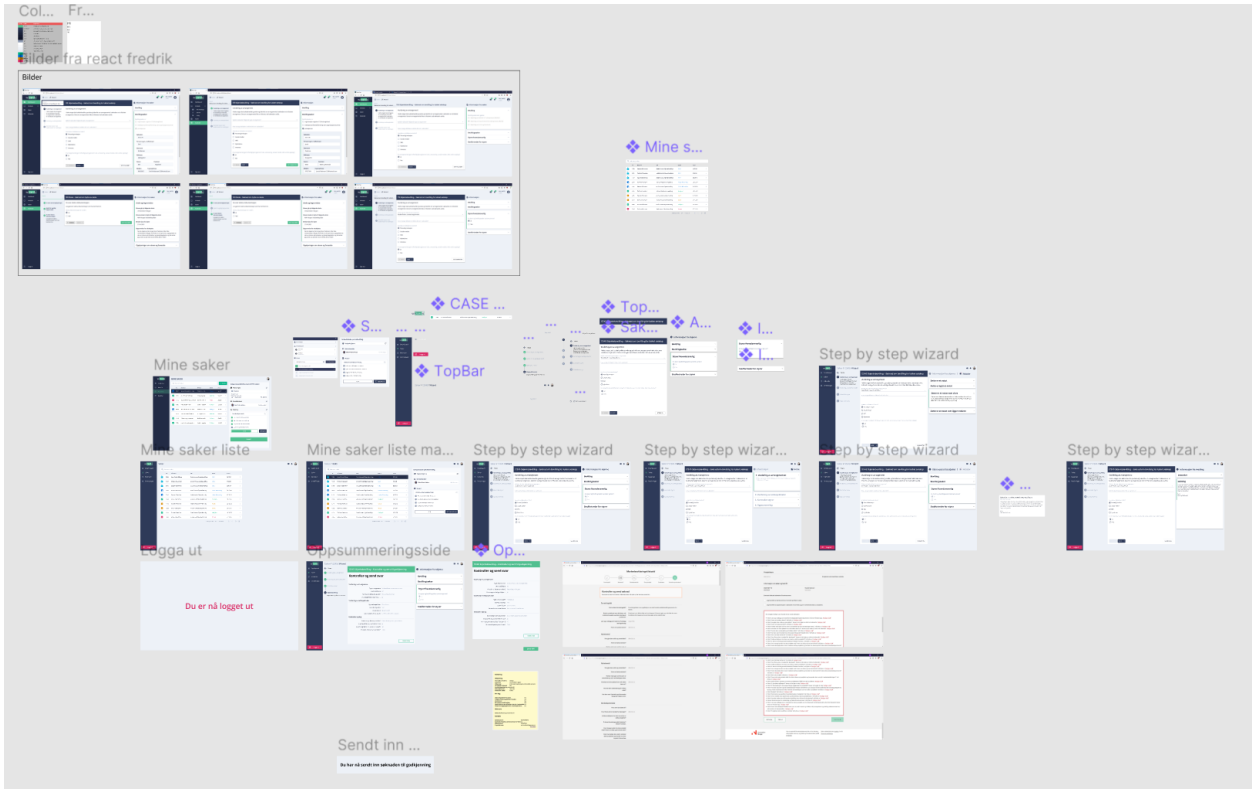



SPRINT 4
SPRINT 5
SPRINT 6
SPRINT 7
SPRINT 8
SPRINT 9
Uke 7
Uke 8
Uke 9
Uke 10
Uke 11
Uke 12
Uke 13
Uke 14
Uke 15
Uke 16
Uke 17
Uke 18
PÅSKE

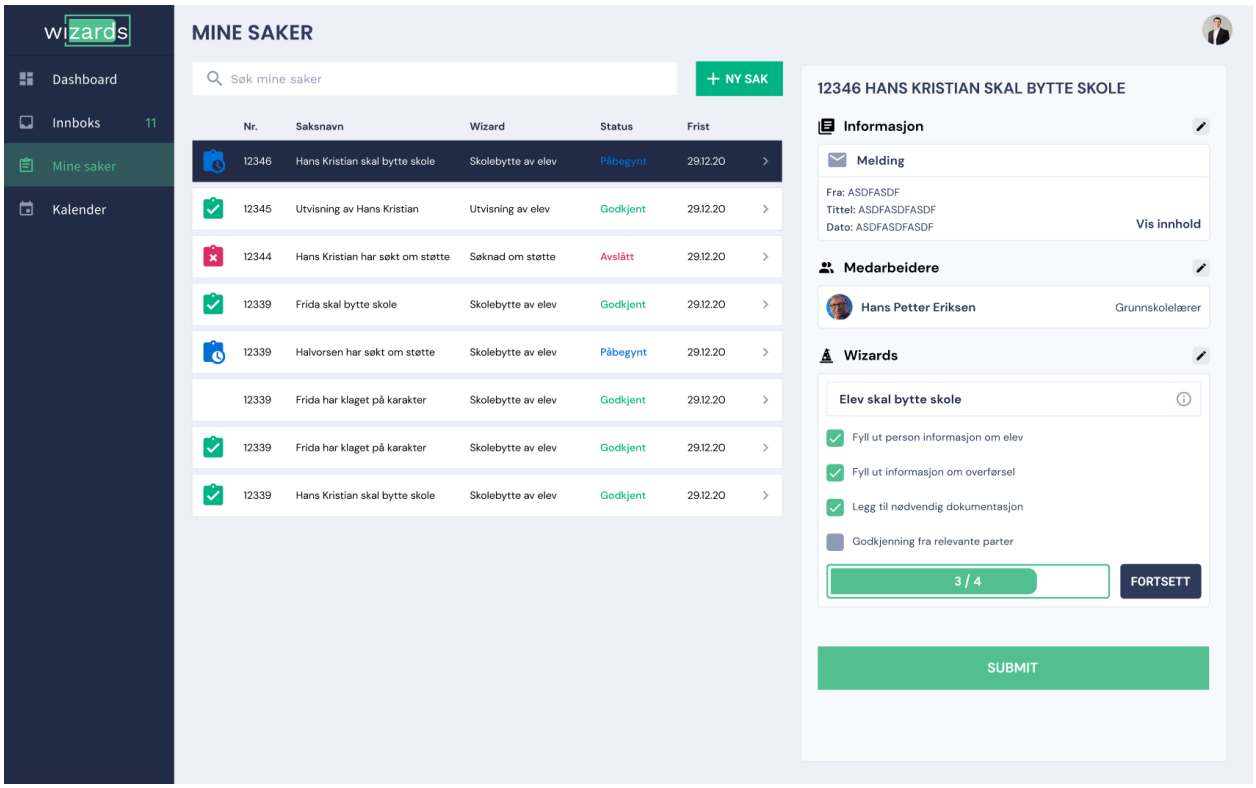
Appendix 9 - Example of User Story

5 Table display of cases
Som en saksbehandler ønsker jeg å se en liste over alle sakene mine, slik at jeg vet hvilke saker jeg skal arbeide med.
MoSCoW: Must have
Kriterie: Sakene til en saksbehandler må vises opp i en tabell.
Kommentar: Dette er hovedkomponenten som viser sakene til en saksbehandler, og går inn under saker-siden.
Akseptansekriterie: 1. En tabell over sakene til en saksbehandler med id og tittel 2. Muligheten til å navigere sider i tabellen, dersom antall saker er over f.eks. 10 (3) 3. Alternativt: Muligheten til å laste inn flere på samme side

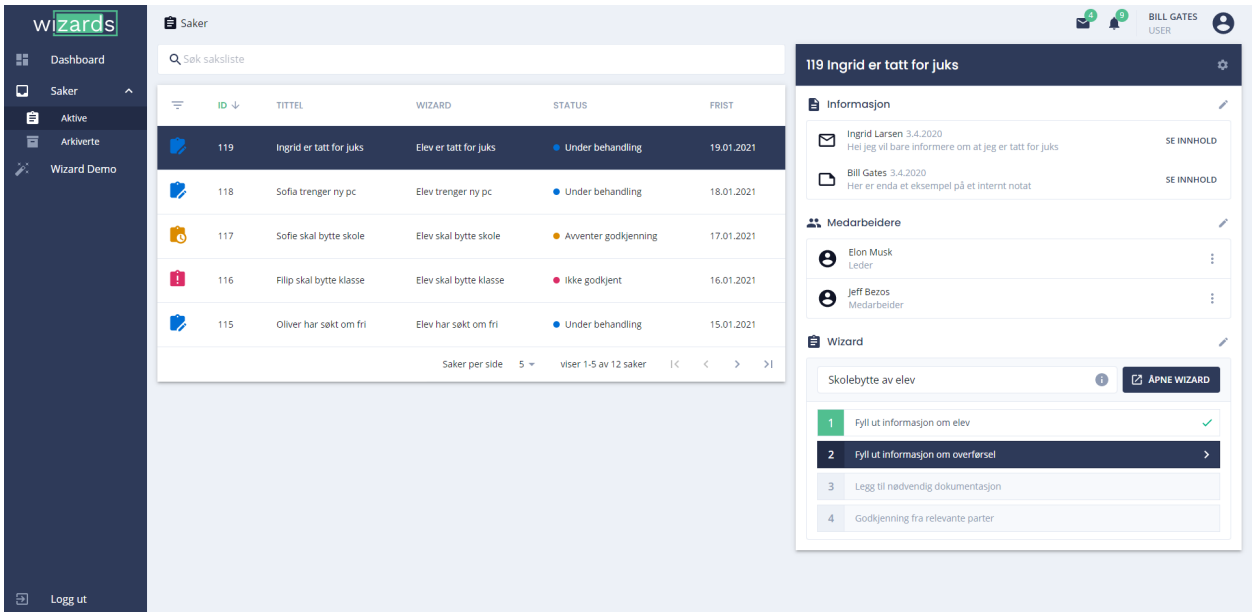
Appendix 10 - Figma Dashboard with Sketches

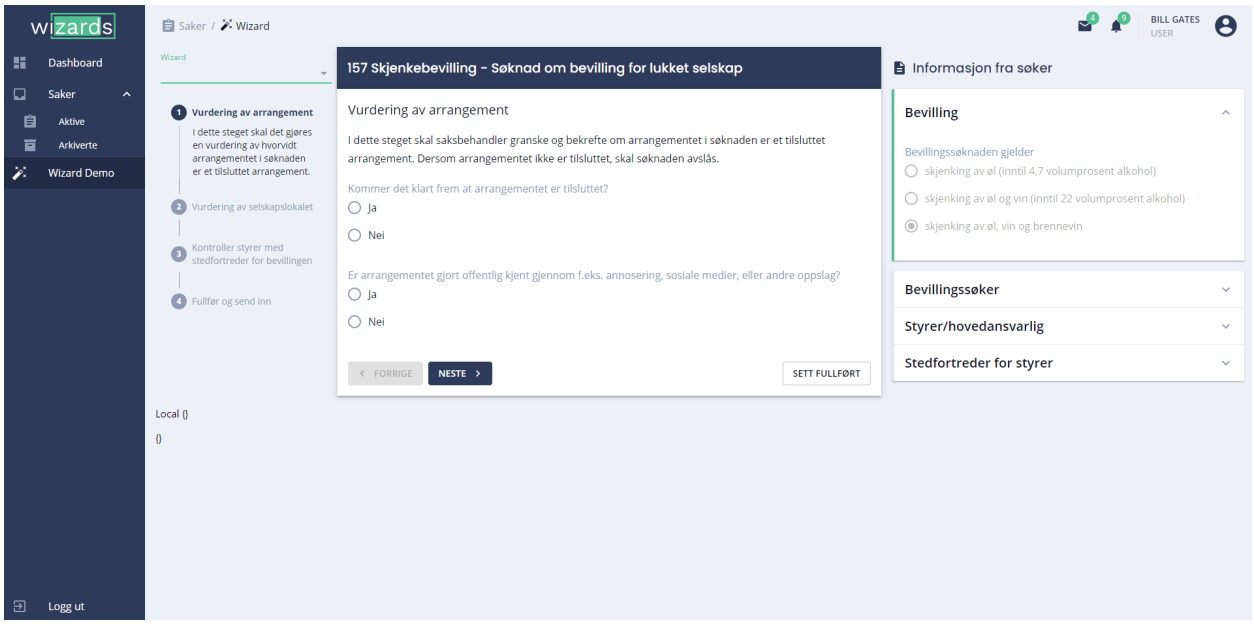


Appendix 11 – Figma Prototype

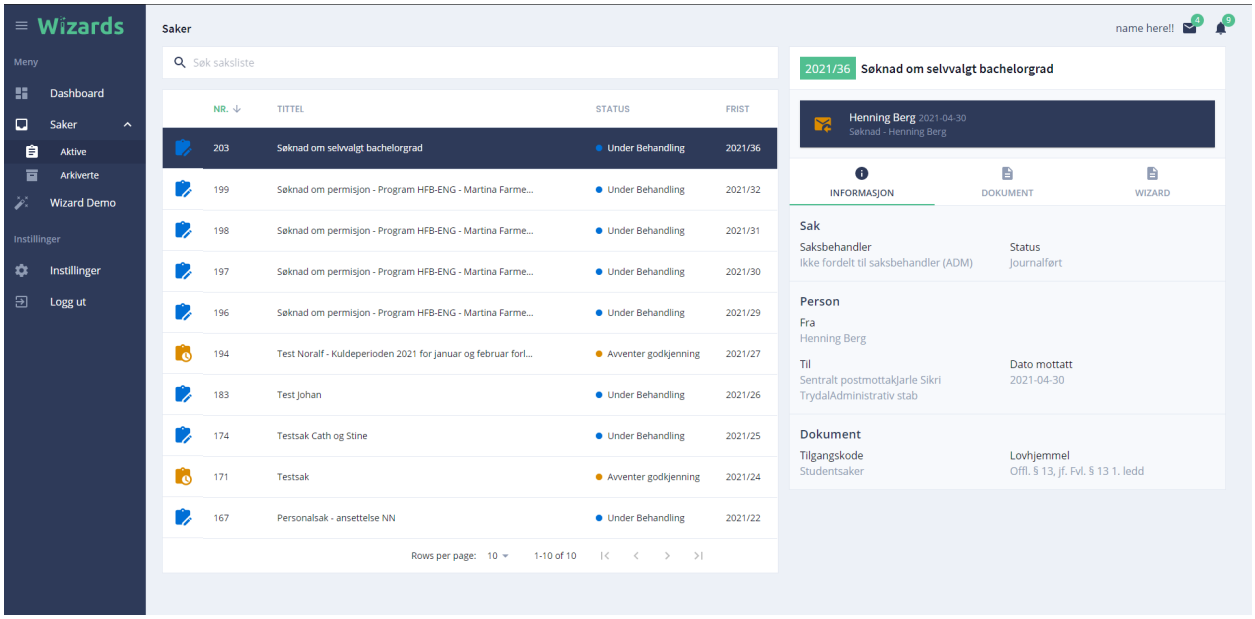


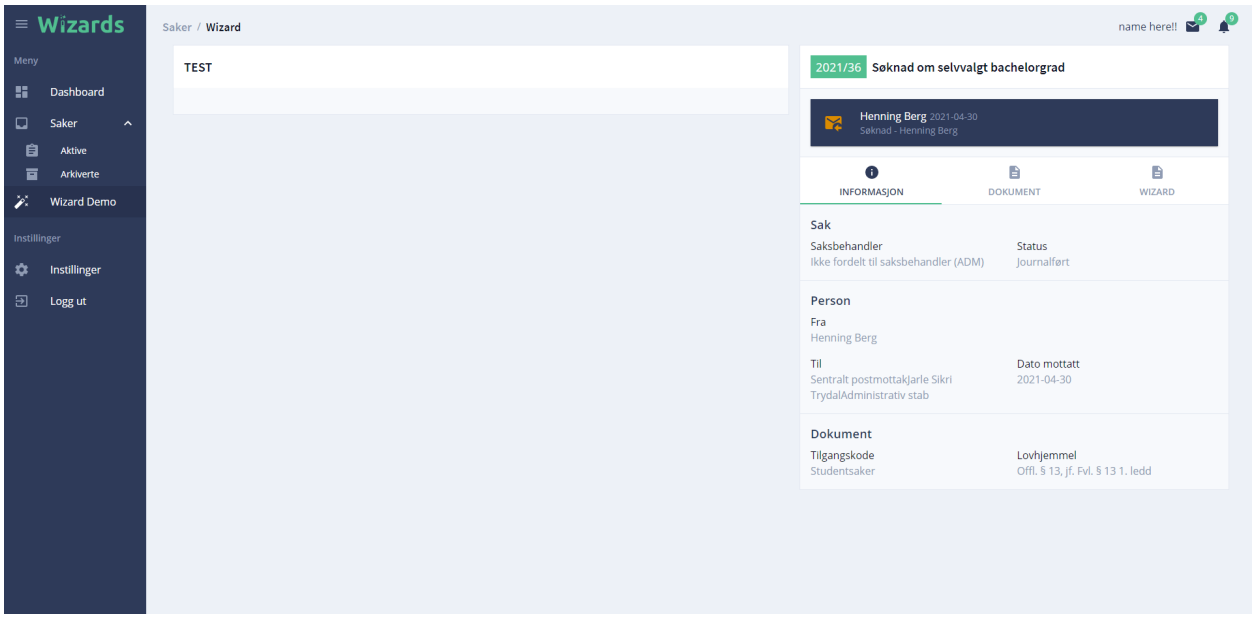
Appendix 12 – Wizard A



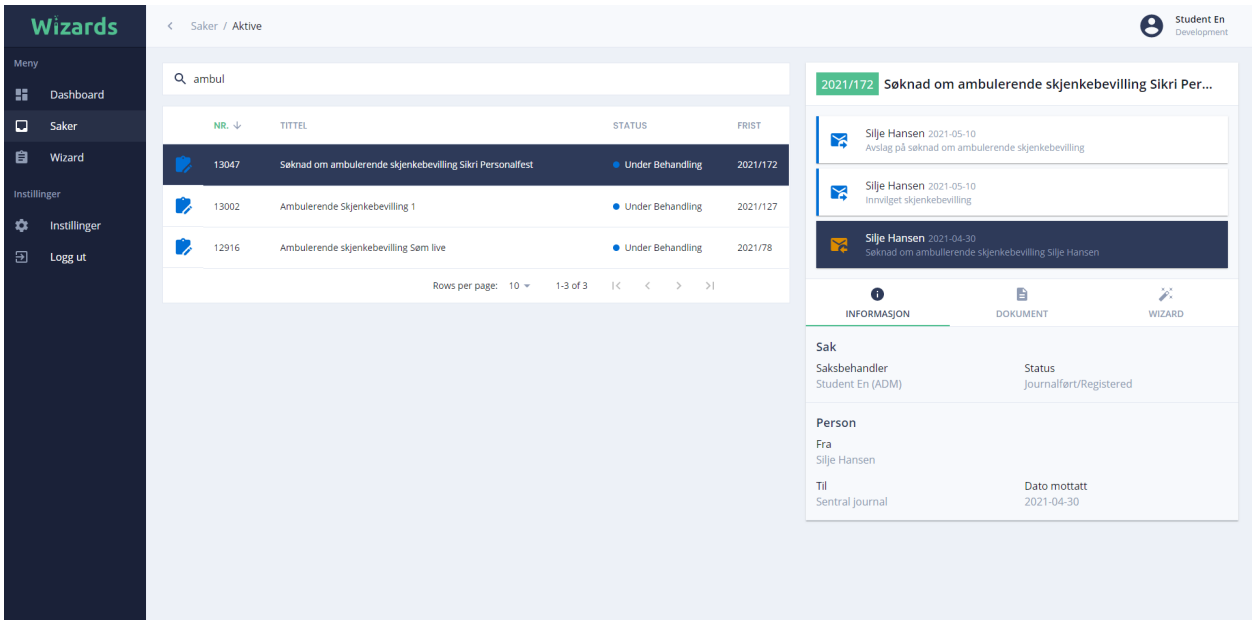


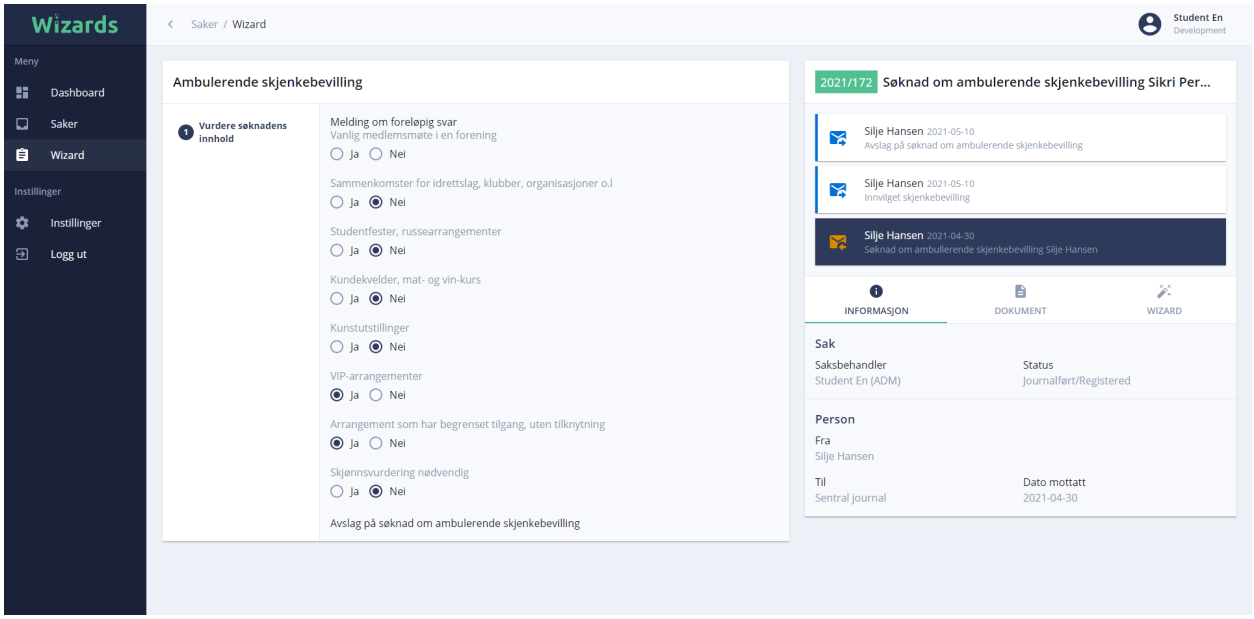
Appendix 13 – Wizard B





Appendix 14 – Wizard C





Appendix 15 – Risk Register

#	Risk	Probability	Impact	Risk Score	How to avoid risk	Limiting incident impact
1	Poor communication with client	2	3	6	Regular weekly meetings	Talk to the supervisor. Lean on group members
2	Poor communication with supervisor	2	3	6	Regular meetings	Lean more on the client and contact UiA for advice
3	Poor group dynamic	1	4	4	Do agreed work on time	Contact the client or supervisor
4	Poor motivation	2	3	6	Make small sub-goals and push each other to get to the finish line	Follow the plan and write down sub-goals
5	Bankruptcy/sale	1	3	3	--	Find a solution in collaboration with the supervisor and UiA
6	Uncertainties about design and analysis	3	2	6	Make good mockups and perform user tests	Get advice and guidance from the product owner
7	Lack of knowledge	3	3	9	Learn the technology you are supposed to work with	Get help from someone with more experience
8	Poor documentation	2	3	6	Creating a document with defined standards	Focus more on the standards
9	Too low quality of tests	4	3	12	Lear how to do testing of high quality	Well written code following standards and well structured files

10	Illness / doctor	4	2	8	Work from home, distance, hygiene	Take vitamin C and. "tran", "sana sol" and omega 3
11	Data loss	1	4	4	Cloud storage, storage routines	Retrieve from other group member
12	Internet problems	2	2	4	Keep equipment in order	Use mobile data
13	Power failure	1	2	2	Units with batteries	Use mobile data
14	Teams-problems	1	1	1	Several communication platforms	Switch to another platform that addresses the same needs
15	Getting stuck	3	3	9	Divide into smaller subtasks	Communicate with product owner or supervisor
16	Not finding the right cut-off point	3	4	12	Well established communication with product owner	Get help from product owner or supervisor
17	Technical problems	3	3	9	Have access to accessories in case something breaks	Have a good plan for how to relieve the tasks that are disturbed due to technical problems
18	Lack of feedback from users in design phase	2	1	2	Do enough user tests	Implement well-proven design principles that others have had success with in the past
19	Not maintaining scrum plan	2	3	6	Have daily stand-up meetings and work structured with the plan	High work ethic and individual structure
20	Change in client team	4	2	8	Several contact persons	Enter into a good collaboration with the new contact person

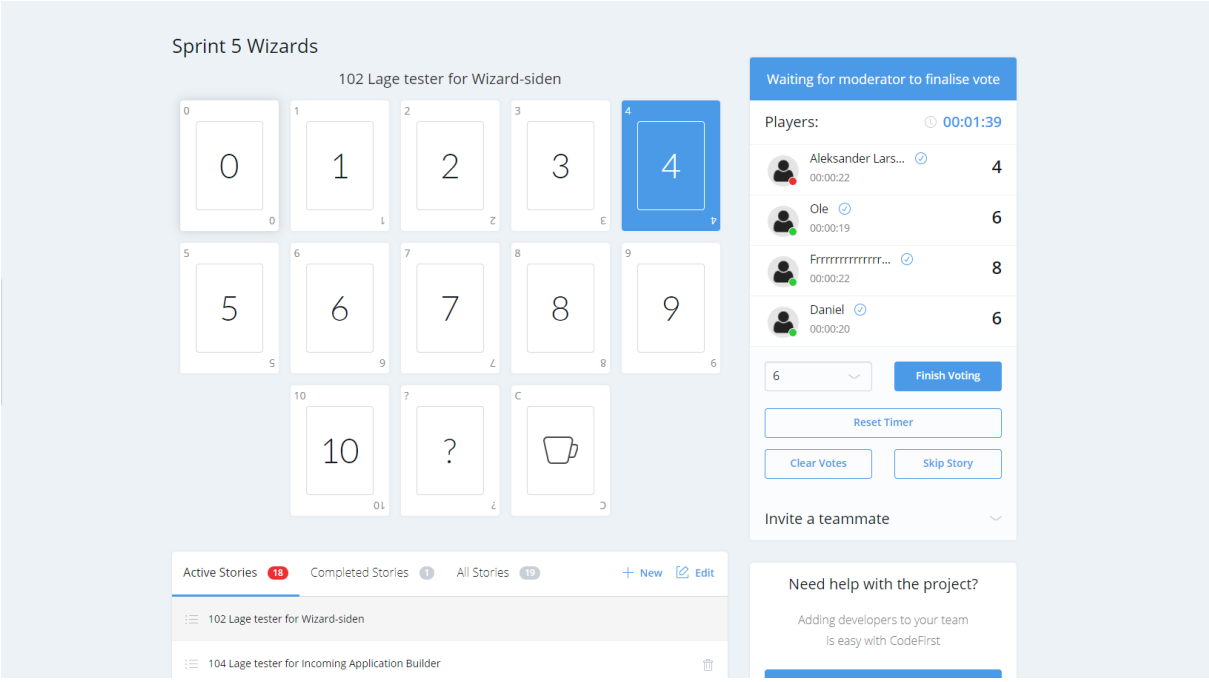
21	Product not finished at estimated time	2	3	6	Stick to plan	Good product standards
22	Insufficient information about API(s)	3	2	6	Have ongoing contact with the client regarding which API we will use	Use different API or contact contact person
23	Product not meeting requirements	2	3	6	Follow the plan and work structured throughout the entire project	Have good documentation so that the product can easily be improved by other parties
24	Group members not able to meet at scheduled meetings	4	2	8	Good routines, daily standup	The other group members needs to cover the uncovered workload
25	Technical problems with development environment	3	3	9	Have several tasks not requiring development environment	Get access to different development environment

Probability & Impact:	**Risk Score:**
1. Green Dark (Very Low) 2. Green Light (Low) 3. Red Light (High) 4. Red Dark (Very High)	1-4. Green Dark 5-8. Green Light 9-12. Red Light 13-16. Red Dark

Appendix 16 – Recorded Risks

#	Issue #	Issue/Type of Risk	Risk Score	Open Date	Close Date	Comment
1	10	Illness	8	18.01	19.01	Fredrik ill with the flu
2	20	Change in client team	8	17.02	17.02	Merethe is leaving Skiri. Got new contact persons in Sikri.
3	24	Group members not able to meet at scheduled meetings	8	25.01	25.01	Fredrik was not able to meet at scheduled stand-up.
4	17	Technical problems	9	04.02	18.04	Technical issues with Fredrik's computer
5	17	Technical problems	9	12.02	12.02	Ole's MacBook was slow and minor storage capacity. Reformatted, solved.
6	24	Group member not able to meet at scheduled meeting	8	01.03	01.03	Daniel was not able to meet at scheduled stand-up
7	15	Group member not able to meet at scheduled meeting		22.03	22.03	Fredrik was not able to meet at scheduled stand-up
8	17	Elements test environment down	9	06.04	08.04	The testing environment has been unstable, and we were unable to get the information we needed for a couple of days
9	10	Sykdom / lege	8	19.04	19.04	Daniel had a doctor appointment
10	17	Technical problems	9	30.04	30.04	Alex had a failed bluetooth module in his laptop, fixed with an external BT module.
11	17	Technical problem	9	07.05	11.05	Fredrik spilled coffe over his laptop. The laptop went black and he had to buy a new after the weekend
12	7	Lack of competence	9	01.01	14.05	The lack of competence will always be a risk, and we had to learn new technologies throughout the whole project

Appendix 17 – Planning Poker



Appendix 18 - Folder Structure

Fil-struktur

Mapper som er relevante er market i **bold**, mapper som dere ikke bør gjøre endringer i (uten diskusjon) er markert med rød

/components
/fonts
/helpers
/images
/screens
/store
/styles

/components

Components mappen holder komponenter som er brukt under skjerm komponenter, eller komponenter som er brukt globalt.

Eksempler på komponenter: “LeftNavigation” (global), “CaseItem” (underkomponent av CaseList), “CaseList” (tilhører CasesScreen)

/screens

Screens mappen brukes for å holde komponenter som holder en overordnet skjerm-komponent tilknyttet en url og side i applikasjonen.

F.eks. “MessagesScreen” eller “CasesScreen”.

For at en komponent skal vises, må du importere og plassere den under skjerm komponenten den tilhører.

Appendix 19 - React Components Guide

Funksjonsbaserte komponenter og “hooks”

Som nevnt i toppen fører bruken av funksjonsbaserte komponenter til noen endringer i metode. I React brukes noe som er kalt “hooks”, for å få tilgang til noe av funksjonaliteten som foreligger naturlig i Class baserte komponenter.

Dokumentasjon: https://reactjs.org/docs/hooks-intro.html

useState hook

useState hook brukes i følgende format:

```
const [count, setCount] = useState(0);
```

Her er `count` variabelen som holder “state” count, dvs. nummeret 0 per default. Med et call til `setCount(3)` vil verdien endres til 3.

`useEffect` hook

useEffect er en hook som dekker funksjonaliteten til `componentDidMount`, `componentDidUpdate`, og `componentWillUnmount` kombinert. Dersom du har laget funksjonalitet som gjør at en komponent må rerendere en million ganger i sekunder, har du sannsynligvis glemt å bruke useEffect hooken.

Brukergrensesnitt, stil

Liste

Material UI komponent: https://material-ui.com/components/lists/
Eksempel på steder hvor Liste brukes i applikasjonen: LeftNavigation, MessageList

Table

Material UI komponent: https://material-ui.com/components/tables/
Eksempel på steder hvor Table brukes i applikasjonen: CasesList

Wizard

Material UI komponent: https://material-ui.com/components/steppers/

Appendix 20 – Redux Guide

Redux prosessbeskrivelse

1. (types.ts) Legg til som en del av State i state interface
    a. F.eks.`selectedCaseId: number | undefined` i CasesState
2. (reducer.ts) Legg til som en del av initialState for reducer, med bestemt initial verdi
    a. F.eks. `selectedCaseId: undefined` eller `selectedCaseId: 1`
3. (types.ts) Definer typer actions som skal kunne "modifisere" staten
    a. F.eks. `export const SELECT_CASE = 'SELECT_CASE';`
4. (types.ts) Definer interface for action med type payload (data)
    a. F.eks. `interface SelectCaseAction { … , payload: number  }`
5. (types.ts) Legg til interface for action i export (med | split)
    a. F.eks. `export type CasesActionTypes = … | SelectCaseAction`
6. (actions.ts) Importer ny action type (og hvis ikke importert enda, action interfaces) til actions filen
    a. `import { … , SELECT_CASE, CasesActionTypes } from './types';`
7. (actions.ts) Legg til action som skal exportes (og importeres for bruk)
    a. `export function selectCase(caseId: number): CasesActionTypes {`
        `…`
        `}`

- (reducer.ts) Importer ny action type (se 6)
- Legg til en ny "case" i switch statementen inne i den tilknyttede reducer, i tilfeller knyttet mot den nye type action dere har lagt til
    - F.eks. `case SELECT_CASE:`
    - Returner den totale staten (...state,) og sett verdien av objektet du skal modifisere til payload (dersom payload)
    -

NB! Hvis du lager en ny reducer må du også eksportere denne til index.ts i /store, og legge den til i rootReducer.

Tilgang til og oppdatering av state fra applikasjonen

For tilgang til state til redux store, må `useSelector` fra 'react-redux' importeres. For å sette en variabel lik en state, vil format være følgende: (eksempel for caseId)
`const caseId = useSelector((state: RootState) ⇒ state.cases.selectedCaseId);`
For å oppdatere redux state fra applikasjonen, må `useDispatch` fra importeres fra 'react-redux' , og action eksportert i steg 7 må importeres fra actions.ts.

Appendix 21 – Branch Guide

1. Main
    a. Global Features
    b. Wizard Management
    c. Inbox
    d. Administrator
        i. Branch
        ii. Branch
        iii. Branch
    e. Case Display and Case management
        i. Branch
        ii. Branch
        iii. Branch
    f. Dashboard
        i. Branch
        ii. Branch
        iii. Branch

Når skal jeg merge?

Du kan merge når:

- Du har fullført en eller flere tasks
- Du har fullført en eller flere user stories

Du skal merge:

- Hver fredag dersom du har fullført en eller flere tasks og/eller en eller flere user stories

Appendix 22 – Sprint Guide

Oppsett for start av sprint

1. Oppsummering
2. Skrive inn overordnet plan
3. Velge user stories å fokusere på
4. Lage tasks
5. Estimere tasks
6. **Tilordne tasks**
7. Oppdatere Gantt chart
8. Se kjapt igjennom dokumentene (Riskanalyse, timeliste, annet)

- Etter sprint 4: Legge til ekstra tasks/user stories dersom tasks/user stories er avhengige av informasjon fra Sikri

- Sprint 5:
    - Legge til testing
    - “Fuck-around”-ing i forhold til backend
    - Lage 10-20 saker for student1 og student2
    - Fikse småting rundt om i applikasjonen
        - Her kan det gjøres individuelle vurderinger, og pushes til en branch spesifikt for dette
    - Opprette grensesnitt for wizard template building
    - Planlegg presentasjon
        - “Storyline” hvor en saksbehandler håndterer en sak gjennom en Wizard
            - Bruk “Ambulerende Skjenkebevilling”
    - Les gjennom mal for oppsett av “Arbeidsflyt”

Oppsett for avslutning av sprint

1. Gjennomgå Azure
2. Merge
3. Sprint Review
4. Sprint Retro

Appendix 23 – Example of Unit Test

```
describe("<LoginScreen />", () => {
  let component;

  beforeEach(() => {
    component = shallow(<LoginScreen />);
  });
  test("It should mount wtihout errors", () => {
    expect(component.length).toBe(1);
  });

  describe("<App />", () => {
    let component;

    beforeEach(() => {
      component = shallow(<App />);
    });
    test("It should render <LoginScreen /> when user not set", () => {
      component.setState({ user: false });
      expect(component.find(LoginScreen).length).toEqual(1);
    });
  });
});
```

Appendix 24 – Overall Plan

SPRINT 2	Uke 4			JANUAR			
	man 1/25/2021	tir 1/26/2021	ons 1/27/2021	tor 1/28/2021	fre 1/29/2021	lør 1/30/2021	søn 1/31/2021
Aleksander	Møtereferat	Arbeidsstandarder					
Daniel	Brukerhistorier rangering	Arbeidsstandarder					
Fredrik							
Ole	Finne ut av Azure Boards	Arbeidsstandarder					
Alle	**AzureDevOps/Backlog/ Arbeidsstandarder**	**Forelesning IS-304 9:15-12:00 Møte med Anne Mette 8:30-9:00 Design**	**Gruppemøte ang. Azure og Kodestandarder: 10:00-12:00 Forelesning IS-305 14:15-16:00 Brukerhistorier og Azure devops backlog**	**Møte med Sikri om Brukerhistorier 13:30-14:30 Brukerhistorier og Azure devops backlog**	**Møte med Sikri 12:00-12:30 Planlegge møte og brukerhistorier**		

	Uke 5			FEBRUAR			
	man 2/1/2021	tir 2/2/2021	ons 2/3/2021	tor 2/4/2021	fre 2/5/2021	lør 2/6/2021	søn 2/7/2021
Aleksander							
Daniel		Kode/design stepper	Kode/design stepper				
Fredrik							
Ole							
Alle	**Brukerhistorier Merethe Planlegge for uken Fikse standarder Møte med Merethe 12-13**	**Forelesning 9:15-12:00 Begynne på koding dersom godkjent**	**Forelesning IS-305 14:15-16:00 Koding**	**Kodeing**	**Sprint review Møte angående testing standarder internt**		

Appendix 25 - Steering Committee Meeting 1

Referat fra styringsgruppemøte 1

Møtenummer	1
Dato	19.02.2021
Møteform/Sted	Microsoft Teams
Møte startet	14:00
Møte hevet	15:00
Til stede	Ole Haraldseth Fredrik Meltveit Daniel Mossestad Aleksander Larsen Sikri: Marius Holen, Fredrik Werpen UiA: Janis Gailis

Temaene som ble presentert i møtet:

- Planlegging / Verktøy
- Brukerhistorer
- Azure DevOps
- Design
- Kode
- Utfordringer
- Planen videre

Det ble lagt frem at det har vært en tidkrevende å få laget ferdig brukerhistoriene med oppdragsgiver. Etter at vi trodde vi var ferdig med dette, kom det beskjed fra ledelsen til oppdragsgiver om at prosjektet skulle spisses ytterligere. Prosjektet gikk altså fra å omhandle alle typer saker til kun å omhandle saksbehandlingen av ambulerende skjenkebevilling.

Planen videre:
I hovedsak ble det presentert at vi skulle fortsette å jobbe med de administrative verktøyene vi hadde, ferdigstille design, teste det mot kundegrupper og kode produktet. En av de viktigste punktene fremover var at vi skulle begynne å skrive på rapporten. Vi kommenterte at vi ikke hadde skrevet så mye på denne foreløpig fordi vi ønsket "litt mer kjøtt på beinet" før vi begynte skrivingen, noe som vi mener at vi nå har.

Aksjonspunkter:

Det ble diskutert noen forbedringspotensialer i møtet. Disse gikk i hovedsak ut på å synliggjøre og synkronisere timer brukt på forskjellige oppgaver som er lagt in Azure devOps og i timelistene vi har i Google Drive. Allikevel ble måten vi hadde gjort det på ansett som bra nok og det var få forandringer som måtte gjøres.

Appendix 26 - Steering Committee Meeting 2

Referat fra styringsgruppemøte 2

Møtenummer	2
Dato	09.04.2021
Møteform/Sted	Microsoft Teams
Møte startet	13:00
Møte hevet	14:00
Til stede	Ole Haraldseth Daniel Mossestad Aleksander Larsen Sikri: Fredrik Werpen UiA: Janis Gailis

Temaene som ble presentert i møtet:

- Planlegging
- Brukerhistorer
- Azure DevOps
- Design
- Kode
- Testing
- Fagkveld
- Utfordringer
- Planen videre

Det ble snakket om planlegging og at vi hadde begynt med planning poker som et verktøy til planleggingen i begynnelsen av en sprint. Videre viste vi til oppdatert Gantt Chart og at vi nå hadde begynt å ta kopi av de gamle chartene mellom hver sprint slik at man kunne se utviklingen fra sist sprint. Videre pratet vi om timelogging, arbeidsfordeling, mappe struktur på drive, discord, Microsoft Teams og det videre samarbeidet med Sikri.

Det ble også vist fram en rapport som var laget tidligere av Sikri, som inneholdt analysen som prosjektet og brukerhistorier var basert på. Oppdaterte brukerhistorier ble også vist da disse hadde trengte ytterligere spissing. MVP var ferdig definert og utvikling var underveis.

AzureDevOps ble vist frem, backlog med tasks og burndown charts. Figma (sketcher) med ferdig design ble vist. Kodestruktur ble vist frem og noe ferdig frontend ble vist. Kontakt med Sikri sin backend ble også vist frem.
Første test med en kunde (saksbehandler) var også blitt gjennomført og dette ble diskutert. Det ble også snakket om enhetstesting og at de første var ferdig.

Videre ble det tatt opp at vi hadde vært med på en fagkveld med hele Sikri. Der hadde vi vist fram en demo av hele prosjektet. Dette ble tatt veldig godt imot av ledelsen og ansatte i sikri.

Planen videre:
Først og fremst å skrive på rapporten er viktig, fortsette å oppdatere gantt chart, bruke administrative verktøy, jobbe med tilbakemeldinger fra brukertesting og videre testing av programvaren.

Aksjonspunkter:
Ingen videre aksjonspunkt ble nevnt denne gangen. Det var enighet om at vi skulle fortsette å forbedre oss på timeberegning i Azure DevOps.

Appendix 27 - Steering Committee Meeting 3

Referat fra styringsgruppemøte 3

Møtenummer	3
Dato	07.05.2021
Møteform/Sted	Microsoft Teams
Møte startet	8:30
Møte hevet	9:30
Til stede	Ole Haraldseth
	Fredrik Meltveit
	Daniel Mossestad
	Aleksander Larsen
	Sikri: Marius Holen, Fredrik Werpen
	UiA: Janis Gailis

Temaene som ble presentert i møtet:

- Intro
- Innspurt
- Wizards demo
    - Refactor
- Avslutning

Vi er nå i siste innspurt av prosjektet og det ble presentert status av relevant dokumentasjon som skal være med i rapport. Refactor av kode og design ble gjort. Design av applikasjonen er endret som gir mer responsivt og intuitivt system samt bedre performance. MVPen er nå fullført og demo er vist frem til veileder og relevante personer i prosjektet. Demoen er på en server og tilgjengelig for alle med link og login detaljer.

Planen videre:
Planen videre er å skrive rapporten som skal leveres innen 14. Mai. i tillegg til å fullføre eventuelle dokumenter som trengs i forbindelse med rapporten. Det er også litt rom for små endringer i koden om nødvendig før handover 19. mai. Det skal også utføres en siste brukertest med en saksbehandler.

Aksjonspunkter:
Ingen konkrete aksjonspunkter kom frem under møtet, men generell tilbakemelding på hva refleksjonsdelen bør inneholde i rapporten.

- Må huske å reflektere over hvordan prosjektet har fungert.
- Velge relevant dokumentasjon som vi viser til i rapporten.
- Få frem endringer og forbedringer gjort mot slutten.