Elements Notification System

A bachelor thesis in collaboration with Sikri AS

DOKKEN, A.

HUSFLOEN, L.

STOKKELAND, K.

SØMME, B.

THEIMANN, H.

SUPERVISOR

Geir Inge Hausvik

University of Agder, 2023

Faculty of Social Sciences

Department of Information systems

Abstract

This report documents a software development project conducted in collaboration with Sikri during the spring semester of 2023. The objective was to implement a modern notification system within their already existing product, Elements. With the objective of ensuring high processes quality, the team took advantage of established project management methodologies. The team utilized Azure DevOps as a tool for controlling and carrying out these management processes.

Through data collection, the team gathered insights of the relevant domain and conducted a user survey to further understand the user needs. This was crucial for the subsequent process of specifying the system requirements. An architectural model was formed as an outcome of utilizing established design principles and discussions with technical leaders at Sikri. The model and system requirements were used in hands with Sikri's proposed user interface design as a foundation for developing the system.

With the use of technologies such as C#, TypeScript, React, and SignalR, the team developed a highly accurate and effective system in correspondence with Sikri's ambitions. While not covering all the requirements to make the notification system production ready, it fulfilled the predetermined proof of concept. By highlighting quality over quantity throughout the span of the project, the team successfully facilitated further development.

As a result of carrying out the project, the team has built meaningful experience within management and development. Through developing a product of high quality with the intention of going into production, the challenges and processes have brought meaningful learning outcomes within the field.

The final product can be viewed here.

Preface

This report is written as a part of the bachelor's degree in IT and information systems, for the subject IS-304, bachelor's thesis in information systems. Through this bachelor's thesis and report, the aim is to demonstrate the students' ability to use acquired knowledge and competence through the execution of a real project associated with information systems (University of Agder, 2021).

Our sincere thanks to Asbjørn Nordgaard & Sikri for including us extremely well during the project – offering a challenging project and giving the necessary help and guidance for our team to excel at it. We especially appreciate being able to work from Sikri's offices alongside their development teams, facilitating communication, learning and collaboration. Thanks to everyone at the Kristiansand office for welcoming us!

We would also like to express our gratitude to all who contributed to this project. Especially to Ammar Haddad as our primary contact and Eren Canpolat, Leif Roger Frøysaa, Tom Are Nyland, Ingebjørg Gregersen, Caroline Andersen, and others for collaborations and mentoring. Lastly, we would like to thank our supervisor, Geir Inge Hausvik – for valuable feedback and advice throughout the project.

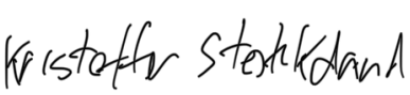
Kristiansand

16.05.23

Lars Blåsmo Husfloen

Bjørnar Olsen-Hagen Sømme

Aleksander Dokken



Kristoffer Stokkeland

Hermann Rødseth Theimann

Table of Contents

List of Figures

List of Tables

List of Appendices

1. Introduction

This report is the documentation of the management, execution, and conclusion of the conducted project. As a part of the bachelor's thesis, it is mandatory to collaborate with a third-party, which is why a partnership with Sikri AS was formed. The team was tasked with developing a notification system for Sikri's case management system, Elements, entirely changing the way users get notified.

The first chapter will introduce the project, while the second chapter covers the management of the project. The third chapter pertains to the analysis conducted, whereas the fourth presents the technologies and tools used. Architectural design will be presented in the fifth chapter, before chapter six describes the projects execution. Chapter seven presents the final product, while the reflection of the project comes in chapter eight. The ninth and last chapter will conclude the project.

The following subchapters in the introduction will introduce relevant definitions, Sikri as a company, the team, current situation, the project, as well as goals and ambitions.

1.1. Definitions

This report will expect some familiarity with the following terms and therefore offer some quick definitions for the reader.

- **Monolith**: "A monolithic architecture is a traditional model of a software program, which is built as a unified unit that is self-contained and independent from other applications." (Harris, u.d.).
- **Micro service**: "an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API." (Lewis & Fowler, 2014).
- **Micro frontend**: "An architectural style where independently deliverable frontend applications are composed into a greater whole" (Jackson, 2019).
- **Monorepository**: "A monorepo is a single repository containing multiple distinct projects, with well-defined relationships." (Eagle, et al., 2022).

1.2. Sikri

Sikri AS is one of Norway's leading providers within case management, document management, and archiving. With a rich history of over three decades, Sikri has a strong focus on fostering collaboration and innovation to help its clients achieve their goals. As the project owner, Sikri has put aside resources for sustaining the bachelor project.

1.3. Team

The team consists of five members, all in their sixth and final semester of the bachelor's degree in IT and Information Systems at the University of Agder. After working together in several previous courses, the team has developed a strong and collaborative relationship.

Furthermore, each member possesses comprehensive knowledge of different aspects within software development, with various areas of expertise. There was a division of primary responsibilities, but despite this, all members were involved in the different aspects of the project. This division specifically referred to frontend and backend. Lars and Hermann took the main responsibility for the frontend, while Aleksander, Bjørnar, and Kristoffer had the backend. Hermann was appointed as the team leader, but as a flat hierarchy was preferred, most of the responsibilities ended up being shared amongst the team. The role of Scrum Master was performed by Aleksander throughout the project.

1.4. Current Situation

Elements, Sikri's case management system, is a popular tool for handling, managing and archiving cases in the public sector, with well over 50 000 regular users. In recent years, there has been a growing demand for greater collaboration and case processing efficiency. Elements has some functionality to support notifying users via mail once a day on what has changed in a “search” or case. The users are not receiving timely notifications about important case updates and events as they occur, leading to delays and potentially missed deadlines. As a result, it has become apparent that the current system needs upgrading when it comes to notifying users of significant events and case-related updates.

1.5. Project

To address the issues mentioned in the previous chapter, the team partnered with Sikri to develop a new notification micro service and micro frontend that will seamlessly integrate with their existing application environment. The system will offer notifications about relevant events when they occur. Examples of such events include changes or updates to a case, upcoming deadlines, or system announcements from Sikri. The system will exist as a part of Elements, and Sikri will be responsible for the UI/UX design ensuring consistency, facilitating that the project have greater focus on functionality.

From using the notification system, users will be able to stay up to date on case developments without having to manually explore changes in the Elements UI. This will allow users to focus on other important tasks while also ensuring that they do not miss critical updates. The system will also have customizable settings, giving users control over events they want to be notified about. By providing these new functionalities, the notification system aims to improve efficiency and user experience.

The notification system will have robust security to ensure that only authorized users have access to sensitive case information. The user interface will be intuitive and easy to navigate, with clear and concise information presented in a user-friendly manner. The notification system will be scalable, reliable, and integrate with existing case management tools and systems. Clear documentation will be provided, making the handoff to Sikri’s developers easier after project close.

The initial scope of the project is to create a proof of concept (POC) that fulfill the criteria specified to achieve the specified minimal viable product (MVP):

- Users should be able to receive notifications in the Elements UI when relevant events occur.
- Users should be able to change their notification settings, such as choosing which type of events they want to be notified about.
- The system should have a user-friendly interface that is easy to navigate and provides clear and concise information.
- The notification system must be able to integrate with Sikri's case management system, Elements.

1.6. Goals and Ambitions

The overarching project goal is to facilitate our academic and professional growth while also delivering a high-quality outcome to the partner company, Sikri. The team is deeply invested in this project and committed to carrying out the work in a professional and rigorous manner. Producing high-quality code that adheres to best practices and using modern methodologies for both project management and technical excellence is a priority. Especially as this system is planned to be used in production, it is important to be mindful of Sikri's preferences and requirements, particularly their emphasis on quality over quantity. Therefore, the team will make well-reasoned decisions that consider their standards and expectations, ensuring that the final delivery meets the needs and is ready for further development and implementation.

2. Project management

Effective project management plays an important role in ensuring a successful project process. It acts as the binding agent that holds the project together, providing the team with realistic plans, well-defined objectives, estimates, and control (Aston, 2023). This chapter is divided into four subchapters showcasing the different aspects of the project management. The first subchapter outlines project methodology, while the following three will draw attention to quality, with the aim to ensure this remains highlighted throughout the project.

2.1. Methodology

As there will always be differences between projects, there is rarely a standardised solution on how to structure project work, but more of a process for the team to find what ensures the best prerequisite for success (Burgan & Burgan, 2014). At the heart of agile, are open communication, collaboration, adaptation, and trust amongst team members (Atlassian, u.d.).

For agile management it was decided to use a combination of the Scrum framework and Kanban method, Scrumban, as the development methodology. “Scrum is an agile project management framework that help teams structure and manage their work through a set of values, principles, and practices.” (Atlassian, 2018), while “Kanban is a popular framework used to implement agile and DevOps software development.” (Atlassian, 2019).

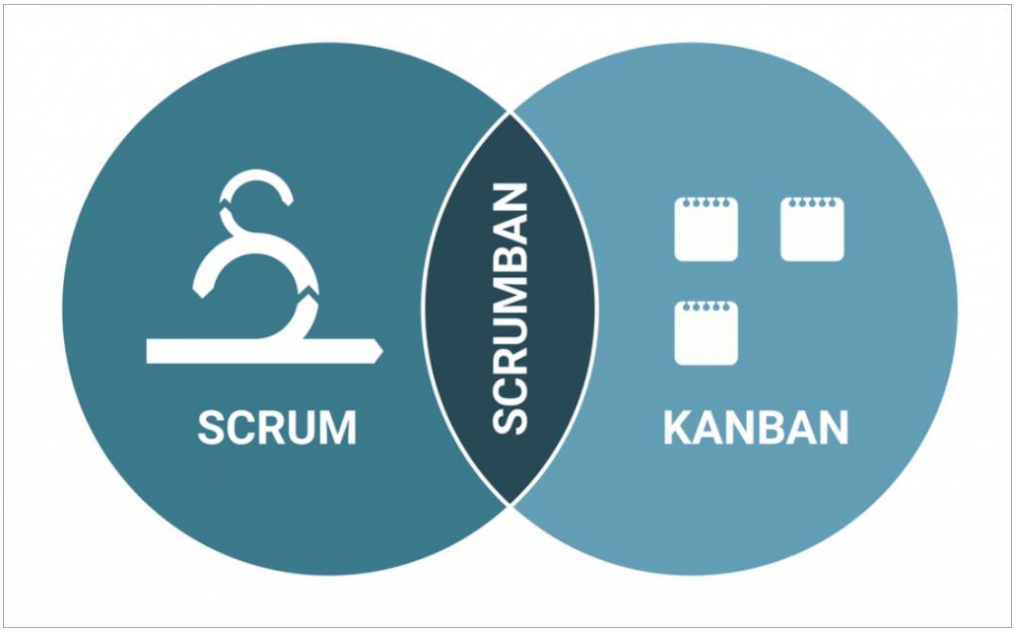


Figure 1: Venn diagram illustrating Scrumban (Kajal, 2021)

Derived from utilizing Scrumban in prior development projects, the team found it appropriate to use Scrum artifacts that endorse control and quality, in combination with a Kanban board. Kanban boards are a shared space where teams can visually manage their work, and using this will give the team a better overview and control over the work items in the project (Lynn, u.d.). Additionally, the kanban inspired “walk the board”, introduced to the team by Skatteetaten in an earlier project, will be used during daily scrums to easier follow up work in progress. This approach to daily scrum entails to change the focus from the people to the tasks on the board, focusing on work items on the kanban board going from person to person (Chec, 2020). Using a combination of Scrum and Kanban artifacts, allowed the team to work in an iterative and flexible way, adapting to changes as they arise and ensuring that the final product meets the necessary quality requirements. Having such an approach was especially important in this project as the scope was quite fluid, meaning there was a significant number of changes to accommodate for as the project went on.

Since Sikri utilize a similar Scrumban approach to what the team have used in prior projects, there were no problems adapting to their workflow. Working in a similar fashion to Sikri was deemed sensible, as this would make collaboration and further development easier. The sprints for this project were conducted in 2 week increments and there was held daily stand-ups to keep everyone updated on the progress and bring up any issues that arose. At the start of every sprint, the team carried out a Sprint Planning meeting to specify the goal of the sprint and the associated tasks in which were estimated.

Towards the end of the sprints, a Sprint Review with the Product Owner or Product Owner representative was held for reviewal and feedback, with the intent of preventing challenges and improving the processes. Subsequently to closing off the sprint, a retrospective was held, further evaluating the progress, and identifying areas of improvement. The retrospective served as an opportunity for the team to reflect on the project and its outcomes, including any incidents or issues that occurred during the sprint. In addition, the team followed the principles of blameless postmortem, which is a process of analysing and learning from incidents without attributing fault to individuals (Atlassian, 2020), it focused on identifying the root causes of any challenges and implement preventive measures. Its purpose was to enable the team to learn from the incidents without pointing fingers or assigning blame to individual team members. This fostered a culture of open communication, collaboration, and trust among the team, allowing for continuous improvement and growth.

The team also worked closely with Sikri throughout the development process to effectively prioritize and address any issues, ensuring that the system would fulfil their requirements. Regular meetings and demos were conducted to gather feedback and make necessary adjustments, contributing to the prevention of challenges and improvement of processes for upcoming sprints.

2.2. Definition of Quality

Defining quality in software development can be a complex task as it depends on the distinct requirements and expectations of the end-users and stakeholders (xbosoft, u.d.). However, most definitions share a fundamental aspect, which is the extent to which a software product or service satisfies these demands and anticipations. While each project may require a unique definition of quality based on its specific context and needs, using an established guideline can facilitate communication and collaboration among team members and stakeholders (McKinsey & Company, 2021). Therefore, to ensure consistency in quality, the team has chosen to use the following definition of software quality from the ISO 25010 standard as the base for the project:

"The quality of a system is the degree to which the system satisfies the stated, and implied needs of its various stakeholders and thus provides value" (ISO, u.d.).

This definition is a part of a comprehensive and standardized framework for assessing and evaluating software quality, which is widely recognized and accepted in the industry (ISO, 2015). Furthermore, quality extends beyond the end-product and encompasses the techniques and methodologies employed throughout the software's development and delivery process (Indeed, 2022). This includes a range of activities such as planning, execution, monitoring, and control, all of which contribute to the product’s overall quality.

In order to achieve high-quality software, it is crucial to gain a good understanding and define the requirements and expectations of the end-users and customers (Indeed, 2023). This can be accomplished through various methods such as conducting user research, gathering feedback from existing customers, and analyzing industry trends and best practices. At its core, agile

development emphasizes the need to gather just enough information to start a project, enabling early testing to ensure value delivery while avoiding the pitfalls of overdesigning and costly mistakes (Atlassian, 2023). Furthermore, the principles of agile development encourage a culture of rapid iteration, embracing failure as a learning opportunity and moving forward with continuous improvement (Agile Alliance, 2015).

"Quality is never an accident. It is always the result of intelligent effort."

- John Ruskin

2.3. Process Quality

Attaining and maintaining high process quality is crucial in software development projects and is often referred to as quality assurance (Stanton, 2022). Achieving high process quality requires careful consideration of the methodology and accompanying processes, ensuring that they best align with the specific needs and goals of the project. Equally important is considering the team and its individual members, as their capabilities and preferences can significantly impact the effectiveness of the chosen processes. By selecting appropriate methodology and processes that are well-suited to the project and the team, it becomes possible to establish and maintain high process quality throughout the project's lifecycle (Krawczyk, 2022).

Therefore, as mentioned, the team is using an agile methodology inspired by Scrum and Kanban, which emphasizes continuous improvement and adaptation, allowing the team to work in short sprints, communicate and collaborate effectively, and address issues and risks promptly. This approach has been adjusted to fit the team and its members, ensuring that it fits the specific needs.

In addition to the traditional ceremonies associated with the chosen methodology, the team has implemented processes to complement them, ensuring that the work adheres to the definition of done (DoD), which is found in Appendix 8. These processes include code reviews, acceptance criteria, pull requests, and testing. A clear DoD is essential for any project as it establishes the criteria that must be met to consider a task complete. By having a well-defined DoD, the team can ensure that all aspects of the project are completed to a satisfactory level, minimizing the risk of rework, and improving overall efficiency (ProductPlan, 2021).

The team is working at the office roughly three times a week, which facilitates easy access to Sikri's resources and enables the team to work more closely together. As a result of this, pair programming has been utilized frequently. This has been shown to result in higher code quality, faster problem-solving, and improved knowledge-sharing among team members (Przystalski, 2021). Additionally, team assessment in Azure DevOps is used at the end of each sprint, allowing the team to reflect on their work and assess the team's health, performance and identify areas for improvement. Based on this, the team can make the necessary adjustments to ensure continuous improvement.

The use of appropriate tools and communication channels is also critical to achieving high process quality (Nath, 2023). Thus, the team utilizes Teams and physical meetings as needed for communication, while Azure DevOps is used for project management. The team also has a dedicated channel in Sikri's Teams for storing shared documents in the cloud.

High process quality often leads to high product quality, as a well-defined and executed process can ensure that requirements are clearly understood, risks are identified and managed, and defects are detected and resolved early (Beck, et al., 2001). By prioritizing process quality, it is possible to ensure that the resulting product meets the specific needs and requirements of the end-users and stakeholders, and that it is reliable, maintainable, and secure.

2.4. Product Quality

Product quality is arguably the most important aspect of software development, as it directly affects the user experience and thus the success of the project. To ensure high quality in the product, the use of established models is often a reliable approach to improve the outcome (Sommerville, 2016, pp. 657-663). Therefore, ISO 25010's product quality model was initially chosen. This model defines eight characteristics of product quality: functionality, reliability, usability, efficiency, maintainability, portability, compatibility, and security (ISO, u.d.). On further inspection we found this to be very similar to the 12-design criteria model proposed in Object Oriented Analysis & Design by Lars Mathiassen (Mathiassen, Munk-Madsen, Nielsen, & Stage, 2018, ss. 179-184). As the team were familiar with this model from previous projects, it was decided to utilize this instead.

It is important to note that while it is ideal to fulfill all twelve characteristics, it is usually not possible to achieve this. Therefore, one should prioritize the characteristics that are most important based on the needs and expectations of the end-users and stakeholders. Figure 2 illustrates the **12 criteria** while the prioritization and reasoning for it can be found in chapter 5.1 Architecture design criteria.

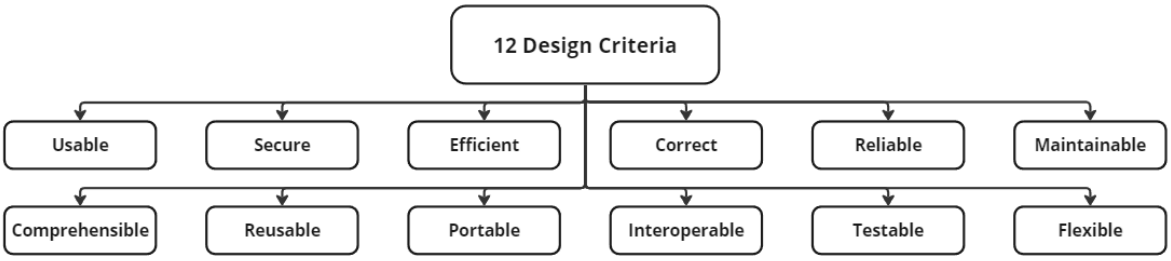


Figure 2: 12 Design criteria model

In the design and development of the notification system, specific requirements and preferences of the target audience will be considered, as well as the resources and constraints of the project. This will help determine which product quality characteristics to prioritize in delivering a product that meets the highest possible standards.

In summary, the team strives to achieve high product quality using the 12-design criteria model, prioritizing those that are most important for the end-users and stakeholders. This approach will help in the delivery of a notification system that is not only reliable, secure, and efficient, but also adaptable to the evolving needs of the users.

3. Analysis

System analysis plays a vital role in software development as it lays the foundation for the rest of the development process. It involves defining software requirements and specifications by gathering and assessing data related to the system and the domain it resides in. System analysis is the first and arguably the most important step in software development, as any inaccuracies can have significant implications for the project's success (Roper, 2021). As Sikri already had specified a lot of the systems requirements and specifications, the team's responsibility for analysis in this project was reduced.

3.1. Data Collection

Gaining understanding of the needs of the system and the domain in which it would operate was essential to develop requirements and specifications of the system. Considering the obtained insights from meetings and discussions with Sikri representatives, there was still a necessity for greater understanding of the current situation and the end user needs. The following subchapters delve into the two data collection processes conducted in this project.

3.1.1. Domain Knowledge Gathering

Success in software development is not only dependent on an understanding of technology but also on how the real-world functions. This understanding of a particular industry or field is often referred to as domain knowledge (Carter, 2023). A common way of gathering this knowledge is through talks with experts within the field, and this investment can lead to more qualitative solutions and better communication with stakeholders (Carter, 2023). Elements exist in a domain that the team were unfamiliar with from both a technical and non-technical standpoint. Therefore, the need to gather data was apparent, making discussions with experts in the field an optimal process. Since the notification system was to implement both front-end and back-end functionality, we scheduled meetings with employees representing these areas.

Starting off by conducting a meeting with the UX developed familiarity with the current Elements UI, its core functionality, and use cases relevant to the project. The outcome from this interaction included a fundamental understanding of Sikri's expectations and the end user needs. Subsequent of this was a technical meeting with tech leads and developers representing the whole tech stack. Discussions resulted in a greater understanding of what the system would be responsible for, from several perspectives.

The domain knowledge gathering phase did not reside within a fixed period, as this was performed in an agile fashion. As time went on, more insights and knowledge of end user

needs were obtained. This directly affected the iterative activities and changes of system requirements.

3.1.2. User Survey

In the purpose of gathering knowledge of the customers perspective, there were agreement with Sikri that a user survey was adequate. A user survey is a tool to obtain in-depth insights from customers (Userpilot, 2023). Utilizing this could uncover new insights and confirm the internal perceptions.

The team independently created a drafted document of questions, which then was handed over to the UX team for feedback. Conversations went back and forth until a final proposal was formed. As of that point in time, the team's responsibilities for the survey ended, as there are organizational regulations prohibiting direct customer interaction. Following this, the UX team and marketing department at Sikri further worked on finalizing the survey, which can be found in Appendix 6. Unfortunately, this process became more time consuming than anticipated and it was not distributed before the 8th of March.

The survey was made available online for the customers, with more and more responses arriving by the end of the project. As most of the results came in towards the end of the project the role it played would have been minimal. Therefore, no time was invested in analyzing the findings, but these will be useful for further development. As of May 11th, there were a total of 35 respondents, which are presented in Appendix 7 with the exclusion of the optional textual feedback.

3.2. System Requirements

The objective of this section was to conduct assessment of the previously collected data to determine requirements of the notification system. These substantiated the development process and were fundamental for delivering a product of quality. Resulting from this was a prioritized user story document, and a thoroughly processed system definition.

3.2.1. System Definition

A system definition is "a concise description of a computerized system expressed in natural language" (Mathiassen et.al., 2018, s. 24). An essential part of discovering the requirements of the system was to obtain an overall abstract description. In addition to providing common understanding and lay ground for the team's further analytical initiatives, the definition will also provide important and understandable information to stakeholders. The goal was to keep it simple and understandable, enabling all relevant parties to comprehend it despite their technical knowledge.

The process of developing the definition can include various activities. One useful method is called FACTOR criterion. Within this resides six elements: functionality, application domain, conditions, technology, objects, and responsibility. Exploring what satisfies each criterion in the chosen system, can then be considered when creating the system definition (Mathiassen,

Munk-Madsen, Nielsen, & Stage, 2018, s. 40). Table 1 was the results from determining the FACTOR criterion of our system.

FACTOR CRITERION

Functionality	*Notify user about relevant events*
Application domain	*Administration of notifications, subscriptions, and settings*
Conditions	*Occasional users. Office workers in the public sector*
Technology	*Micro service developed with Visual Studio and Docker* *Micro frontend developed with Visual Studio Code*
Objects	*End-users, notifications, subscriptions, and settings*
Responsibility	*Notification service*

Table 1: FACTOR criterion

The contents of each criterion were utilized when creating the system definition. To ensure the quality of the definition, the team thoroughly examined that the discoveries from FACTOR were present. Both the system definition and FACTOR were iterated back and forth, to ensure that both were as precise as possible. Following is the final system definition:

"A notifications microservice for Sikri's case management system Elements. The system will be developed in Visual Studio using C#, .NET, and SignalR for the backend, and React and TypeScript for the micro-frontend. The system will integrate with Sikri's existing application environment and provide several functionalities that will improve communication between the services and users.

The goal of this system is to improve the functionality of Elements and enhance the user experience for regular and occasional users in the public sector. With this system in place, users will be able to stay informed about the cases they are subscribed to, and administrators will be able to ensure that critical information is shared with the right people at the right time."

The system definition was repeatedly revisited throughout the project to ensure correctness and functioned as a base line for developing the user stories presented in the following section.

3.2.2. User Stories

The user stories were created in consideration of preliminary analysis efforts, to facilitate development processes. The entire team had great familiarity with the concept beforehand, unanimously agreeing on its positive effect on development projects. User stories are general explanations of features, written in a natural, non-technical language. They should emphasize

the end user's perspective, making developers oriented on the purpose of the implementation (Rehkopf, User stories with examples and a template, u.d.).

To benefit a concise and repetitive format, a decision was made to follow the format: as a (who), I want to (what), so that (why). This is recognized as the "Connextra format" (Agile Alliance, u.d.) and justified by the team's positive experience, in addition to its wide acknowledgement. Acceptance criteria were supplied in correspondence with the individual user stories as they were brought into the Product Backlog. These criteria are "the conditions that a software product must meet to be accepted by a user, customer, or other system" (Altexsoft, 2021). The focus of these was decided to aim at the end users of the system and was reasoned to increase understanding of the implementations both for the team and the stakeholders. The team chose to format the acceptance criteria with "Given - When - Then", to provide an end user perspective of obtaining a fulfilled implementation. All the user stories were prioritized using the familiar MoSCoW prioritization. This method includes four categories of initiatives: must-have, should-have, could-have, and won't have. Beneficial of conduction is sustaining the development work in prioritizing tasks (ProductPlan, u.d.). The user stories were further arranged in importance in sequential, numeric order.

The user stories were updated multiple times to stay tuned with the increasing knowledge of the team. Stakeholders, primarily represented through developers and the UX team at Sikri, were included in prioritizations with the purpose of keeping goals and objectives clear. The final list (depicted in Appendix 5) contained a total of 40 user stories. These were initially composed of an overall priority number, a MoSCoW priority, and the user story itself. The acceptance criteria and additional descriptions were documented in feature items created in Azure DevOps. A total of 19 user stories, comprising of almost all the defined must-haves, were introduced to the Product Backlog. Figure 3 shows an example of a user story.

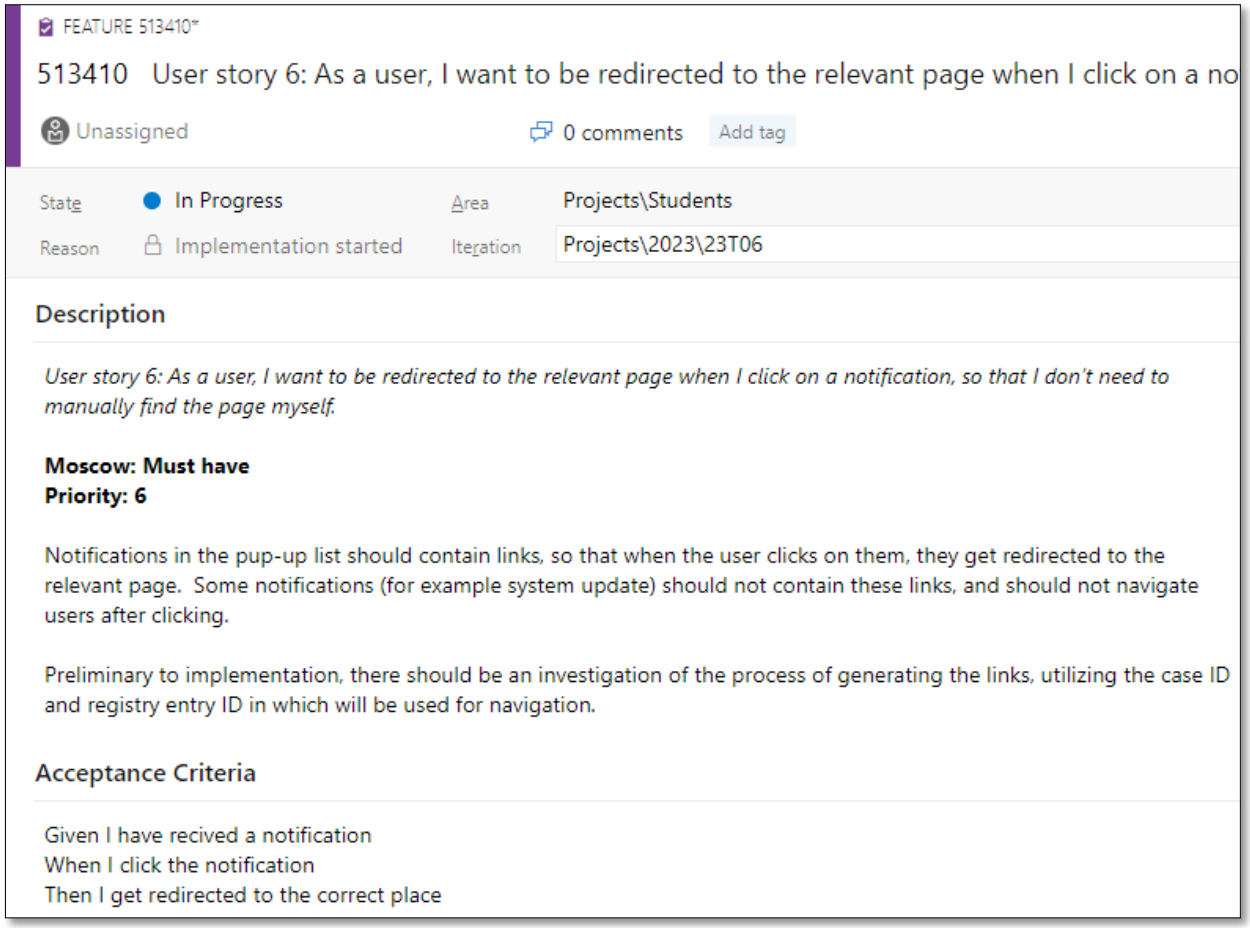


Figure 3: Depicts the 6th User Story

4. Technology and tools

This chapter will explain the tools, programming languages, and frameworks that were used to create the system. The reoccurring reasoning for choosing technologies for this project is the use of the same tech stack as Sikri. The main reason being that the product will be further developed and used by Sikri, thus facilitating the best possible handover. Unless the team is in a situation where the advantage of changing technology exceeds the disadvantages, this will take precedence.

4.1. Azure DevOps

The Product Owner required the use of Azure DevOps as a project management tool, as this is where all of Sikri developers operate. All repositories, code, and management work that Sikri does are contained here, making it easier for the team to access internal references in one place. Azure DevOps increases project control by providing an overview of remaining work, progress, and both assigned and unassigned tasks (Microsoft, 2022).

4.1.1. Azure Boards

Azure Boards is a tool that helps to keep track of the product and Sprint Backlog by allowing team members to organize tasks in a Kanban board based on their progress (Azure Boards, u.d.). The Azure board for managing the project can be viewed in Figure 4. For better organization, tasks are categorized into three swimlanes: front-end, back-end, and meta. By using user stories to create PBIs, team members can write acceptance criteria, descriptions and attach relevant documentation on respective tasks.

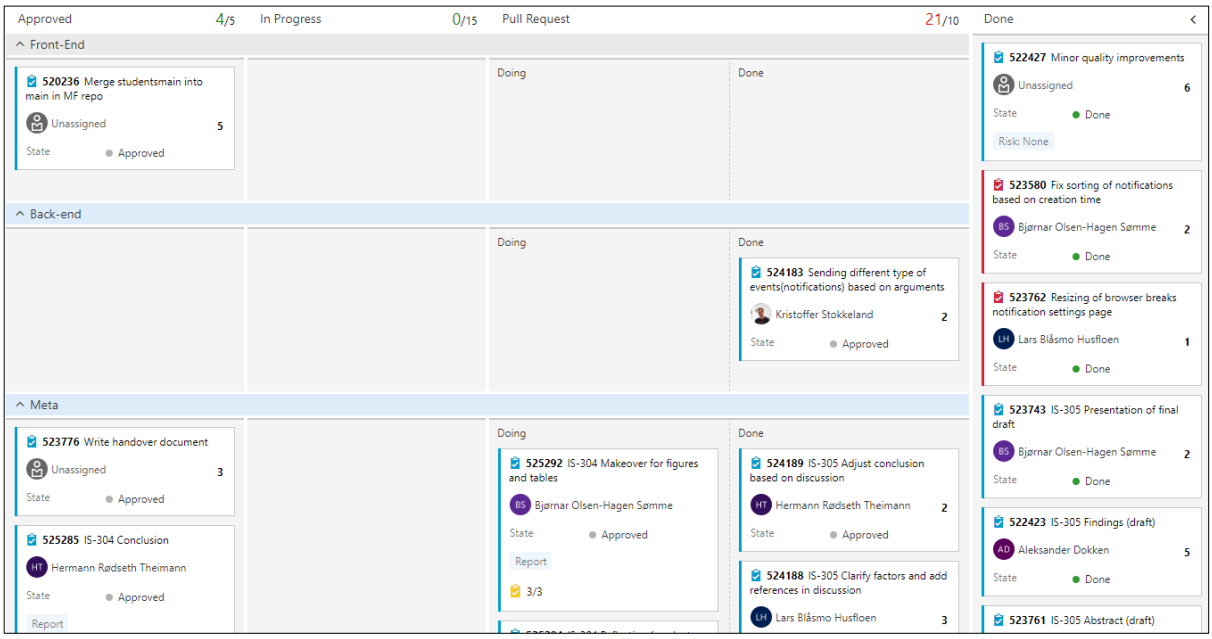


Figure 4: The Azure board for the project

4.1.2. Retrospective and Team Assessment

Two extensions of Azure DevOps used throughout this project were the Sprint Retrospective (Figure 5) and Team Assessment. The retrospective allowed the team to assess and document the successes and downfalls of each sprint, facilitating changes and improvements in future sprints. Meanwhile, the team assessment feature provides a platform for the team to reflect on their performance during a sprint or project, helping them to analyze and assess their performance and identify areas where they need to improve. Team assessment allows teams to evaluate themselves against specific metrics or criteria, such as team velocity, sprint burndown, code quality, and team morale. Overall, the purpose of team assessment is to help teams identify areas for improvement, increase team performance, build stronger teams, and improve the quality of work produced, leading to greater efficiency and effectiveness (CMOE, u.d.) (Indeed Editorial Team, 2023).

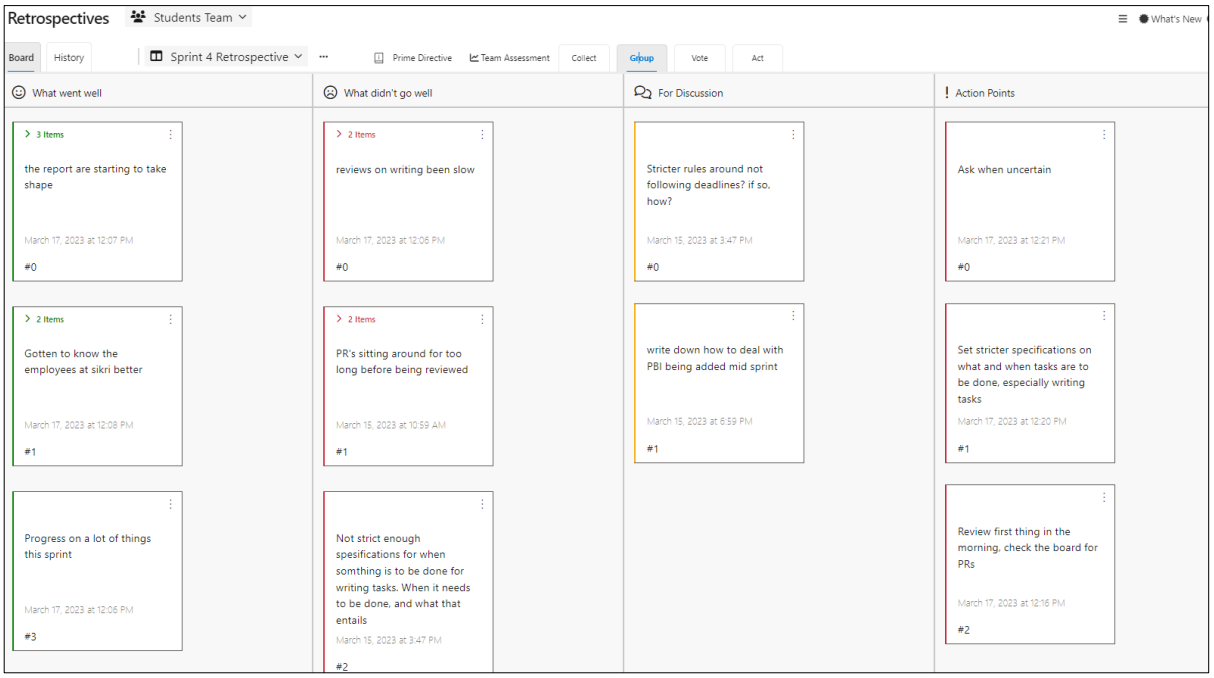


Figure 5: Azure Retrospective

4.1.3. Git Version Control

In addition to facilitating certain Scrum artifacts, Azure DevOps offers a Git version control system. Version control is the practice of managing and tracking changes to software code over time, aiding development teams in working faster and smarter (Atlassian, u.d.). Using the version control offered by Azure, code was linked to their respective tasks, increasing the team's control over completed features. There were two main repositories relevant to the project: one for the front-end and another for the backend. The team had a repository solely for the project's back-end, using feature branches that merge directly into the main branch using pull requests. The front-end repository was shared with the rest of the developers at Sikri who are working on establishing a new user interface for Elements. This makes version control even more crucial. Similar to the backend, feature branches were used for the frontend, but instead of merging into the main branch, the team had a separate branch

functioning as the main of this project. The use of version control is an important measure taken to maintain quality-assured code throughout the project (Atlassian, u.d.).

4.1.4. Azure Pipelines

Azure Pipelines was an important tool in the development process. It is a cloud-based service provided by Azure DevOps and offers continuous integration and continuous delivery (CI/CD) capabilities that can automate the build, testing, and deployment processes of an application (Microsoft, 2023). During development there was used two different CI pipelines for different stages to ensure higher code-quality throughout the project:

1. The first pipeline is triggered on every commit/push to a branch in Azure DevOps. Once activated, it runs several tasks. Firstly, it builds the solution and validates that the updated codebase is still buildable. Secondly, the CI pipeline runs all unit tests to confirm that the existing functionality works and that new features display the intended behavior. The last step is to publish a report detailing the code coverage of the branch. This code coverage report serves as a measure of how much of the codebase the unit tests cover.

2. The second pipeline is activated on every pull request into the main branch. Its responsibility is to perform static code analysis using SonarCloud. Static code analysis is a method of examining the source code before a program is run. By integrating SonarCloud, the team can detect potential bugs, vulnerabilities, and code smells in the pull requests before they are merged into the main branch as shown in figure 6. This automated process helps us to maintain high code quality by helping us reduce technical debt and mitigate potential security issues.

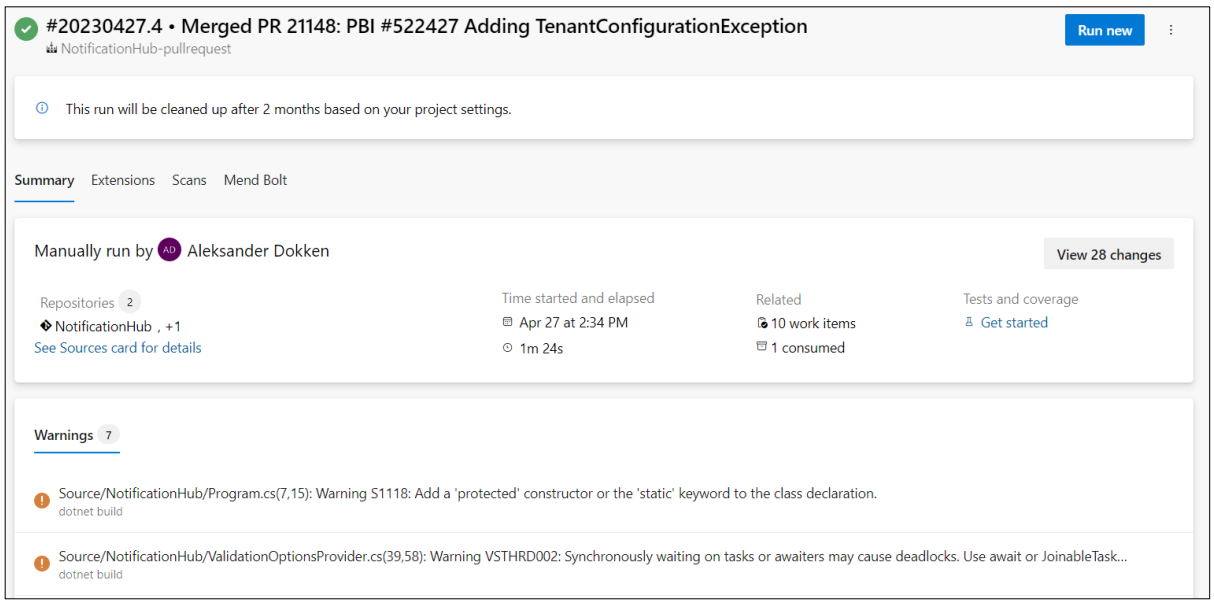


Figure 6: A complete run from the pipeline performing the Sonar scan

4.2. Frontend

In this chapter, the tools and technologies used in developing the micro frontend are introduced, namely TypeScript, React, Material UI, and SignalR. Each technology or tool is briefly described, with an explanation of why it was used and how it influenced the project.

4.2.1. Language and Framework

TypeScript is a programming language that is based on JavaScript and has strict syntax rules (Agrawal, 2022). It provides several benefits, such as improved readability, cleaner code, and more intuitive programming by precisely defining the data type that a variable can hold. Although the team had no prior experience with TypeScript, its similarity and the teams prior experience with JavaScript made it manageable for the team to learn and use effectively. To put it simple, Typescript is a version of JavaScript with extra features (Simplilearn, 2023).

Similarly, **React** is a popular JavaScript library that is widely used for building user interfaces (Deshpande, 2023). Although based on JavaScript, it is also possible to use with TypeScript (Simplilearn, 2023), which was done in this project. It offers reusable UI components, manages component state, and breaks down the interface for more efficient development. In addition, React ensures that the user interface remains fast and responsive even when handling large amounts of data or complex interactions. The team had limited experience with React, only having taken some online courses, as well as prior knowledge of JavaScript, the foundation of the library. As a result, the adaptation of React was manageable.

Material UI is a React library with pre-built, customizable UI components based on Google's Material Design guidelines (MUI, u.d.). It simplifies development, reduces time, and increases productivity, while ensuring a consistent and professional look. Its modular components enable creating complex interfaces easily (Sirotka, 2022). Although the team had no prior experience with this library, it was found to be easy to use, thanks to its excellent documentation. Material UI was particularly helpful in creating components that matched Sikri’s existing front-end theme, as well as removing the need for building common components from scratch.

SignalR is a real-time communication library that simplifies the implementation of real-time functionality in web applications (Fletcher, 2020). It enables communication between clients and servers through an API for sending and receiving messages. This makes it ideal for adding real-time functionality and notifications to web applications, resulting in more responsive and engaging user experiences. Additionally, SignalR is highly scalable, making it suitable for handling large volumes of traffic. Using SignalR allowed updating users' notifications in real-time, resulting in increased usability and responsiveness (Microsoft, 2020). Although none of the team members had prior experience with SignalR, existing JavaScript knowledge simplified its implementation on the client-side.

When building a notification-related system, these technologies can be combined to create a powerful front-end stack. React and Material UI can be used to create a visually appealing and dynamic UI for managing notifications, while TypeScript ensures code robustness and maintainability. SignalR can be used to push new notifications to the front-end in real-time,

making it responsive and improving the overall user experience. In summary, combining React, TypeScript, Material UI, and SignalR enhances the quality and user experience of a web application, making it more dynamic, responsive, and efficient.

4.2.2. Testing Frontend

In chapter 3.2, the system requirements were outlined in the form of user stories and a system definition. The requirements define what is expected of the product and its functionality, but how can one determine if they have been met?

"Testing is the process of evaluating a system or its component(s) with the intent to find whether it satisfies the specified requirements or not." (Toturialspoint, u.d.). Furthermore, testing can reduce costs, improve a systems reliability and overall quality, increasing customer satisfaction (IBM, 2016). There are diverse types of software tests, that each have their own purpose.

As mentioned, decisions were made so that the system would adhere to Sikri's standards. During development there were little focus on testing on the front-end side, where only manual testing was done. The team recognized the importance of testing and wanted to stray away from Sikri's approach by adding unit tests. An attempt at this was made, but as there were no established testing standards or existing tests to draw inspiration from, this turned out to be much more time-consuming than expected. As a result, the team ended up going for the same approach as Sikri utilizing manual testing to ensure fulfillment of the established requirements.

4.3. Backend

In the following subchapters, the technologies employed in development of the backend are discussed, with the programming language and development framework, database, and testing frameworks being highlighted.

4.3.1. Language and Framework

C# is an object-oriented programming language developed by Microsoft and commonly used to develop applications and services within the .NET framework (Microsoft, 2023). Being a strongly typed language, it enables developers to write cleaner, more maintainable, and intuitive code, with precisely defined variable data types (Microsoft, 2022). C# also supports type inference, which enhances code readability and maintainability. Type transparency ensures the correct data is passed between components and functions, resulting in fewer bugs and faster error detection.

C# shares similarities with Java, which is the programming language that the team had the most experience and knowledge of. Additionally, a couple of the team members had used C# in an earlier project, making the transition easier.

.NET is a developer platform made by Microsoft that provides a range of tools and resources for building scalable, secure, and high-performing applications (.NET architecture, 2023). It includes the runtime environment, libraries, and tools needed for building, deploying, and managing applications and services. **ASP.NET** is a web application framework within the .NET ecosystem that allows developers to build web applications, web services, and APIs using .NET languages like C# (Microsoft, u.d.). With its modular architecture, ASP.NET provides flexibility and robust security features for modern web applications. The combination of flexibility, security, and performance of both .NET and ASP.NET provides a powerful framework for building backend systems.

While only one team member had prior experience with ASP.NET, a few had experience with .NET, making it manageable for the team to learn and use effectively. Together, these technologies form a cohesive ecosystem for creating web-based solutions on various platforms.

MassTransit-transport is a messaging framework for .NET applications that simplifies building distributed systems (MassTransit, u.d.). It offers a range of messaging patterns and features, such as pub/sub and message routing, enabling developers to create scalable and maintainable messaging applications. MassTransit-transport promotes separation of concerns and easy integration with other technologies. The framework supports multiple transport protocols and messaging patterns, providing built-in features like message serialization and error handling. For the project, MassTransit was primarily used for Azure Service Bus and RabbitMQ, which are both messaging services (Yousuf, 2022). RabbitMQ was used for local testing of the system, while Azure Service Bus is what Elements are using, and what the final product will have to connect to.

4.3.2. Database

Microsoft SQL Server (MSSQL) is a relational database management system for building applications (TutorialPoint, u.d.). It integrates well with other Microsoft technologies, such as C#, .NET and ASP.NET, providing developers with a comprehensive set of tools for building high-performance data-driven applications (Microsoft, 2023). By providing a secure and reliable data store, SQL Server ensures that applications are scalable, maintainable, and performant.

Entity Framework Core (EF) is a popular object-relational mapper that allows developers to use C# objects for interacting with the database and removes the need for writing much of the code required manually (Entity Framework Core, 2021).

The team had limited experience with both technologies but regarded the risks as limited. MSSQL is the database system primarily in use by Sikri allowing for easy help and potential integration. EF is a bit more familiar and allowed the team to concentrate on non-database functionality.

4.3.3. Testing Backend

xUnit is a widely used testing framework for .NET applications that enables developers to write robust and maintainable tests (Microsoft, 2023). It provides an easy way to define test methods, assertions, and fixtures, ensuring the correctness of the code (Sheth, 2021). This framework was chosen as it is used by Sikri, and some team-members had experience with it. xUnit was used to develop unit tests primarily of business logic. To generate test coverage reports (Figure 7) of the automated tests, the Grunt.js task-runner was used. This report was used to ensure sufficient testing of business logic.

	Line coverage					Branch coverage		
Name	Covered	Uncovered	Coverable	Total	Percentage	Covered	Total	Percentage
NotificationHub	**406**	**704**	**1110**	**2427**	**36.5%**	**63**	**212**	**29.7%**
SignalRServer.Contracts.NotificationCreated	1	2	3	11	33.3%	0	0	
NotificationHub.ValidationOptionsProvider	3	12	15	45	20%	0	2	0%
NotificationHub.Startup	52	9	61	121	85.2%	2	4	50%
NotificationHub.Services.TypeSettingService	8	3	11	34	72.7%	2	6	33.3%
NotificationHub.Services.SubscriptionService	9	4	13	36	69.2%	2	2	100%
NotificationHub.Services.NotificationService	16	12	28	61	57.1%	2	2	100%
NotificationHub.Program	0	8	8	26	0%	0	0	
NotificationHub.Models.User	10	0	10	19	100%	0	0	

Figure 7: Coverage report sample

Postman API is a popular platform for API development that simplifies designing, testing, and documenting APIs (Postman, u.d.). Developers can easily collaborate on development projects and create clear and concise API documentation. With Postman API, the API development process is accelerated, ensuring the quality and functionality of the final product. The team had prior experience using this API, and in this project, Postman was used to quickly test different API calls instead of running the entire application.

5. Architectural Design

The creation of a software architecture is fundamental in the development of any software system. It serves as the base for understanding and addressing the essential requirements for the system being developed (Martin, 2018). This chapter will present the architecture of the system where it introduces and prioritizes design criteria for the architecture, presents a high-level model of the architecture and dives a bit into the database design.

5.1. Architecture Design Criteria

There are numerous design concepts that can be applied to systems development. Following a set of well-established criteria when creating a system design allows the team to stand on the shoulders of other professionals in the field and communicate in a common language. As mentioned in chapter 2.4 Product Quality, a list of 12 criteria specifically for design of the system's architecture was selected, reasoned by the team’s previous experience.

When prioritizing the different criteria, the team chose to base it on the amount of time that will be spent on each criterion and not its overall importance to the whole project. This choice was made to keep the criteria as useful as possible for the team and avoid spending time on tasks Sikri already had solved or would need to solve. The prioritization was done

based on the importance expressed from Sikri, their preferred focus for the project, and previous analysis. Listed in **Feil! Fant ikke referansekilden.**2 are the prioritized criteria.

Criteria	Very important	Important	Less important	Irrelevant	Easily fulfilled
Usable			x		
Secure					x
Efficient	x				
Correct		x			
Reliable		x			
Maintainable	x				
Testable	x				
Flexible	x				
Comprehensible		x			
Reusable		x			
Portable			x		
Interoperable	x				

Table 2: Prioritized design criteria.

The following subchapters dive deeper into the different criteria giving reasoning behind the prioritization, with examples of how they have been worked into both design and practice. As the importance of the criteria decreases so does the level of detail, putting emphasis on the once that played a vital role in this project.

5.1.1. Very Important

Efficient: In a high traffic microservice architecture, which Elements is built upon, it is important to make sure the service provided is efficient in order to ensure the overall system performs well and can handle the scale of the user base. A high level of efficiency can also help reduce operational costs for the company, as well as improving the user experience.

In practice, this has primarily meant being careful of the number of requests and data flow from this part of the system to other services and within itself. The background service was designed to get most information pushed to it from other services and store what is relevant for its functoriality. Database tables expected to be used in logic are lightweight and easy to access by indexed keys. The micro frontend balances correctness of information presented to users and request-rate.

Maintainable, Testable, Flexible: All these criteria are very important based on the requirements from Sikri, clarifying that the code should be of such a quality that they easily could take over the development of the system when the project is done. They have stressed that writing a small amount of high-quality code is preferable over creating a cluttered codebase with several unfinished or low-quality features.

The service utilizes dependency injection and familiar patterns such as repository-service to strengthen all of these. Further, classes implementing business logic are largely covered by unit tests. A primary goal when it comes to flexibility has been to facilitate ease of adding new notification types and expand functionality around them.

Interoperable: Interoperability in the notification service is important because it allows the service to easily integrate with other components of the microservice architecture. Our system interacts with other services for configuration, logging, and authentication, communicates with other services via event queues, and will likely need to look-up some data in shared data storage. The repository pattern was implemented partly because the database might be exchanged with a service owning the database in the future.

5.1.2. Important

Correct: When it comes to the fulfillment of requirements or correctness, Sikri has given specific requirements to how the system should be and that these were important to follow, especially on the backend. On the front-end, Sikri gave the team more room to try new designs as they were responsible for providing high-fidelity mock-ups as they are developed for the overall design.

Reliable: A reliable system ensures that notifications are delivered in an accurate manner, and that the service is available and functioning as expected. For example, if a user relies on notifications to receive important updates or messages, a reliable service can help to ensure that they never miss an important notification. The reason this criterion is not a “very important” one is because Sikri already has most of the infrastructure in place to make sure that the correct users would get the right notification and therefore not expose any sensitive information.

Comprehensible: It's important for a notification service to be easy to comprehend for future developers because this can help to ensure it becomes a long-term success as well as making the maintenance of the system easier. If a notification system is difficult to understand or modify, it can be a major obstacle to future development and improvement efforts.

Reusable: Adhering to Sikri’s requirements, the notification systems micro frontend should be possible to reuse in different applications. Despite this, it was clear that the focus should be held towards first implementation in the new Elements UI, making this of lower consideration in the project scope. As this is a micro frontend it should be easy to reuse.

5.1.3. Less Important

Usable: Considering the project as a whole, usability could be in the “very important” category. When it comes to the time that will be spent towards this it is not as important, mainly since the UX team have already proposed a solution based on data and their experience. The team would therefore only focus on implementing their suggestions.

Portable: The system will primarily be run in virtualized Linux containers in Azure, if applicable the other technical platforms will also rely on virtualized containers.

5.1.4. Easily Fulfilled

Secure: Since the system that is being developed will exist inside Sikri's Elements, the security will not be a major concern since it will use authentication and authorization provided by them. The focus in development has been to avoid demonstrably unsecure practices.

5.2. Architectural Model

To be able to comprehend the notification services placement and intended data flow in the greater system of Elements, flowcharts and models like the one presented in Figure 8 have been frequently utilized. These have been developed together with tech leads and been used to have a frame of reference in discussions. Within these discussions around technical solutions, the design criteria described in the previous section were carefully considered to strike the best balance. For instance, the architectural model highlights the interoperability of the system, with the NotificationHub service integrated with other services.

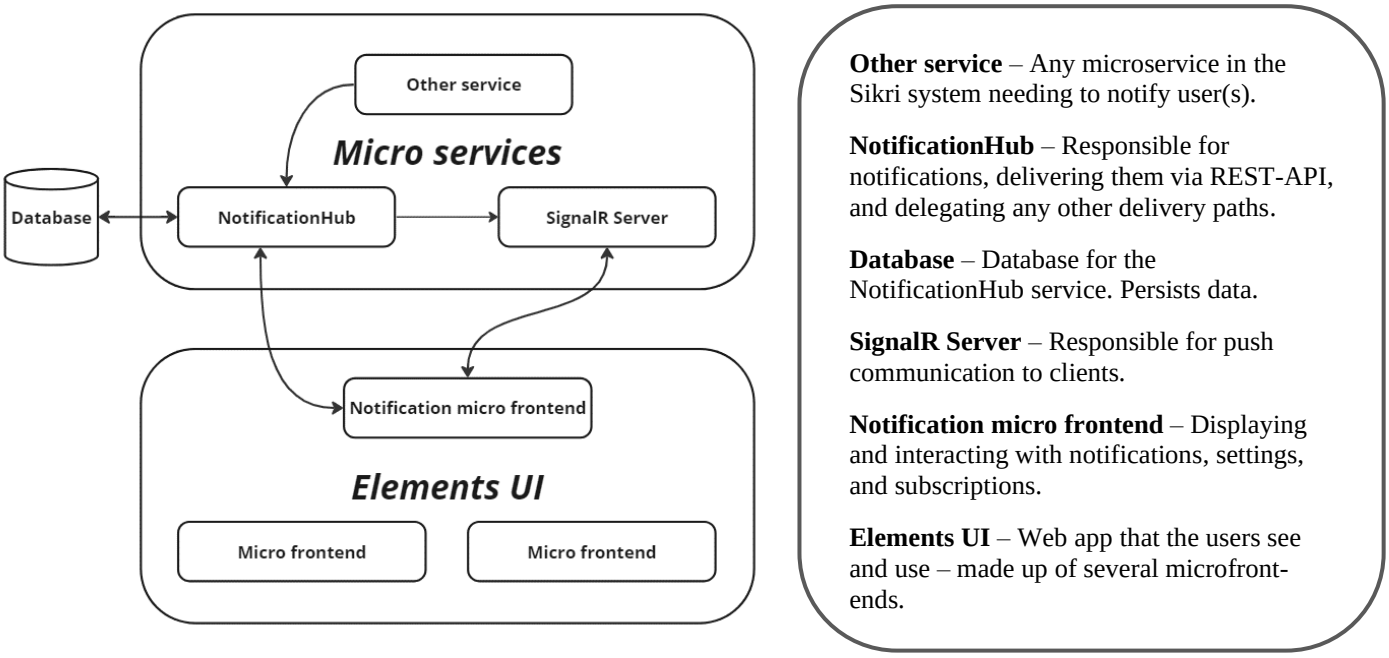


Figure 8: Architectural model

5.2.1. Example Flow of a Notification

A **Service** sends an occurred event through the service bus. **NotificationHub** picks up the event and checks the contents of the event against the users' settings and subscriptions in the **Database**. A notification is persisted for each applicable user, and a message is sent via service bus to **SignalR Server**. The **SignalR Server** then notifies all connected clients of said users. The client then sends a request to the API within **NotificationHub**. When the user

marks a notification as read, the client makes that call to the **NotificationHub** API to store this change in the **Database**, while the client informs other clients via **SignalR Server**.

For the efficiency design criteria **SignalR Server** allows us to get information about new notifications pushed from the server side to the clients instead of sending frequent requests from the clients to check if there are new notifications.

5.3. Database Design

Sikri's current database NCore can be described as a monolith, currently moving towards a modular monolith design. Integrating the notification systems database into the current iteration of NCore would add unnecessary complexity. Therefore, it was decided to create a separate database related to the NotificationHub micro service with clear contracts. From early on, it was clear that this database would likely be replaced. As a result of this, the database development was not a major focus in the project. The final design of the database, after being iteratively updated, can be viewed in an ER diagram presented in Figure 9. Note that Users is a table that only contains what is currently relevant for this project about the users, authentication, and more details are stored elsewhere. Further, the Notifications table would likely be iterated on more when introducing localization of notification content.

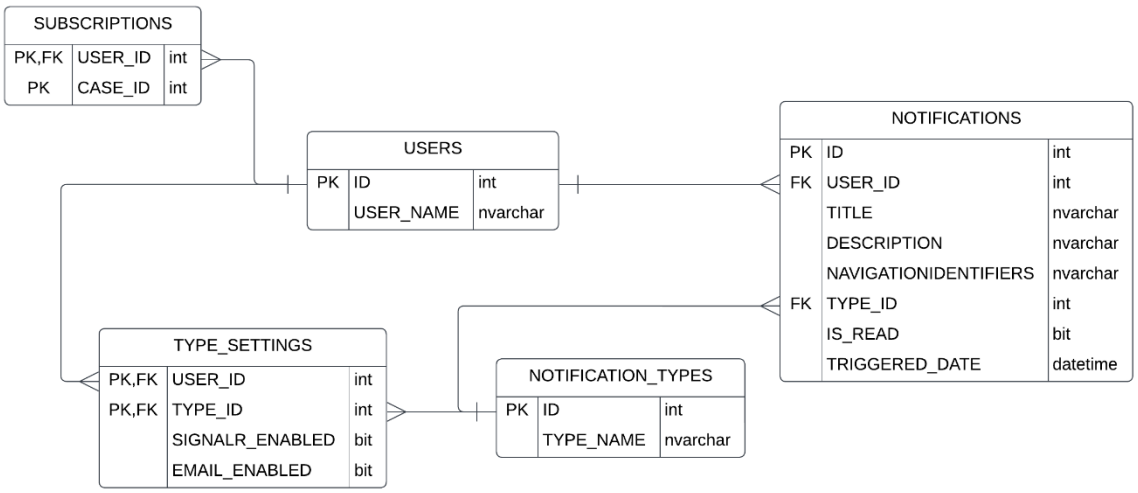


Figure 9: ER Diagram

6. Project Execution

This chapter will outline the project's progression throughout the semester. It will start with the opening phase, before outlining the development in the seven sprints and at the end wrap up talking about the closing stage. This should give a simple, clear view of the project from start to finish. It is worth noting that while there is some reasoning for important choices in this section, the most important decisions are covered in depth in chapter 8 Reflection.

6.1. Project Start

The project started in earnest 12th January with a start-up meeting with several Sikri employees. Prior to this, only some cursory information about the project had been shared at the end of the previous academic term. At this meeting the project was introduced and discussed in greater detail together with some of the employees the team would collaborate with. Afterwards, the team discussed how it should organize itself, landing daily stand-up times, primary working days (Monday, Thursday & Friday), Scrum Master, and much previously discussed under the Methodology chapter. A second meeting with Sikri featured a small walkthrough of the Elements program, an exchange of domain knowledge, and discussion about questions for a user survey.

Over the following week, the team made a system definition, project description, and several user stories. The user stories were discussed with Sikri before being prioritized. The team also made a project plan, performed a risk analysis (Appendix 4) for the project, and signed a group contract (Appendix 3). Furthermore, the team started investigating and drafting the software architecture and an initial database schema.

On the 18th of January, a meeting was held with the project owner, discussing the scope of the project with the team presenting their current progress, and the way forward was agreed on. From this point forward, a representative of the project owner was appointed, and a meeting with developers was planned to discuss the architecture. While awaiting clarifications and access from Sikri, the team decided to start getting familiar with C# and technology that the project would require. The team worked individually on SignalR projects from the 19th till the 23rd. The first sprint was supposed to start on the 23rd but as this was postponed to the 26th, the team instead spent the day having demos of the individual projects to support communal learning.

6.2. Development

The following subchapters will dive into the project's sprints, except for the pre-sprint which was covered in the "Project start" chapter above. The first sprint will cover a lot of the artifacts used each sprint in much more detail compared to the rest. The reason for this is to avoid repetition by only discussing changes or additions of artifacts in the following sprints where these occur.

6.2.1. Sprint 1 (26.01 – 03.02)

Despite the team's initial plan to begin their first sprint on the 23rd of January, it was decided to delay the Sprint Planning until the 26th due to what felt like a lack of information on some of the project's aspects. The reason being two scheduled meetings with Sikri employees on the 26th addressing this. The first meeting was regarding the architecture of the microservice, where the team together with a tech lead and a couple developers from Sikri created a draft. Having a solid architecture is important but meeting the customer's demands and creating a good user experience starts and ends with the users. Therefore, the second meeting was with

the UX-team where the front-end's behavior, contents, and overall scope was discussed. As mentioned earlier in the report, Sikri would be responsible for the design, giving the team more time to focus on the development of the system. During both meetings, thoughts and ideas got traded and discussed, removing uncertainties as well as creating a more uniformed view of the project.

With this improved context and domain knowledge, the team held the first official Sprint Planning of the project. The Sprint Planning started with selecting user stories and if necessary, breaking them into smaller tasks to create Product Backlog items (PBIs). The user stories for the sprint were chosen based on the existing MoSCoW prioritization, as well as other factors that impacted the development and completion of the stories/PBIs in the current sprint. In this sprint, it was chosen user stories that from a coding standpoint created a solid foundation and a good starting point for further development. After debating if the right amount of PBIs were created and added to the sprint backlog, tasks were self-assigned and new tasks were picked from the board when a team member finished a PBI. Daily standups were held on the days the team worked on the bachelor's project, which normally was Mondays, Thursdays, and Fridays.

The second week of the sprint started with a meeting with the project owner representative and two other front-end developers. Here the team was mainly introduced to the mono repository structure of Sikri's new Elements UI implementation, where the notification micro frontend would reside. The next day, permissions for the necessary repositories were sorted and the team started writing code for both the front- and backend parts of the system. It made sense for the micro frontend to "live" in the existing mono repository, while an entirely new repository was created for the backend service. To not interfere with Sikri developers' workflow, it was decided that a "students_main" branch would be created in the mono repository so that approval from their side was not needed on every pull request (PR).

As the notification micro frontend was being set up, the team encountered some initial challenges with the code. Despite it being configured and structured in the same way as the existing ones, modifications made to the code were not reflected in the user interface. All the micro frontends were hosted in the cloud and connected through a shell micro frontend. Running all of them locally was computationally heavy and time consuming, so only the specified or affected ones were spun up. However, due to the notification micro frontend's direct use in the shell micro frontend, that differed from the existing ones, the environment variables needed to be adjusted. The problem was solved by running the shell micro frontend locally without using their deployed cloud instance.

During the sprint, the team completed all the PBIs (Figure 10) added during the Sprint Planning and got started on an additional three new ones, that were added towards the end of the sprint. This meant the team now had a working micro frontend, that consisted of a bell icon with a popover where a list of notifications got displayed when clicking it. A database schema was also created together with a docker image running a Microsoft SQL Server database. The tables were then populated with simple test data. Further on the backend side, an implementation utilizing Microsoft entity framework for communicating with the database and a controller for retrieving all the notifications were developed. Since there was no connection between the front- and backend at that point in time, Postman was used for testing the backend endpoints. The three PBIs that were added at the end of the period did not get completed during this sprint, thus the functionality they included will be covered in the next chapter.

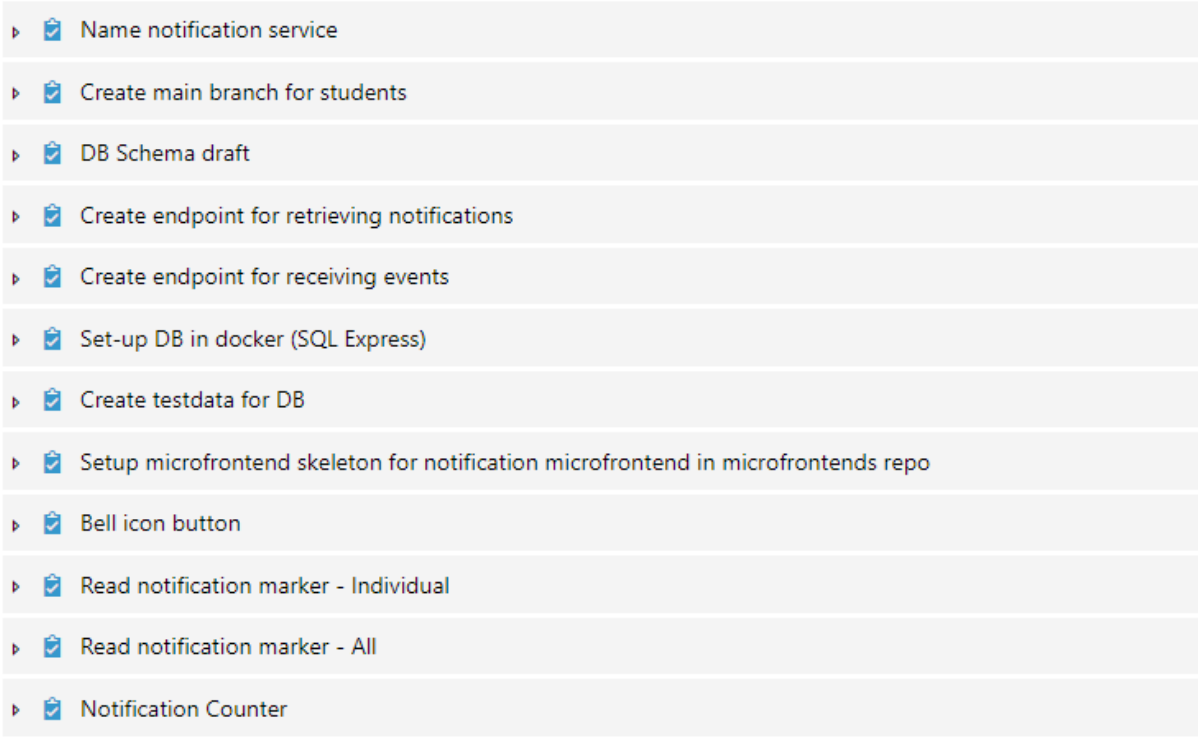


Figure 10: Backlog for sprint 1

On the last day of the sprint, both a review with a demo and a retrospective were held with the project owner representative present. By continuously consulting and getting feedback from Sikri, the team were able to ensure a high degree of correctness, which was deemed important as one of the design criteria in chapter 5.1 Architecture Design Criteria. It was decided to use the Azure DevOps built in retrospective functionality, evaluating the progress, and identifying areas for improvement. The retrospective identified several positives, including productive meetings with Sikri employees, good development progress and good communication both within and out of the team. However, there were also points to improve, including spending too much time on configuration, created tasks being too small, waiting for permissions, and a need for a better daily stand-up routine. Overall, the retrospective provided valuable insights for future sprints.

6.2.2. Sprint 2 (06.02 – 17.02)

To improve the Sprint Planning process from the previous sprint, estimation was introduced. This made it easier to ensure that the PBIs were not too large as well as selecting the right amount of PBIs to take on, based on the number of hours the team had at its disposal. Using the built in estimation tool offered in Azure DevOps, the PBIs were estimated using effort represented by Fibonacci numbers in the same way teams at Sikri does it. Because of uncertainty regarding the way forward, especially on the frontend side, the initial Sprint Planning was limited. Luckily this was resolved quickly, and a sprint replanning was held on the first Friday of the sprint, filling up the board with enough PBIs for the rest of the period. As a measurement to improve the daily standups from the previous sprint, "walk the board" was added to the ceremony.

During the previous sprint, a solid foundation for both the front- and backend was achieved, but as they stood, they could not communicate with each other. After connecting these two, additional functionality such as sorting between all, and unread notifications was implemented. Additionally, it was now possible to toggle between read and unread state on individual notifications. The Elements UI is built to support four different languages; English, Norwegian bokmål and nynorsk, and Swedish, so translations for these were also added. To receive notifications in real time, a SignalR client was developed with a connected SignalR hub in the backend repository. The events that the notification service will send notifications about occur in other services and are sent to a service bus. Then the notification service will consume these and turn them into notifications that are stored in the database. Since the interface for the events were not determined yet and it was difficult to create test events in Sikri's existing environment, a local queue was added for testing.

It became apparent that the user stories created at the beginning of the project were far from good enough, both in quality and quantity. Their scope was too big and ambiguous, making it time consuming and difficult to break down into smaller PBIs. As they also overlapped, it was near impossible to know when a story was completed. Therefore, great effort was put towards improving these, and the total amount went from 13 to 40 well defined ones. There had also been a lot of ambiguity surrounding PBIs and their descriptions, so the document containing the "definition of done" was updated to address this. Now, for a PBI to be added to a sprint it needed to include a description of what is to be done, an estimate, and acceptance criteria. In the cases where this was still not adequate, additional description of the task was added.

Since the steering committee meeting had to be held during this sprint, it was decided that it would be combined with the Sprint Review so that both the supervisor and the project owner representative could attend together. Afterwards, the retrospective was held, and the team assessment functionality in Azure DevOps was added to the retrospective process, making it possible to further assess the team's health, performance, and areas of improvement.

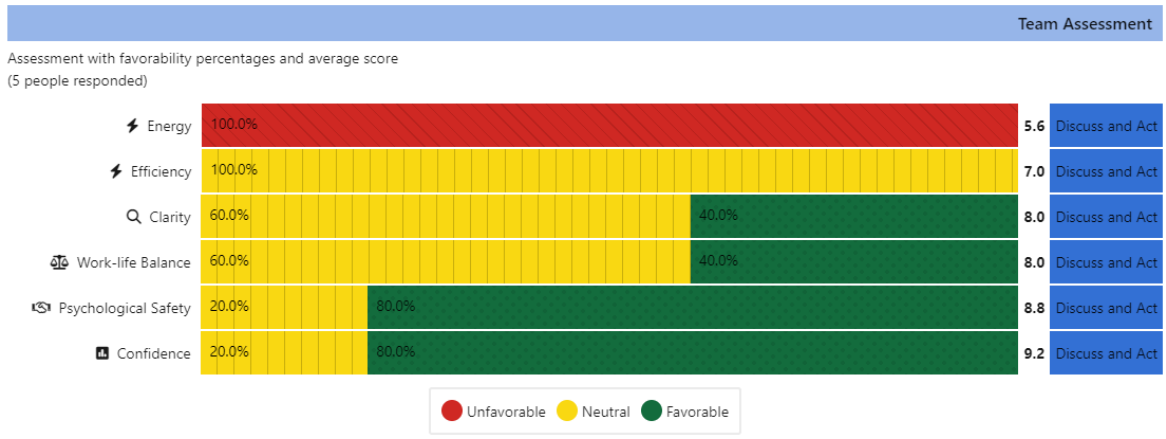


Figure 11: Team Assessment in sprint 1

Looking back on the sprint, the daily standups and retrospective both improved in quality, and there was more pair programming amongst the team members. Even though this Sprint Planning was overall better than the last, there were still created too few PBIs, and the estimating process using effort did not work as well as hoped. The PBIs were better defined, but not good enough, causing uncertainties when picking up tasks later in the sprint. Because of the user stories encompassing too much, a lot of ad hoc PBIs were created, straying a bit from using the backlog as intended. Furthermore, a lot of writing for IS-305 was also done, giving less time toward development. As a team that favors programming over writing, together with the things mentioned above, resulted in a low score on the energy for this sprint as shown above in Figure 11.

6.2.3. Sprint 3 (20.02 – 03.03)

Before starting the Sprint Planning there were still some user stories that needed to be prioritized before the process of adding new ones could begin. As the requirements for pulling a new PBI to a sprint had increased, the process took a lot longer than before. As for the estimation, it was discovered that all the team members had a different understanding of effort, which led to poor estimates. By changing the estimation metric to hours, the process went a lot smoother, and the estimates ended up being more accurate. Even though this new process was more time consuming when creating the PBIs, it made up for it by elevating their quality, removing uncertainties in the long run.

The sprint goal was to set up a functional RabbitMQ queue that is seamlessly integrated with SignalR and the backend service. Additionally, the team planned to introduce a filter mechanism to distinguish between read and unread messages, display notification creation time, and enable support for multiple users on SignalR. To achieve both changes related to

SignalR, it was decided to move to Sikri's service for SignalR, adding the required additional functionality for the notification system. Furthermore, the team intended to develop a test case to confirm the correlation between notifications and cases and to provide feedback in the event of a service malfunction.

Since Sikri would be responsible for most of the design, their frontend team developed a react component for the notifications that we could use. This new component now needed to be substituted for the "dummy" component created in the first sprint. Since their component lacked formatting of the date, this was added to cohere to the designers' sketches. To transfer the notifications safely between the back- and frontend, authorization using Sikri's existing bearer tokens was implemented. In case of something going wrong with the service, exception handling was set up to inform the users. This was done by adding a red X to the notification bell and giving the user feedback in a tooltip when hovering the bell button. This can be viewed in figure 12.

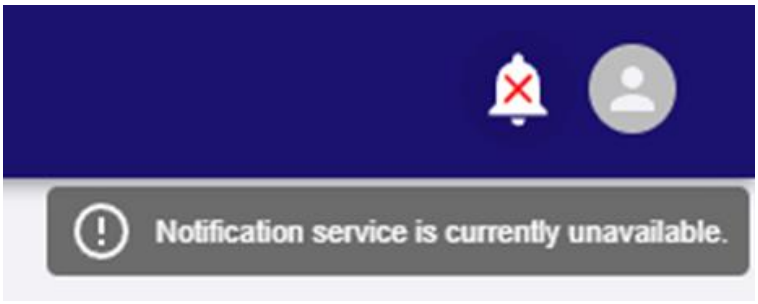


Figure 12: Error feedback when hovering over notification bell

Since it had been a long time since the team were in contact with the Product Owner, a meeting was scheduled where a demo showing what had been done so far was presented. Due to difficulties of setting up meetings with multiple employees high up the chain in Sikri at the same time, another demo was held for the operations team lead during the Sprint Review. In both instances, the team received great feedback on the demo.

The retrospect was done as in previous iteration and concluded that this was overall the best executed sprint so far. This was also reflected in the team assessment, where almost all the metrics increased, and the energy went from 5.6 all the way to 8.3. This was mostly thanks to the new and improved user stories and the refactored definition of done, facilitating improvements throughout the sprint. Furthermore, major progress was achieved on both the front- and backend, with a great deal of assistance from Sikri's employees. Previously, the team members had mostly worked on either front- or backend, but during this sprint everyone contributed on both ends. As for improvements, writing tasks should have been added during the Sprint Planning and not during the sprint as it was realized they were missing. The sprint goal proved overly ambitious, resulting in the transfer of a few partially completed and untouched PBIs into the next sprint.

6.2.4. Sprint 4 (06.03 – 17.03)

The focus in sprint four shifted from development to report writing. Even though documentation and some writing had been done continuously throughout the project, putting everything together and setting up a structure for the entire report was needed. The necessities to fill the holes between what was already present in the report draft became the basis for a lot of the PBIs this sprint. The focus was not on writing delivery ready quality but rather filling the report skeleton, while what was done was fresh in memory. At the end of the sprint, a draft for what would be around half the final report was completed.

On the development side, the background service in the form of a consumer was implemented. It contained a queue listener along with a worker utilizing multitenancy and MassTransit to make the sending and receiving of events through the queue possible. In addition, some bugfixes like removing race condition on database calls, and more extensive exception handling were done. Both a frontend implementation for the notification settings, and utilization of the queue lister with SignalR was started but not finished during this sprint.

The "Digitalkonferansen" took place at the theater in Kristiansand on the final Thursday of the sprint, and both the team and Sikri's employees were invited to attend. The conference was both informative and enjoyable, featuring a variety of interesting presentations followed by a meal and socializing. Although the "Digitalkonferansen" was a rewarding experience for the team and Sikri's employees, it resulted in the loss of one workday for the bachelor, leaving less time than usual for the sprint.

The Sprint Review was held with just the team members and there was no demo due to a lack of a considerable product increment from the sprint. After the retrospective, it was clear that even though a lot of writing was done, there were still improvements to make regarding writing task lifecycles. Starting with their creation just like programming PBIs, these needed better descriptions pertaining to their expected outcome. The deadlines for their completion were also not upheld. Remembering to move PBIs on the board to reflect their current situation was also not done, and it took too long before PRs got reviewed, hurting the continuous delivery process. Just like in the second sprint, the team assessment results and especially the “energy” was quite low, as the sprint this time as well was dominated by writing tasks.

6.2.5. Sprint 5 (20.03 – 31.03)

After a suboptimal previous sprint regarding development, the team wanted to come back stronger. In addition to the two unfinished PBIs from the last sprint, a lot of new development PBIs were added. The goal of this sprint was to get closer to the envisioned MVP, aware that easter and a lot of writing would consume much of the remaining time of the project. After the planning, the team conducted a meeting with the supervisor and received feedback on the report structure and decided that “end of code” for development would be the 28th of April.

Being really motivated, the team rapidly completed most of the development tasks in the first week, making a replanning necessary. Mostly writing tasks were added as the deadline for the final report in the other course IS-305 was closing in. Additionally, report feedback from the

supervisor was desired, so a bit more was done on that front, prior to sending it for reviewal. The need for defining the so far fluid scope of the project became apparent and invitations to a meeting was sent out. Furthermore, additional invitations were also sent regarding interviews for the IS-305 course.

At the end of the sprint, the system now had settings for notification types and a working producer for events (notifications), that simulated the expected behaviour of Elements. A refactoring of the component structure, and naming for files and methods in the notification micro frontend was also done to minimize technical debt. Furthermore, an investigation into how to solve subscriptions to specific cases was started, and a visual bug on notifications with short texts was fixed. To further ensure quality and consistency, a build pipeline for the backend repository was set up.

Even though a lot was done on the development front, the Sprint Review was held internally in the team with no demo. The retrospective was very positive as the team had good progress on both development and writing tasks. In addition, the scrum processes were well executed and there was a good amount of pair programming. As for what did not go so well, it was discovered that when merging the main branch in the frontend repository into the students_main, the commits were unfortunately squashed, overwriting a lot of the existing git history. This meant that when the time would come for merging students_main into the main repository, it must have been manually moved over at the cost of the team's development history. As a result, the team decided to make use of a blameless postmortem to investigate and analyse the incident. Resulting from this, a runbook was written for merging the two branches, describing how such an incident could be avoided.

6.2.6. Sprint 6 (03.04 – 14.04)

Since the first week of the sprint was easter break, the team had a mini Sprint Planning, straight after the precious sprint's retrospective. Here, each member was assigned only one writing PBI (IS-305) to ensure everybody got some time off, without falling behind in the course.

The first day back after the vacation, a meeting discussing the project's current and intended scope was held with multiple of Sikri's employees. They were pleased with the current situation and continued to emphasize the importance of prioritizing quality over quantity, advising not to add too many new features during the final stages. In their view, there was already delivered more than expected, and the team came to the agreement that after implementing subscription logic, focus would be placed towards refactoring and improving code quality.

Even though most of the two last scheduled workdays of the sprint was lost to conducting interviews for IS-305, the team managed to implement the backend logic for subscriptions as well as some minor refactoring. A rework of how new notifications got sent was also started. Originally, new notifications were sent to the frontend using SignalR, but after discussing back and forth with Sikri's developers, an agreement was reached that SignalR was to be used for triggering a fetch call to the web API, instead of sending the notification directly.

Finally, a review was conducted in unison with the steering committee meeting before the retrospective was held with just the team.

6.2.7. Sprint 7 (17.04 – 28.04)

Going into the last development sprint of the project as "end of code" had been set to the 28th of May, the feeling of closing in on the finishing line fueled the team. As a result, a record number of PBIs were created and added to the board during the planning.

Following Sikri's recommendation, the team mostly focused on refactoring and bug fixes, but were also able to implement some new functionality. Since users of elements could operate in different browser tabs, browsers, and devices, SignalR was used to ensure synchronous updates across the board, ensuring consistency. Further enhancements were also made to the subscription and link logic, and a frontend component for viewing and unsubscribing from cases was implemented.

The file containing all the API calls for our micro frontend was refactored to use the same fetching library (axios) for all the requests. The notification settings page had a resizing bug that was resolved, and the test data was updated to resolve an issue regarding the creation date of the notifications. As was done in the fronted repository earlier in the project, the backend was refactored with focus on structure and naming conventions for files, methods, and variables.

Towards the end of the sprint, the team agreed that instead of having a demo at the end of this sprint for just the team and the Product Owner representative, it was suited to conduct a demo for anyone interested in Sikri. Therefore, the demo was rescheduled to the 10th of May, where an invitation to all Sikri employees was sent out. As for the last Sprint Retrospective, it was mostly positive as the team were able to complete a large quantity of PBIs, both for development and writing tasks. Furthermore, the team was satisfied with the product created, and felt a sense of accomplishment. Due to working in an agile manner, it was important to look at what could be improved, even though there were no more development sprints. As the deadline rapidly approached, the quality of PR reviews unfortunately dropped, and a couple bugs were merged into the main branch as a result. Luckily, these were caught thanks to the pipeline, as well as manually testing all the functionality before handing the product over. This emphasized the importance of maintaining process quality, regardless of the situation, to ensure high standards.

6.3. Project Close

Although the final development sprint was concluded, the same sprint structure was kept for the project close. While this was after the end of code, some last code finalizations had to be done. On May 1st, the team had the last Sprint Planning, where a schedule of planned deliveries was developed. Included in these were final code implementations, report finalization, required system documentation, and a handover document.

At the beginning of the sprint, the priority was to finalize the IS-305 report, as this would help minimize context switching. Aware of a coming shift of attention mid-sprint after delivering the IS-305 report, a replanning took place on May 4th. Shifting to the bachelor report in IS-304, all remaining objectives in the report were separated into specific items on the board in Azure DevOps. In addition, a timeline was made including meetings and delivery dates.

As a closing segment of the project and cooperation with Sikri, the team presented the final product with a demo. This took place May 10th with an open invitation to all Sikri employees. The supervisor from the university was also invited and was present during the demo at Sikri's office. For those unable to attend physically, the demo was streamed and recorded. Incorporated in the demo was a display of main functionality and discussions regarding important processes during the project. Following the demo, the team, supervisor, and project owner representative held the last steering committee meeting. This included a summary of the project and discussing diverse topics regarding the report.

During the remaining time of the project, the team worked with finalizing the bachelor report as well as making final adjustments to the handover document. The latter were developed in consideration to project owner's wishes, to best facilitate quality and a smooth takeover for the developers continuing building the system. After completing the handover document, it was delivered to Sikri. Finally, submitting the report on May 16th marked the conclusion of the project.

7. Final Product

In this section, a description of the final product will be presented. This will include the system's functionality and what steps the team has taken to facilitate further development.

7.1. Functionality

The notification system that has been developed for Sikri's case management system, is designed to provide users with an efficient and customizable way to stay updated on relevant information. This section will primarily present the various functionalities offered by the notification system from a user perspective, instead of delving into the underlying logic that supports these features.

When a new notification arrives, users are immediately notified through a visual indicator in the form of a counter. This counter shows the number of unread notifications and is updated in real-time, keeping the users informed about the latest relevant events within Elements. To improve user experience, the list of notifications uses pagination. This allows the system to render small batches of notifications at a time, making the system more efficient. Every notification related to cases in the list contains a link, which when clicked, takes the user to the relevant case in Elements. As of now, the link will take the user to the "old" Elements case view, as the new UI is still under development. Furthermore, to provide users with

greater control over their received notifications**,** the system includes a filtering option. This allows them to easily switch between viewing "all" or just the "unread" notifications. This can be viewed in Figure 13.

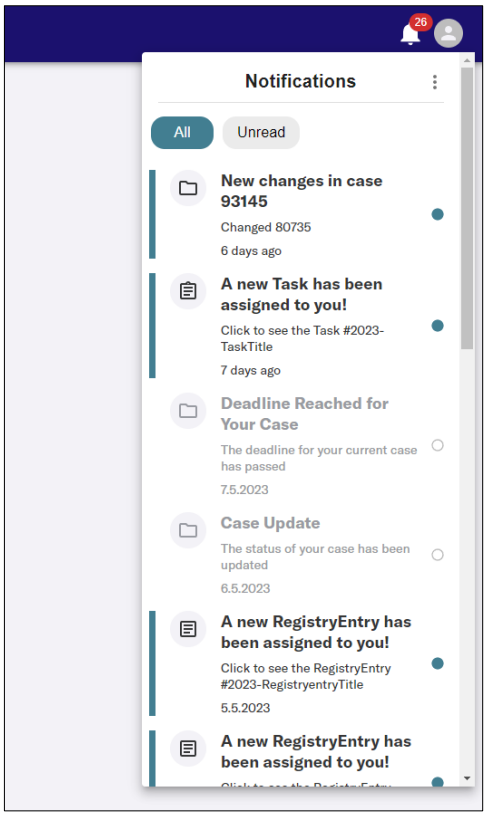


Figure 13: List of notifications

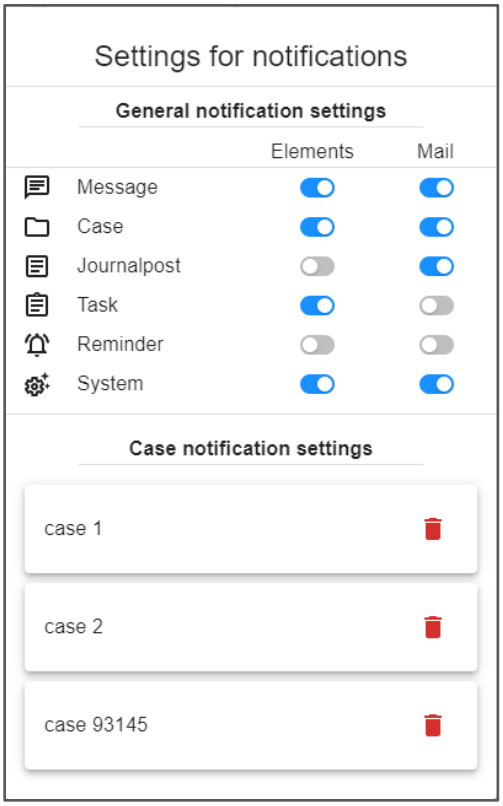


Figure 14: Settings for notifications

Users also have the possibility to mark individual notifications as "read" or "unread" by either clicking the notification (only marks it read), or the dot on the right side (can mark as both read and unread). This state change not only alters the appearance of the notification to easily differentiate, but also updates the unread counter. Additionally, users have the option to mark all notifications as read in a single action. Importantly, these functionalities that change a notifications state are synced across browsers and devices, ensuring a consistent experience no matter where or how the user accesses the system. These features enable users to manage their notifications effectively and stay organized.

Recognizing that users have varying preferences and needs, the notification system allows for customization of their notification settings. Users can choose to receive specific types of notifications, by changing their preferences in the notification settings tab shown in Figure 14. The system also enables users to manage their subscriptions to individual cases in the same tab, providing a list of currently subscribed cases and the option to unsubscribe if they are no longer interested.

Finally, to ensure that users are informed about the status of various parts of the notification system, feedback is provided when the system is experiencing issues, which enables users to be aware of any potential problems affecting their experience. These can be viewed in Figures 15 through 17 below. The notification system will also be able to maintain real-time

synchronization across different browsers and devices, ensuring that unread counters and notification read status remain consistent. This means it will work regardless of the device or browser utilized by the user, making it easier to integrate with multiple platforms.

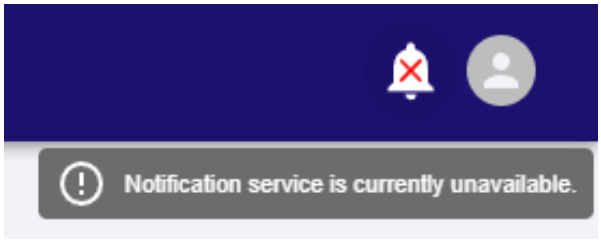


Figure 15: Hovering over notification bell

Figure 16: Error feedback

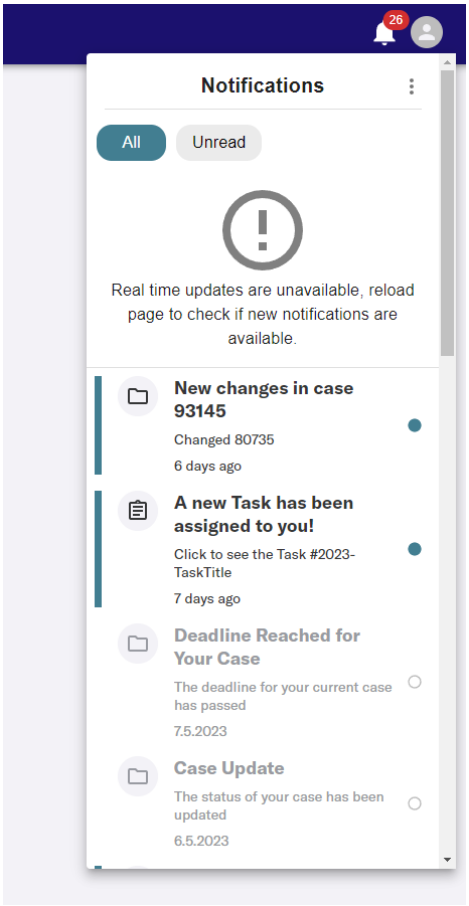


Figure 17: Feedback when SignalR server is down

7.2. Facilitating Further Development

Early in the project, it was made clear to the team that the solution developed was to be used by Sikri and their customers. Therefore, it was important to create a structure on how to best possible facilitate further development of the product. The team's aim was to produce a product of high quality that Sikri could build upon and expand.

To ensure that the most important functionality was developed first, the team used the prioritized user stories. This list was provided to Sikri, serving as a guide for future development. Showing the progress made and the tasks yet to be accomplished, the user stories informed Sikri about the project's status.

In terms of documentation, the team prepared a handover document that outlines the expected behavior of the product, potential improvements, known faults, and To-Dos. This document provided Sikri with a better understanding of the product and how to continue working on it. Additionally, the Readme.md file in the different repositories included information on how to

run the different parts of the application, as well as information pertaining to the different API endpoints.

The team employed the design principles maintainable, testable, and flexible as mentioned in chapter 5.1 Architecture design criteria. For these, the focus throughout the project was on quality over quantity, with manual and written testing, as well as establishing pipelines being a key aspect of ensuring the product's quality. Early in the development process, it was decided to work on fewer functionalities but complete them with high quality, rather than to deliver many unfinished or low-quality ones. This was to give Sikri the best possible foundation for further development.

It was also included some functionality that was not defined in the team's POC, but that could be built upon by Sikri in the future for a more complete notification system. For example, settings for email notifications were added, even though a service for sending emails has not been implemented. Additionally, it is possible to view and unsubscribe from cases in the micro frontend. The endpoint for subscribing is also ready, but Sikri will have to implement the UI for this as it is outside of the team's scope. Inclusion of the endpoint makes it simpler for Sikri to further develop the complete functionality for subscriptions.

Lastly, the team had access to several of Sikri's repositories that made it possible to follow their code standards. This included following their established practices for repository history, naming conventions, and test structure. Ensuring this throughout the project made sure that further development and handover went smoothly for both parties.

8. Reflection

In this chapter, the team will reflect on the work that has been done this past semester. Some of the challenges faced, the important decisions that was made, and how these decisions affected the final product will be discussed.

8.1. Process and methods

"Process and methods" will present and discuss some of the central decisions the team made related to processes and methods for ensuring quality and control in the project. By reflecting upon these decisions, the aim is to provide insights into the rationale behind choices and the impact they had on the overall success.

8.1.1. Methodology

As mentioned in chapter 2.1 Methodology, the choice of Scrumban as the development methodology was influenced by a few factors. First and foremost, Scrumban is the standard at Sikri, which played a significant role in the decision, as it allowed both the team and Sikri to be on the same page with the development process. Additionally, it enabled working

iteratively and flexibly by combining the aspects of both Scrum and Kanban that best suited the project's needs and requirements.

When it comes to alternative methodologies that could have been chosen, there was an option to either use pure Scrum or pure Kanban. However, the team believed that the additional flexibility provided by the Scrumban method would be more beneficial for the project. While Kanban offers greater flexibility than Scrum, it does not provide the same level of structure and guidance for managing software projects (Rehkopf, 2023). As a result, it was decided to adopt a hybrid approach, combining elements of both Scrum and Kanban to achieve the right balance of flexibility and structure.

One challenge faced with the methodology was that there were periods with an empty Kanban board, leading to the need for re-planning in the middle of the sprint. This could potentially have been solved by adopting a more Kanban-focused approach using the “pull-based” system for adding new PBIs. This again might have led to longer Sprint Plannings, as the team would have needed more PBIs ready in the backlog.

Throughout the project, there have been made incremental improvements to the various ceremonies, including Sprint Planning, and daily stand-ups. For instance, in the first couple of sprints, during Sprint Planning, some PBIs were created ad-hoc without a strong connection to a user story. This was mostly due to poorly defined user stories in which the scope was too big. As the project progressed, the team tweaked both the user stories to be more granular, as well as the process of creating PBIs by ensuring user stories were the foundation for any new PBI. This made Sprint Planning more effective as the team now could easily take in the highest prioritized user stories and create PBIs from these. When it comes to daily stand-ups, during the first sprint they only consisted of each team member sharing their progress and challenges thus far. However, as the project progressed, "walk the board" was incorporated, which made it easier to remain focused on the work items on the board and avoid distractions.

Overall, the team is satisfied with the choice of methodology despite facing some challenges along the way. These challenges were proactively addressed and changes to the methods were made as needed.

8.1.2. Local Development Environment

A reoccurring dilemma was choosing between using local, temporary solutions or integrating with Sikri’s cloud environment during development. For the database, the team decided that it was best to go for a local implementation. Being independent from the existing monolithic database meant having fewer dependencies and more control over the iteration process. This decision was supported by the developers at Sikri, as the database schema could easily be implemented in their monolith and the database interface in the project’s backend repository updated to support its use.

Following the reasons for choosing a local database implementation, a local queue was developed. In addition, the events that were generated in Elements and placed in the queue

contained too little information for the proposed solution, making a local implementation even more appealing. As a result of moving forward with the local alternative, the team was able to experiment and modify independent of Sikri's environment limitations.

8.1.3. Processes

This section will discuss the different processes used by the team throughout the project to ensure quality and control in both work and project management. It also highlights the commitment to continuous improvement, emphasizing the iterative effort to improve different processes throughout the project.

When encountering an error during the merging process with the main branch, the team performed a blameless postmortem. As a preventive measure to reduce the chance of a similar incident happening in the future, a runbook detailing the proper merging process was created. This could be referred to later if anyone had to go through the process again. Furthermore, the issue was discussed with the leader of the frontend team and asked for permission to merge students_main into the main branch.

In addition to the postmortems, the team maintained structure and ensured that all requirements were met by following a well-defined lifecycle for PBIs as outlined in the "Definition of Done" document (see Appendix 8). As the project progressed, these processes were continuously improved based on experiences and feedback from team members. For instance, PBI requirements were refined to include more detailed acceptance criteria and clearer definitions of the tasks to be accomplished. This allowed for better estimation and improved collaboration among the members. The pull request review process was also improved by providing more explicit guidelines for code smells, test review, and manual testing. This made the review process more efficient and ensured a higher quality of code in the final product.

It is also important to discuss the decision to consider the code as "done" when it has gone through all the processes described in the "definition of done" document as well as being merged into the main branch. Although acknowledging the potential benefits of using Sikri's different development environments such as dev and test, the team opted against it to avoid extra complexity in addition to already having chosen to use a local environment for the database. Moreover, it was decided not to include Sikri's testers in the Definition of Done, mainly due to them being a scarce resource.

Another important aspect of the project was addressing technical debt, which refers to the accumulation of shortcuts and sub-optimal decisions in the development process that eventually hinder progress. In order to pay down technical debt, time was allocated periodically in sprints to revisit and refactor code.

Furthermore, the team constantly sought to improve processes by reflecting on performance in Sprint Retrospectives and discussing any areas for improvement. During these retrospectives, the team assessment feature in Azure was utilized to measure the efficiency of some processes by presenting team members statements like "I feel safe and do not fear

making mistakes" and "Tools/resources/processes/procedures allow me to effectively meet my customers' needs". This helped identify potential inefficiencies, bottlenecks, and other issues in the processes and implement changes to address them.

Overall, there was a feeling that the processes utilized, as well as the commitment to continuously improving them, had a positive impact on the project. Furthermore, centralizing the processes related to a PBI's lifecycle in the "Definition of Done" document proved to be a great help to the team.

8.1.4. Estimation and Time Tracking for Product Backlog Items (PBIs)

The team adopted a simple estimation technique for PBIs, primarily for making sure each PBI on the kanban board was not larger than a typical day's work (8 hours). Even though the team were conscious and acknowledged the relevant literature on the importance of estimation and time tracking in project management, a choice was made to not make this a big priority for this project based on a couple of arguments. First, the team has during this bachelor's project worked as in-house developers rather than hourly consultants which means no-one is being charged by the number of hours worked. DeMarco and Lister (2013) also argue in "Peopleware: Productive Projects and Teams" that overly precise estimation and time tracking might lead to increased overhead and reduced productivity in software development projects.

Since estimation is not the primary metric for maintaining control over the project's progress, several other alternative measures were employed to track the remaining work. These measures included:

1. Regularly updating the Product Backlog: The team continuously refined the Product Backlog, adding or removing tasks as necessary. This helped maintain a clear view of the project's scope and the work left to complete.
2. Conducting daily stand-up meetings: Daily stand-up meetings were held to discuss each team member's progress and any obstacles encountered. This practice fostered communication and collaboration, ensuring that the team always was aware of the project's status.
3. Reviewing progress in Sprint Reviews and retrospectives: At the end of each sprint, both reviews and retrospectives were conducted to evaluate progress, performance and identify areas for improvement. These sessions helped fine-tune processes and maintain a clear understanding of the work remaining in the project.

8.1.5. Usage of a Separate "main" Branch Frontend

When developing the micro frontend, the team were supposed to work directly in the existing mono repository. Being aware that other developers at Sikri were currently working in the frontend repository, the team wanted to make sure that they did not introduce any breaking changes or negatively impact Sikri's progress and workflow. Therefore, a decision was made to use a separate branch that made it possible to develop code separately without impacting the main branch.

Looking back, the worries about interrupting other developers' workflow were unnecessary, as the components in the frontend repository were already separated. This separation meant that the development was unlikely to introduce breaking changes to the other developers' work. The use of a separate branch did come with some challenges, such as the need to periodically merge updates from the main branch into the separate branch, which resulted in several merge conflicts. Additionally, the continuous integration (CI) pipeline for the frontend repository was not configured to deal with the separate branch, causing the pipeline pass rate to drop as it failed every time a change was pushed to the students_main branch.

8.2. Product Decisions

This section will reflect on important product decisions that were made throughout the project. These include choices regarding analysis, design, and use of technology that directly affected the outcome of the final product.

8.2.1. Prioritization of Functionality

The prioritization of user stories consisted of fundamental decisions directly affecting the product. This was done in cooperation with Sikri, to best facilitate a correct understanding. Due to continuously developing a greater understanding, rework was undertaken while in the early phase of development. Within this process, the stories became smaller and more specific, so they could be easier managed. Looking back, this could have been done more often, iteratively updating the document to further ensure correctness regarding the requirements and specifications.

Even though not reflected in the final prioritization of the user stories, there were decisions made during the development phase that excluded some features from the scope. The reason for this was that the team had created user stories for a more complete service in the hopes of delivering more than expected. An example was sending notifications through e-mail, but as there simply was not enough time, it was removed from the scope. Since the changes and decisions regarding prioritizations should reflect and represent the product, there should have been more iterations and official meetings related to this. However, this turned out might be a blessing in disguise for further development, as the final list of user stories outlines a more complete service.

8.2.2. Message Design

The design of the message contracts within the queue was important as it set the requirements for generating notifications. Through the development, how generic or specialized the messages should be, was a recurring consideration. Early on, the message contract was constructed much like the notification object, allowing for rapid development. In later iterations, a new message contract was added based on an event Sikri currently published in their queue. Seeing clear use cases for both, the team proposed a design with a generic message based on the requirements of the microservice, and a proof of concept on how to

utilize the existing specialized message designs. Working with the generic message contract proved to be somewhat tedious. When the contract changed, tight coupling meant changes were needed elsewhere. This is something to be mindful of in future development.

8.2.3. SignalR

The system requirements highlighted the need for real-time notifications. While SignalR was the preferred library from Sikri, the team assured that this technology would fulfill the needs. While other solutions like Socket.io and pusher could also cover the requirements, it made sense to go for Sikri's preference as they already had SignalR in existing solutions. Early implementation consisted of publishing notifications to the client using a separate SignalR server which resided as its own project in the NotificationHub repository. Later, it was brought to attention that Sikri had their own SignalR microservice, and a decision was made to shift and use this instead. Since this service had differences, it required slight changes in both the micro frontend and micro service. Even though this transition came with extra effort, it facilitated nicely for further development as this part of the system now was integrated in Sikri's existing service. Sometime after this, the responsibility of SignalR was also changed. Moving away from sending notifications directly, SignalR would now be responsible for triggering clients to send a request to the backend for the new notification. This modification considered the architectural design criteria "efficient" and "comprehensible" presented in chapter 5.1, by reducing traffic payload and making clear separation of duties between SignalR and the API.

8.2.4. Pagination

Retrieving a user's notifications was something that would occur frequently, leading to discussions to optimize this process. Clients would preliminarily make calls to retrieve all their notifications, but as the total number of these would increase over time, this could introduce heavy traffic. Therefore, pagination was implemented. By using pagination, the team could define a specific amount of notification to be retrieved on initial load, reducing the size of the payload. When a user gets close to the bottom of the notification list when scrolling, a new call to the API is made, fetching the next defined number of notifications. This way of doing it could result in a greater number of calls to the API, but with an expected optimalisation of the system in the long run where countless notifications exist.

8.2.5. Subscription

The system requirements developed in the analysis made clear the expected functionality regarding subscription to cases. In development on the other hand, the implementation of this came with multiple difficulties. Since the other components in the micro frontend UI were still under development, none of them contained cases, thus the option to subscribe to a case did not exist. In addition, there was unclarity surrounding the functional requirements for its implementation. The UX team were in the process of designing the entirety of the new Element UI, which reasoned for an unfinished design regarding the display of subscriptions

in the system. These challenges combined hindered the development, leading to a different outcome then first expected.

In consideration of the circumstances, the team implemented the possibility for subscribing to cases through an API, in which other micro frontends could utilize in the future. As a feature related to the presentation of subscriptions in the UI, the user should be able to modify their notification settings related to individual cases. This ended up being down prioritized due to the limited time frame and unclear requirements. Despite the lack of function in the UI for subscribing to cases, the team think that the decisions taken made the most out of the circumstances as well as the time frame of the project.

8.2.6. Linking

It was clear from the beginning that the notifications presented to the user should link to the corresponding location in the UI. However, this implementation experienced difficulties. One of the issues was a result of Elements having multiple domains. From a backend perspective, it was difficult to generate links due to the domains being unknown. One of the message designs posed a solution of bringing this information from the sending service, but this set requirements for other services outside of the project's scope. Common within the URLs for the domains were identifiers used to navigate to a certain case, registry entry, or document. Since these were present within the messages consumed from the queue, it was decided to bring them into the notifications sent to the client. This enabled the client to construct the links, only using predefined domains. As the new Elements was still under development it did not contain actual cases to navigate to. Therefore, in the proof of concept, the links were generated towards the old Elements UI presenting the corresponding page based on the notifications content. This implementation did not fulfill a final solution, as it utilized partially hard coded URLs due to the lack of logic towards the different domains. Nevertheless, the implemented logic for generating the links can be used as a base and be built upon in further development.

8.2.7. User Interface and Design

In the early stages of the project, it was made clear that the design of the new Elements UI was Sikri's UX team's responsibility. Due to this, there was little emphasis on design from the team throughout the project. This meant that greater focus could be held towards functionality and backend development. Throughout the project there was uncertainty regarding design as the UX team were in the process of making these. As a result of this the team decided to put a bit more effort into design than initially thought, to exchange ideas with the UX team. The design of the POC ended up being a combination taking inspiration from both sides. The UI for the notification settings ended up with an unfinished design as it was implemented late in the project where refactoring and ensuring quality of the functionality took precedence. Although having more complete designs would likely have increased the efficiency of the development process, the absence of this allowed the team to incorporate their own creativity into the product.

9. Conclusion

The team started off with limited to no knowledge regarding multiple of the different technologies chosen for the project, resulting in a steep learning curve. A certain level of domain knowledge was required and how notification services operate was also unknown, meaning there was a lot to digest and learn before any development could begin. From a learning point of view, this has been a blessing as there have constantly been new things to learn in a lot of different fields.

Looking back, there are multiple successful aspects regarding the project execution that we want to highlight. However, the most important one would be our ability to identify and continuously improve our processes and methods. Throughout the project, this has improved how we work as a team, resulting in a higher quality product. Additionally, it has provided the team with a greater understanding of project management as well as the technical aspects of software development.

In conclusion, the team is very proud of the outcome of the project and pleased with the positive feedback from Sikri found in Appendix 1: Statement from Sikri. It was also reassuring that both parties agreed that the requirements and expectations set out at the beginning of the project had been met to a high degree. Overall, this project has been a valuable and rewarding experience for our team, and we look forward to taking the skills and knowledge we have gained into future projects.

References

Agile Alliance. (2015). *What is Agile?* Retrieved from Agile Alliance: https://www.agilealliance.org/agile101/

Agile Alliance. (n.d.). *User Story Template*. Retrieved April 12, 2023, from Agile Alliance: https://www.agilealliance.org/glossary/user-story-template/

Agrawal, H. (2022, August 18). *What is TypeScript and why should you use it?* Retrieved March 23, 2023, from contentful: https://www.contentful.com/blog/what-is-typescript-and-why-should-you-use-it/

Altexsoft. (2021, May 18). *Acceptance Criteria for User Stories: Purposes, Formats, Examples, and Best Practices*. Retrieved April 12, 2023, from Altexsoft: https://www.altexsoft.com/blog/business/acceptance-criteria-purposes-formats-and-best-practices/

Association for Project Management. (2022). *What is project Management?* Retrieved from https://www.apm.org.uk/resources/what-is-project-management/#:~:text=Definition,a%20finite%20timescale%20and%20budget.

Aston, B. (2023, March 14). *10 Reasons Why Project Management Is So Important For Orgs*. Retrieved May 12, 2023, from dpm: https://thedigitalprojectmanager.com/personal/new-pm/why-is-project-management-important/

Atlassian. (2018). *What is scrum?* Retrieved from Atlassian: https://www.atlassian.com/agile/scrum

Atlassian. (2019). *What is kanban?* Retrieved from Atlassian: https://www.atlassian.com/agile/kanban

Atlassian. (2020). *How to run a blameless postmortem*. Retrieved from Attlasian.com: https://www.atlassian.com/incident-management/postmortem/blameless

Atlassian. (2023). *Engineering higher quality through agile testing practices*. Retrieved from Atlassian: https://www.atlassian.com/agile/software-development/testing

Atlassian. (n.d.). *What is Agile?* Retrieved May 12, 2023, from Atlassian: https://www.atlassian.com/agile

Atlassian. (n.d.). *What is version control?* Retrieved May 13, 2023, from Atlassian Bitbucket: https://www.atlassian.com/git/tutorials/what-is-version-control

Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., . . . Sutherland, J. (2001). *Manifesto for Agile Software Development*. Retrieved April 12, 2023, from The Agile Manifesto: https://agilemanifesto.org/iso/en/manifesto.html

Burgan, S. C., & Burgan, D. S. (2014, October 26). *One size does not fit all: Choosing the right project approach*. Retrieved April 20, 2023, from Project managment institute: https://www.pmi.org/learning/library/choosing-right-project-approach-9346

Carter, K. (2023, January 17). *Coding is Not Enough: The Importance of Investing in Domain Knowledge*. Retrieved from Level Up Coding: https://levelup.gitconnected.com/coding-is-not-enough-the-importance-of-investing-in-domain-knowledge-a8afb690f758

Chec, M. (2020, November 03). *Walking the Board on Daily Scrum*. Retrieved May 12, 2023, from medium: https://medium.com/serious-scrum/walking-the-board-on-daily-scrum-5b468c760329

CMOE. (n.d.). *Team Performance Assessment*. Retrieved May 13, 2023, from Center for Management & Organization Effectiveness: https://cmoe.com/glossary/team-performance-assessment/

DeMarco, T., & Lister, T. (2013). *Peopleware: Productive Projects and Teams* . New York.

Deshpande, C. (2023, February 07). *The best guide to know what is react*. Retrieved March 23, 2023, from simplilearn: https://www.simplilearn.com/tutorials/reactjs-tutorial/what-is-reactjs

Eagle, A., Chau, K., Cross, J., Savkin, V., Gonzalez, P., Reock, J., & Weinberger, B. (2022). *Mnorepo Explained*. Retrieved from https://monorepo.tools/#what-is-a-monorepo

Fletcher, P. (2020, September 10). *Introduction to SignalR*. Retrieved March 23, 2023, from Microsoft: https://learn.microsoft.com/en-us/aspnet/signalr/overview/getting-started/introduction-to-signalr

Harris, C. (n.d.). *Microservices vs. monolithic architecture*. Retrieved from Atlassian: https://www.atlassian.com/microservices/microservices-architecture/microservices-vs-monolith

IBM. (2016). *What is software testing?* Retrieved from IBM: https://www.ibm.com/topics/software-testing

Indeed. (2022, December 10). *Indeed*. Retrieved from Quality Assurance in Project Management (With 3 Types): https://www.indeed.com/career-advice/career-development/what-is-quality-assurance-in-project-management

Indeed. (2023, March 10). *What Is System Analysis and Design? (Plus Benefits)*. Retrieved from Indeed: https://www.indeed.com/career-advice/career-development/what-is-system-analysis-and-design#:~:text=Benefits%20of%20system%20analysis%20and%20design&text=Enabling%20comprehension%20of%20complicated%20structures,the%20workload%20of%20IT%20employees

Indeed Editorial Team. (2023, February 03). *Guide To Assessing Teams (With Benefits, Elements and Tips)*. Retrieved May 13, 2023, from indeed: https://www.indeed.com/career-advice/career-development/assessing-teams

ISO. (2015). *ISO - Standards*. Retrieved from ISO: https://www.iso.org/standards.html

ISO. (n.d.). *ISO/IEC 25010*. Retrieved May 10, 2023, from ISO 25000: https://iso25000.com/index.php/en/iso-25000-standards/iso-25010

Jackson, C. (2019, June 19). *Micro Frontends*. Retrieved from martinFowler: https://martinfowler.com/articles/micro-frontends.html

Kajal. (2021, August 24). *Scrumban – Beginner's Guide to Scrumban Methodology*. Retrieved May 15, 2023, from Unichrone: https://unichrone.com/blog/agile/beginners-guide-to-scrumban-methodology/#What_is_Scrumban

Krawczyk, B. (2022, November 17). *What is quality assurance (QA) in software development?* Retrieved from LogRocket: https://blog.logrocket.com/product-management/what-is-quality-assurance-qa-software-development/

Lewis, J., & Fowler, M. (2014, March 25). *Microservices*. Retrieved from MartinFowler: https://martinfowler.com/articles/microservices.html

Lynn, R. (n.d.). *WHY USE KANBAN BOARDS?* Retrieved May 12, 2023, from planview: https://www.planview.com/resources/guide/introduction-to-kanban/use-kanban-boards/

Martin, R. C. (2018). *Clean Architecture: A Craftman's Guide To Software Structure And Design.* Pearson education.

MassTransit. (n.d.). *What is MassTransit?* Retrieved March 23, 2023, from MassTransit: https://masstransit.io/introduction

Mathiassen, L., Munk-Madsen, A., Nielsen, P. A., & Stage, J. (2018). *Object Oriented Analysis & Design.* Hadsund: Metodica.

McKinsey & Company. (2021, February 17). *When code is king: Mastering automotive software excellence*. Retrieved May 14, 2023, from McKinsey & Company: https://www.mckinsey.com/industries/automotive-and-assembly/our-insights/when-code-is-king-mastering-automotive-software-excellence

Microsoft. (2020, February 19). *Hands On Lab: Real-Time Web Applications with SignalR*. Retrieved May 13, 2023, from Microsoft: https://learn.microsoft.com/en-us/aspnet/signalr/overview/getting-started/real-time-web-applications-with-signalr

Microsoft. (2021, 05 25). *Entity Framework Core*. Retrieved from Microsoft Learn: https://learn.microsoft.com/en-us/ef/core/

Microsoft. (2022, September 21). *The C# type system*. Retrieved April 20, 2023, from Microsoft: https://learn.microsoft.com/en-us/dotnet/csharp/fundamentals/types/

Microsoft. (2022, October 10). *What is Azure DevOps?* Retrieved May 13, 2023, from Microsoft: https://learn.microsoft.com/en-us/azure/devops/user-guide/what-is-azure-devops?view=azure-devops

Microsoft. (2023, February 13). *.NET architecture*. Retrieved March 23, 2023, from Microsoft: https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/#net-architecture

Microsoft. (2023, February 13). *A tour of the C# language*. Retrieved March 23, 2023, from Microsoft: https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/

Microsoft. (2023). *Azure Pipelines*. Retrieved from Microsoft Azure.

Microsoft. (2023, October 02). *Unit testing C# in .NET Core using dotnet test and xUnit*. Retrieved May 01, 2023, from Microsoft: https://learn.microsoft.com/en-us/dotnet/core/testing/unit-testing-with-dotnet-test

Microsoft. (2023, March 23). *What's new in SQL Server 2022 (16.x)*. Retrieved April 25, 2023, from Microsoft: https://learn.microsoft.com/en-gb/sql/sql-server/what-s-new-in-sql-server-2022?view=sql-server-ver16

Microsoft. (n.d.). *Azure Boards*. Retrieved May 13, 2023, from Microsoft Azure: https://azure.microsoft.com/en-us/products/devops/boards/

Microsoft. (n.d.). *What is ASP.NET?* Retrieved March 23, 2023, from Microsoft: https://dotnet.microsoft.com/en-us/learn/aspnet/what-is-aspnet

MUI. (n.d.). *Material UI - Overview*. Retrieved March 23, 2023, from MUI: https://mui.com/material-ui/getting-started/overview/

Nath, A. (2023, April 10). *What is the Role of Communication in Quality Management?* Retrieved May 10, 2023, from tutorialspoint: https://www.tutorialspoint.com/what-is-the-role-of-communication-in-quality-management

Postman. (n.d.). *What is Postman?* Retrieved March 23, 2023, from Postman: https://www.postman.com/product/what-is-postman/

ProductPlan. (2021, September 09). *The Definition of Done: What Product Managers Need to Know*. Retrieved April 12, 2023, from ProductPlan: https://www.productplan.com/learn/agile-definition-of-done/

ProductPlan. (n.d.). *MoSCoW Prioritization*. Retrieved April 12, 2023, from ProductPlan: https://www.productplan.com/glossary/moscow-prioritization/

Przystalski. (2021, September 06). *Pair Programming: Pros, Cons, Best Practices*. Retrieved April 12, 2023, from Codete Blog: https://codete.com/blog/pair-programming-pros-cons-best-practices

Rehkopf, M. (2023). *Kanban vs. scrum: which agile are you?* Retrieved from Atlassian: https://www.atlassian.com/agile/kanban/kanban-vs-scrum

Rehkopf, M. (n.d.). *User stories with examples and a template*. Retrieved April 12, 2023, from Atlassian: https://www.atlassian.com/agile/project-management/user-stories

Roper, K. (2021, September 26). *Importance of System Analysis in Software Development*. Retrieved from Linkedin: https://www.linkedin.com/pulse/importance-system-analysis-software-development-kimone-roper?trk=public_profile_article_view

Sheth, H. (2021, March 22). *NUnit vs. XUnit vs. MSTest: Comparing Unit Testing Frameworks In C#*. Retrieved May 10, 2023, from LambdaTest: https://www.lambdatest.com/blog/nunit-vs-xunit-vs-mstest/

Simplilearn. (2023, February 24). *Guide To Using Typescript With React*. Retrieved May 13, 2023, from simplilearn: https://www.simplilearn.com/tutorials/reactjs-tutorial/react-typescript

Sirotka, A. (2022, March 15). *What is Material UI?* Retrieved March 23, 2023, from Flatlogic: https://flatlogic.com/blog/what-is-material-ui/

Sommerville, I. (2016). *Software Engineering* (10. ed.). Pearson.

Stanton, K. (2022, February 24). *Quality Assurance vs. Quality Control Explained: 5 key differences*. Retrieved from Qualio: https://www.qualio.com/blog/quality-assurance-vs-quality-control#:~:text=QA%20is%20process%2Doriented%2C%20and,results.

tcagley. (2013, November 07). *Questions I am Frequently Asked: What Does "Walking the Board" Mean?* Retrieved from Software Process and Measurement:

https://tcagley.wordpress.com/2013/11/07/questions-i-am-frequently-asked-what-does-walking-the-board-mean/

Toturialspoint. (n.d.). *Software Testing - Quick Guide*. Retrieved from Toturialspoint: https://www.tutorialspoint.com/software_testing/software_testing_quick_guide.htm

TutorialPoint. (n.d.). *MS SQL Server - Overview*. Retrieved April 28, 2023, from TutorialPoint: https://www.tutorialspoint.com/ms_sql_server/ms_sql_server_quick_guide.htm

University of Agder. (2021). *Bachelor Thesis in Information Systems*. Retrieved May 10, 2023, from University of Agder: https://www.uia.no/en/studieplaner/topic/IS-304-1

Userpilot. (2023, March 8). *What are User Surveys and How to Conduct One?* Retrieved from Userpilot: https://userpilot.com/blog/user-surveys/

Visual Paradigm. (2023, March 28). *Agile Methodology: Embracing Flexibility, Collaboration, and Continuous Improvement for Effective Project Management*. Retrieved from Visual Paradigm: https://guides.visual-paradigm.com/agile-methodology-embracing-flexibility-collaboration-and-continuous-improvement-for-effective-project-management/

xbosoft. (n.d.). *Definition of Software Quality*. Retrieved April 15, 2023, from xbosoft: https://xbosoft.com/software-qa-consulting-services/definition-software-quality/

Yousuf, F. (2022, October 10). *RabbitMQ vs Azure Service Bus – What's the Difference? (Pros and Cons)*. Retrieved May 14, 2023, from Cloud infrastructure services: https://cloudinfrastructureservices.co.uk/rabbitmq-vs-azure-service-bus-whats-the-difference/

Appendix

Appendix 1: Statement from Sikri

sikri

09.05.23

Statement from Sikri

An agreement was signed between the bachelor group and Sikri in December 2022. The bachelor group was tasked with creating a notification system that could be integrated with the various applications used by Sikri. The applications were complex and diverse, making this a challenging task.

Despite the complexity of the task, the bachelor group did an excellent job. They conducted a survey and interviews to gather information about the existing applications and their notification systems and worked hard to understand the specific details of the different systems and frameworks in use. They were able to leverage their technical expertise to create a solution that could be easily integrated into the existing systems.

The group validated their proposed solution through further discussions and a larger workshop. The bachelor group and a representative from the organization had regular meetings every other week, and also communicated in writing via Teams. The collaboration between the two parties was very successful, with the students being professional, punctual, well-prepared, and clear in their communication.

The bachelor group developed an MVP that functioned well and had more features than originally planned. Their work was invaluable, as they were able to create a solution that seamlessly integrated with the complex applications used by Sikri. The organization was very impressed with the group's technical prowess and their ability to deliver a high-quality solution. The inspiration and knowledge that the bachelor group provided will be used to further improve the notification system.

Best regards

Asbjørn Nordgaard
Product owner

Ammar Khaled Haddad
Product owner representative

Sikri AS • Dronning Mauds gate 10, 0250 Oslo
905 30 049 • post@sikri.no • www.sikri.no
Org.nr.: 922 308 055

Appendix 2: Team Evaluation

Throughout the project, all members have brought great efforts and contributions. As a team where everyone has strong opinions, we have facilitated open communication to view all perspectives. The team members had varying responsibilities with the goal of best utilizing the individual team members' strengths. All members have actively participated in the managerial activities and conducted technical processes such as reviewing pull requests and pair programming. Regarding development, all members contributed to various parts of the system ensuring a high understanding of the entire system. As a team, we are proud of the individual efforts and high work ethic which led to the success of the project. The following subchapters further outline the team members' individual contributions.

Aleksander

In this project, my primary role centered around backend development with a more specific focus on implementing service bus, with MassTransit as an example. Although my primary role was in backend, I also worked on the frontend where one example is that I had the responsibility of ensuring that the states of notifications for a user would synchronize across multiple browsers and devices.

Serving as the Scrum Master for our team, I facilitated Scrum ceremonies like planning, retrospective and daily standup. Furthermore, I was also responsible for setting up the two Continuous Integration pipelines for the backend repository.

Hermann

Throughout this project my main responsibility has been the micro frontend implementation. To some extent I touched upon most of the components and functionality here but worked the most on creating a skeleton for the micro frontend, the notification component, and the notification list. I also did a lot regarding pagination and the SignalR client. Even though most of my contributions came on this front, I also developed for the backend, for example implementing pagination for retrieving notifications.

I was appointed as the team leader, but as a flat hierarchy was preferred, most of the responsibilities ended up being shared amongst the team. Looking back, I was involved with a lot of the project's different aspects resulting in a great learning experience.

Bjørnar

I have had most of my focus toward building the Web API of the project. This involved development of the various layers from the controller to the database. While minimal involvement on the client side, I added authentication onto the requests, as I introduced authorization of the controllers in the backend.

Kristoffer

As an individual team member, I have primarily been a backend developer with a special focus on architecture and how our system interoperates. I have worked much on SignalR, and some on database and queue. I have done little frontend work, except work on the SignalR client but have helped with debugging on occasion.

Lars

The main responsibility I had in this project was around frontend development, with the most work done on settings, filtering and changing of read state. The work done on settings was only situated frontend, making their design and functionality, and connecting it up to the API endpoints made by Bjørnar. Filtering of the notifications was originally a frontend solution but was later changed largely by Hermann to have a more backend-oriented solution. Changing the read state of a notification required work both front- and backend to work. These were the largest specific things done by me, and other than this I have done work here and there in both front- and backend.

Appendix 3: Group Contract

Group Contract

We in group 4, consisting of following members:

Aleksander Dokken,

Bjørnar Olsen-Hagen Sømme,

Hermann Rødsedt Theimann,

Kristoffer Stokkeland,

Lars Blåsmo Husfloen

Will work together in the following subjects:

IS-304 Bacheloroppgave i informasjonssystemer,

IS-305 Aktuelle IT-relaterte tema, bærekraft og digitalisering

Agreements on attendance and cooperation:

I commit to my group that I will:

- Attend lectures, participate in group work, meetings and exercises, ask for/accept guidance, do self-study of literature, and conduct design research; actively participate in group field studies.
- Respect the requirement of at least 80% attendance and also provide a reason if I am prevented. 15 minutes or more late is considered as not attending/absent.
- Meet deadlines and submit assignments published on Canvas, within the deadlines.
- Ensure that work is distributed evenly among group members, but at the same time utilize each student's special skills and background. Any conflicts regarding cooperation and effort will be resolved through discussion in the group.

Other agreements:

Group leader: Hermann Theimann

- The group leader acts as the contact person of the group.
- If a democratic voting ends in even numbers, the group leader has the last word.
- The group leader is responsible of scheduling meetings.

Scrum master: Aleksander Dokken

- The scrum master is responsible of holding daily scrums.
- The scrum master has the main responsibility of producing project related documents, such as meeting reports.
- The scrum master takes place as group leader in the absence of the actual group leader.

Place, time:

Kristiansand, 16.01.2023

Underskrifter:

Aleksander Dokken

Bjørnar Olsen-Hagen Sømme

Hermann Rødset Theimann

Kristoffer Stokkeland

Lars Blåsmo Husfloen

Appendix 4: Risk Analysis

			Probability				
			Rare	Unlikely	Possible	Likely	Highly likely
			1	2	3	4	5
Consequence	Insignificant	1	1	2	3	4	5
	Less	2	2	4	6	8	10
	Moderate	3	3	6	9	12	15
	More	4	4	8	12	16	20
	Catastrophic	5	5	10	15	20	25

Nr.	Risk	Probability	Consequence	Impact	Response	Comment
1	Minor sickness	5	1	5	Accept	Work from home if possible
2	Long-term sickness	2	4	8	Accept	Distribute the person's tasks to the rest of the group
3	Communication failure with Sikri	2	3	6	Avoid	This is thought to be small communication gaps with associated low consequences
4	Communication failure internally in the group	1	2	2	Avoid	The group meets almost every day for school purposes, and 80% also live together
5	Wrong priorities	2	4	8	Limit	Will use Sprint Planning and part of the associated processes. For example, backlog, Scrum Master, etc.
6	Unnecessary use of time	4	2	8	Limit	Scrum Master. Asking for help
7	Avoidance/poor motivation	3	1	3	Accept	Motivate each other along the way, and collectively make sure that everyone contributes
8	Conflicts within the group that prevent progress	1	4	4	Avoid	The group consists of people who dare to speak up when there is something, so it is important to be open to conversations and take them in a professional manner. We have daily meetings where any

						problems can be raised so we can solve them early
9	Turning up late	3	2	6	Accept	Showing up late every now and then is something that happens. If it happens frequently and is also not reported, it will escalate into a more serious problem
10	Lack of knowledge and skills	4	2	8	Limit	Use each other, the internet, and available resources at Sikri when help is needed
11	Different coding standards	2	2	4	Avoid	Agree on a coding standard before we start
12	Low quality code	4	4	16	Limit	The quality of the code will probably increase with the time we spend on the project. This means that it will gradually improve with the semester. It will therefore be natural to fix older code over time. Usage of code reviews
13	Low quality testing	4	5	20	Limit	The group has some knowledge of testing, but not to the extent required for this project. Naturally, this will improve throughout the semester through experience and improvement of our skills
14	Faulty equipment	2	3	6	Accept	Is largely outside our control and something that can always happen
15	Improper use of software	2	2	4	Avoid	Build up knowledge before use
16	Poorly structured work	1	4	4	Avoid	Structure our work well and have a good dialogue between us throughout the sprints. Have an overview of

						which task you are working on at the time
17	Loss of data	2	4	8	Avoid	Have regular backups of your work. Using Azure can make it much easier
18	Low security standard	2	4	8	Avoid	Follow industry standards
19	Breach of confidentiality	1	5	5	Avoid	Do not share information about Sikri or the project with outsiders
20	Weak documentation	4	3	12	Limit	Documenting the code will make it easier for both group members and employees at Sikri to know what is going on without reading all the lines. Is good training and makes handover easier
21	Stuck on a task	5	2	10	Avoid	This will happen, and the group together with Sikri must ensure that there is a low threshold for asking for help
22	Loss of a team member	1	5	5	Accept	Highly unlikely that it will occur, but in the event the person must be removed from everything involved with the project and the person's tasks must be distributed among remaining members
23	Wrong scope	2	3	6	Limit	A dynamic process
24	Rushing decisions	3	2	6	Limit	Most likely to occur if group is tired from using (too) much time on other decisions.

Appendix 5: User Stories

Must Have

#	User Story	MOSCOW
1	As a user, I want to view my notifications in the pop-up from the bell icon, so that I can easily get an overview of both old and new notifications.	Must have
2	As a user, I want to be notified in Elements when a case is assigned to me, so that I can start working on it.	Must have
2.1	As a user, I want to be notified when new relevant documents arrive in a case I am working on, so that I can stay up to date on the progress of the case.	Must have
2.2	As a user, I want to be notified when others approve or disapprove an item that I have sent for approval, so that I know the status of the item.	Must have
2.3	As a user, I want to be notified when an item I am responsible for is approaching a deadline, so that I can ensure it is completed on time.	Must have
2.4	As a user, I want to be notified when other users change the metadata on an item I own, so that I can review the changes.	Must have
3	As a user, when online in Elements, I want new notifications to be added to the notification list in real-time, so I can view these without having to reload the page.	Must have
4	As a user, I want to see the number of unread notifications on the bell icon, so that I can quickly get a view of how many notifications I haven't looked at.	Must have
5	As a user, I want to be able to receive notifications through e-mail, so that when I am offline in Elements, I can still be updated.	Must have
6	As a user, I want to be redirected to the relevant page when I click on a notification, so that I don’t need to manually find the page myself.	Must have
7	As a user, I want to be able to subscribe to a specific case, so I can get notified when changes in this occur.	Must have
8	As a user, I want to be able to unsubscribe from a specific case, so that when I no longer need notifications regarding it, I can easily choose so.	Must have
9	As a user, I want to see when a notification was created, because it is an important to know how long ago, I received it.	Must have
10	As a user, I want to clearly view what notifications are read and unread, so that I have control over which notifications I have looked at or not.	Must have
11	As a user, I want notifications to be marked as read after clicking on it, so that I can keep track of the notifications I have received and viewed in my inbox.	Must have
12	As a user, I want to modify my overall notification settings to either send notifications to Elements, to my e-mail, or both, so I can choose based on my preference.	Must have
13	As a user, I want to modify my notification settings over types of notifications, so that I for example can turn on notification of being assigned to a case but don’t get notified when it is updated, so that I don’t have to do this on all independent cases.	Must have

14	As a user, I want to view all my notification settings in a central place, so that I can easily modify them and won't need to navigate to different pages to do so.	Must have
15	As a user, I want to have the ability to manually mark a single notification as read without opening the notification, so that when I know I don't need to investigate the notification, I can quickly check it as read.	Must have
16	As a user, I want to have the ability to manually mark a single read notifications as unread, so that I can make it unread again if clicked by accident or I did not have the time to "complete the task"/"read the information" related to the notification.	Must have
17	As a user, I want to get feedback if the initial load of notifications is not working, so I know why none of my notifications are visible in the log.	Must have
18	As a user, I want to get feedback if the real-time notification service (SignalR server) is down, so I know that I won't get new notifications without reloading the page.	Must have
19	As a user, I want to get feedback if the notification component (micro frontend) is not working, so I know why I can't view my notifications.	Must have

Should Have

#	**User Story**	**MOSCOW**
20	As a user, I want to mark all notifications as read, so that when I don't need to investigate all new notifications, I don't need to click on all of them individually.	Should Have
21	As a user, I want to get a periodic summary e-mail of notifications, so I my inbox won't get spammed with e-mails.	Should Have
22	As a user, I want to see an icon of the type of notification, so that I don't have to read through the notification to understand what type it regards.	Should Have
23	As a user, I want to have the ability to fine-tune my notification settings on an individual case, to have more granular control over the information I receive.	Should Have
24	As a user, I want to have the ability to change my e-mail notification settings, so that I can choose when I receive them, to not be disturbed outside working hours.	Should Have
25	As a user, I want the contents of the notification to mark out the most important, so that it is easy to understand what the main contents of the notification is.	Should Have
26	As a user, I want to be able to change the language to meet my preference, so I understand what is written.	Should Have
27	As a user, I want to be able to filter between all and unread, so that I easily get an overview of the notifications I haven't looked at.	Should Have
28	As a user, I want to turn off all notifications for a set time interval so I can better focus on my tasks.	Should Have

29	As a user with administrative privileges, I want to control the notification settings of a group I am responsible for, so that these can't be changed by the users and ensures that they receive the notifications they are supposed to.	Should Have
30	As a user with administrative privileges, I want to set the default notification settings of a group I am responsible for, so that they have the same base settings but can change them as they want.	Should Have
31	As a user, I want to be notified when something I have sent through digital channels such as the post's 'Reply Out' service fails, so that I can take appropriate action.	Should Have
32	As the owner of Elements, I want to push out notifications to all users of critical information, so that they are aware of things that can affect the Elements environment, such as updates and bugs under investigation.	Should Have

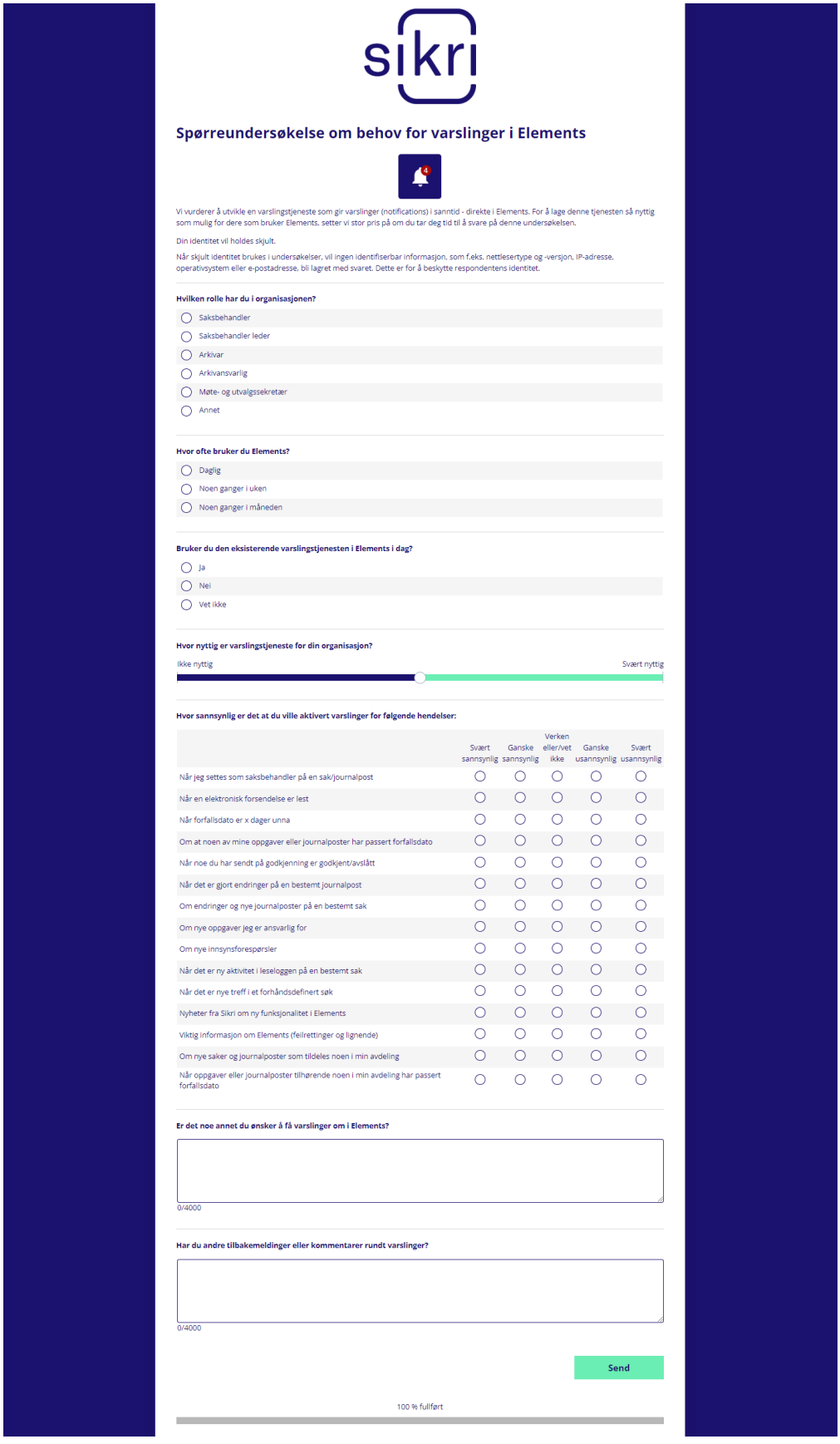
Could Have

#	**User Story**	**MOSCOW**
33	As an administrator, I want the ability to push out custom notifications to all users I'm responsible for, so that I can inform them of important information regarding their work.	Could Have
34	As an administrator, I want the ability to push out custom notifications to certain groups or users, so that I can inform them of important information regarding their work.	Could Have
35	As a user, I want to find information on how to modify the notification settings, because I often find it hard to get familiar with new technical functionality.	Could Have

Won't Have

#	**User Story**	**MOSCOW**
36	As an administrator, I want to be able to generate reports about the notifications (such as the number of notifications sent, the number of recipients, and the types of notifications), so that I can monitor the system behavior.	Won't Have

Appendix 6: User Survey Form



sikri

Spørreundersøkelse om behov for varslinger i Elements

Vi vurderer å utvikle en varslingstjeneste som gir varslinger (notifications) i sanntid - direkte i Elements. For å lage denne tjenesten så nyttig som mulig for dere som bruker Elements, setter vi stor pris på om du tar deg tid til å svare på denne undersøkelsen.

Din identitet vil holdes skjult.

Når skjult identitet brukes i undersøkelser, vil ingen identifiserbar informasjon, som f.eks. nettlesertype og -versjon, IP-adresse, operativsystem eller e-postadresse, bli lagret med svaret. Dette er for å beskytte respondentens identitet.

Hvilken rolle har du i organisasjonen?

- ◯ Saksbehandler
- ◯ Saksbehandler leder
- ◯ Arkivar
- ◯ Arkivansvarlig
- ◯ Møte- og utvalgssekretær
- ◯ Annet

Hvor ofte bruker du Elements?

- ◯ Daglig
- ◯ Noen ganger i uken
- ◯ Noen ganger i måneden

Bruker du den eksisterende varslingstjenesten i Elements i dag?

- ◯ Ja
- ◯ Nei
- ◯ Vet ikke

Hvor nyttig er varslingstjeneste for din organisasjon?

Ikke nyttig — Svært nyttig

Hvor sannsynlig er det at du ville aktivert varslinger for følgende hendelser:

	Svært sannsynlig	Ganske sannsynlig	Verken eller/vet ikke	Ganske usannsynlig	Svært usannsynlig
Når jeg settes som saksbehandler på en sak/journalpost	◯	◯	◯	◯	◯
Når en elektronisk forsendelse er lest	◯	◯	◯	◯	◯
Når forfallsdato er x dager unna	◯	◯	◯	◯	◯
Om at noen av mine oppgaver eller journalposter har passert forfallsdato	◯	◯	◯	◯	◯
Når noe du har sendt på godkjenning er godkjent/avslått	◯	◯	◯	◯	◯
Når det er gjort endringer på en bestemt journalpost	◯	◯	◯	◯	◯
Om endringer og nye journalposter på en bestemt sak	◯	◯	◯	◯	◯
Om nye oppgaver jeg er ansvarlig for	◯	◯	◯	◯	◯
Om nye innsynsforespørsler	◯	◯	◯	◯	◯
Når det er ny aktivitet i leseloggen på en bestemt sak	◯	◯	◯	◯	◯
Når det er nye treff i et forhåndsdefinert søk	◯	◯	◯	◯	◯
Nyheter fra Sikri om ny funksjonalitet i Elements	◯	◯	◯	◯	◯
Viktig informasjon om Elements (feilrettinger og lignende)	◯	◯	◯	◯	◯
Om nye saker og journalposter som tildeles noen i min avdeling	◯	◯	◯	◯	◯
Når oppgaver eller journalposter tilhørende noen i min avdeling har passert forfallsdato	◯	◯	◯	◯	◯

Er det noe annet du ønsker å få varslinger om i Elements?

0/4000

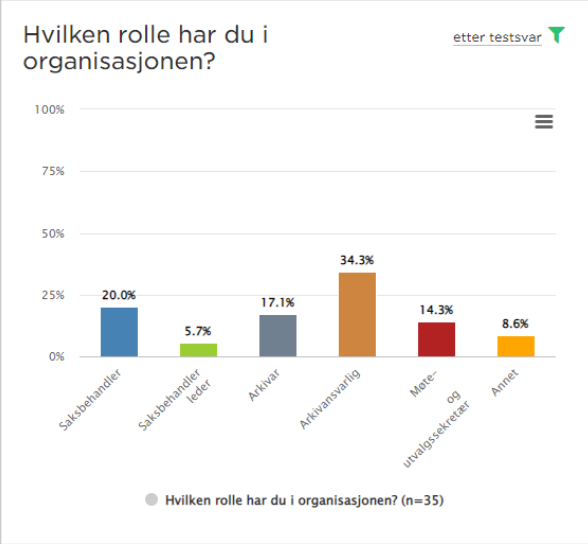
Har du andre tilbakemeldinger eller kommentarer rundt varslinger?

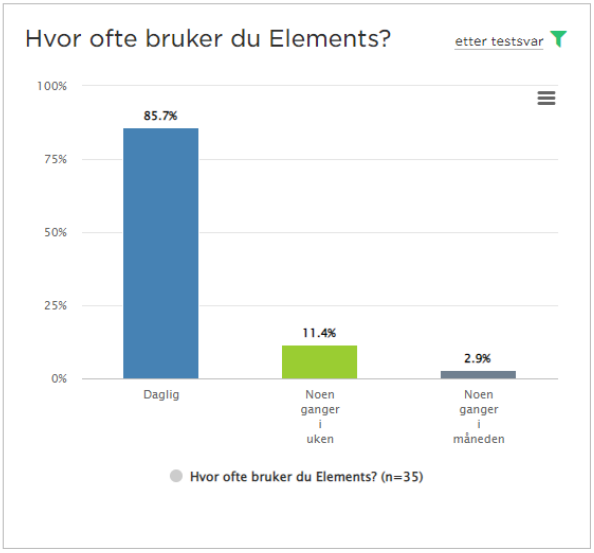
0/4000

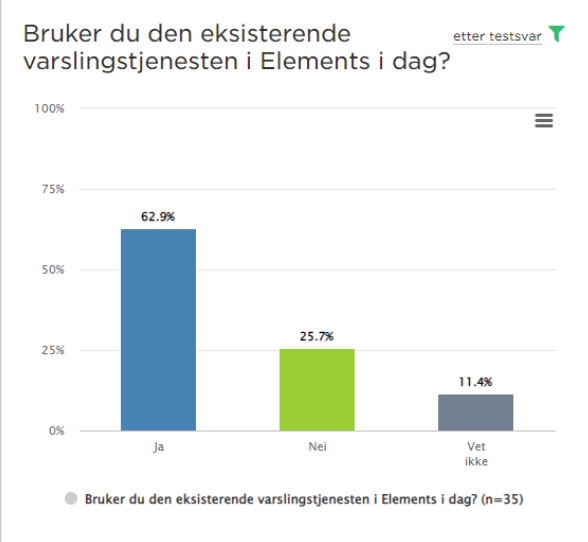
Send

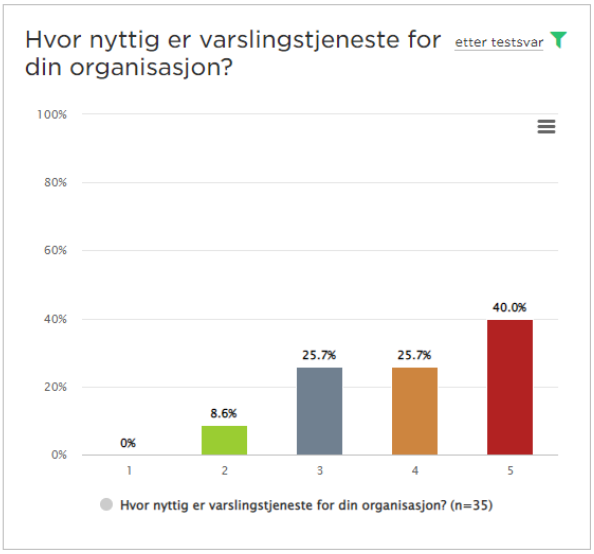
100 % fullført

Appendix 7: User Survey Result



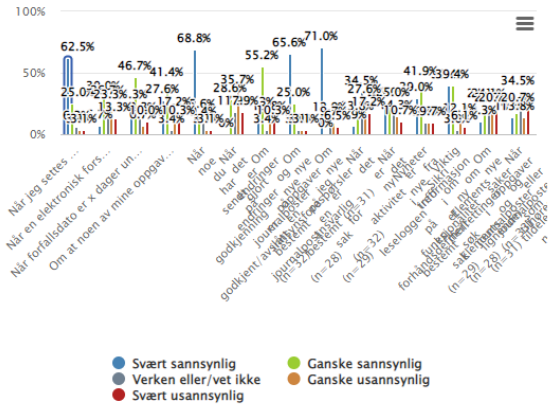






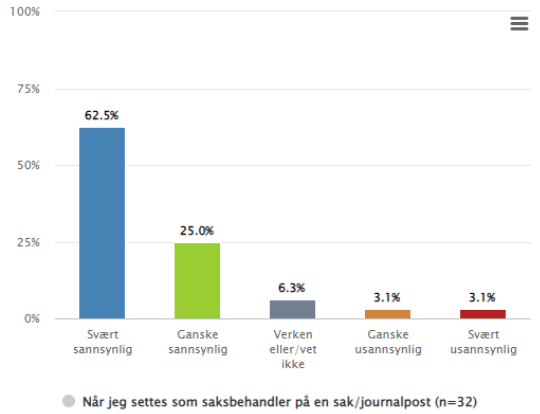
Hvor sannsynlig er det at du ville aktivert varslinger for følgende hendelser:

etter testsvar



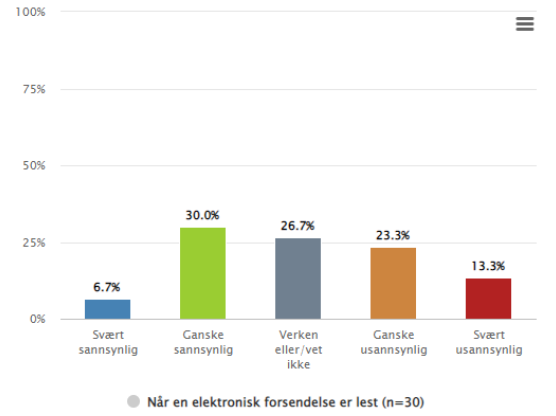
Når jeg settes som saksbehandler på en sak/journalpost

etter testsvar



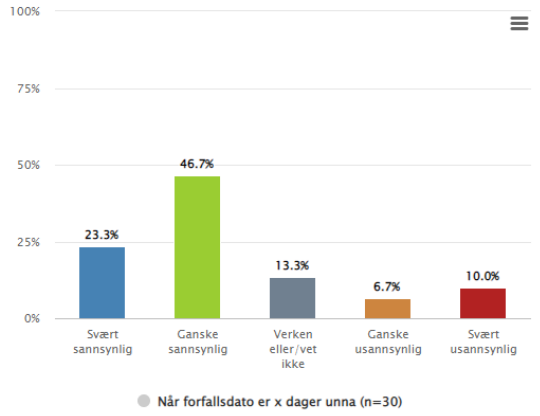
Når en elektronisk forsendelse er lest

etter testsvar



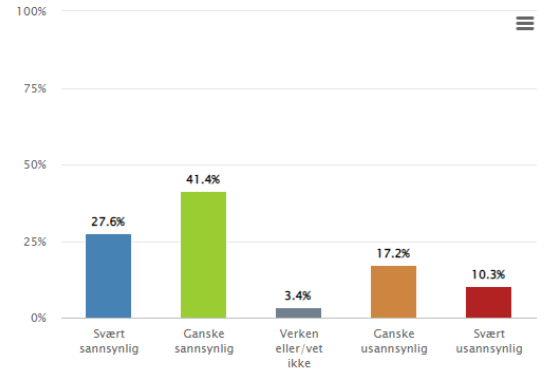
Når forfallsdato er x dager unna

etter testsvar



Om at noen av mine oppgaver eller journalposter har passert forfallsdato

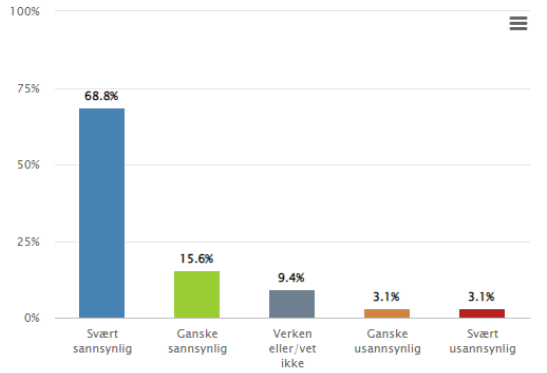
etter testsvar



Om at noen av mine oppgaver eller journalposter har passert forfallsdato (...

Når noe du har sendt på godkjenning er godkjent/avslått

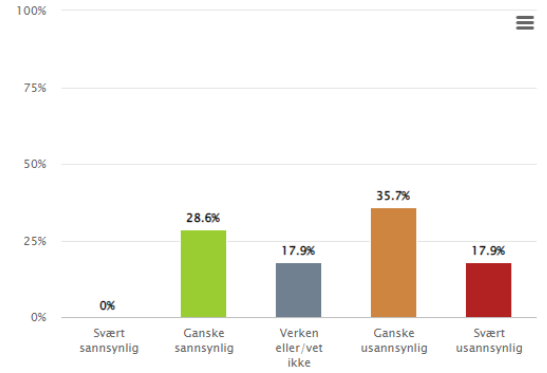
etter testsvar



Når noe du har sendt på godkjenning er godkjent/avslått (n=32)

Når det er gjort endringer på en bestemt journalpost

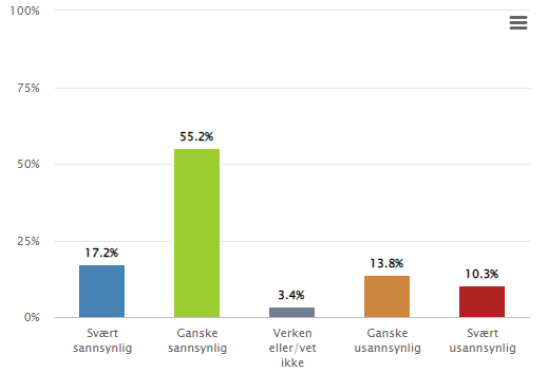
etter testsvar



Når det er gjort endringer på en bestemt journalpost (n=28)

Om endringer og nye journalposter på en bestemt sak

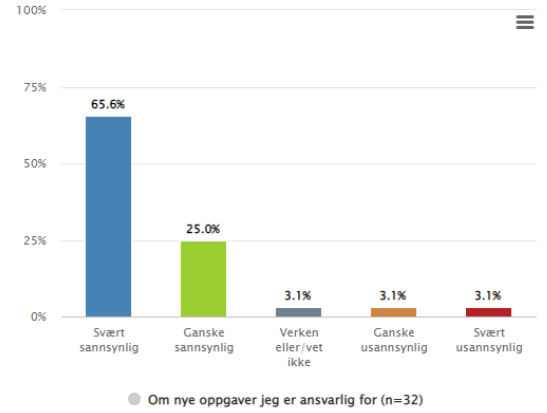
etter testsvar



Om endringer og nye journalposter på en bestemt sak (n=29)

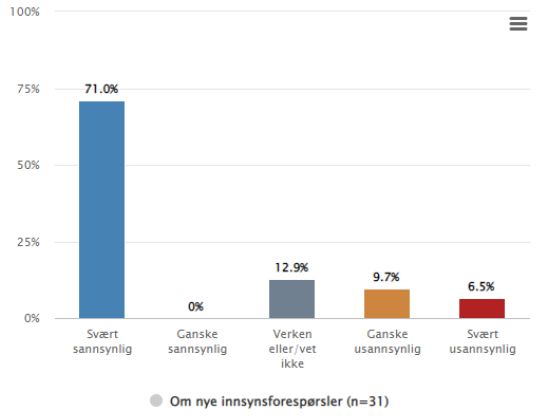
Om nye oppgaver jeg er ansvarlig for

etter testsvar



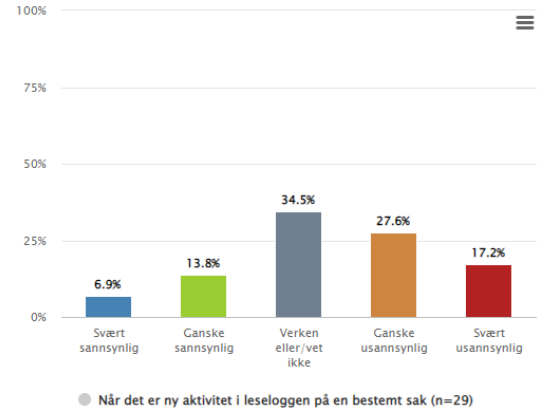
Om nye innsynsforespørsler

etter testsvar



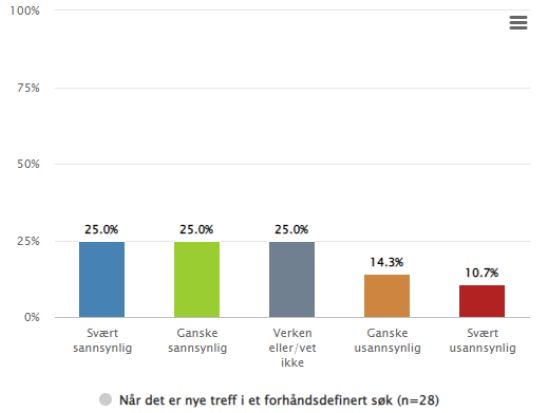
Når det er ny aktivitet i leseloggen på en bestemt sak

etter testsvar



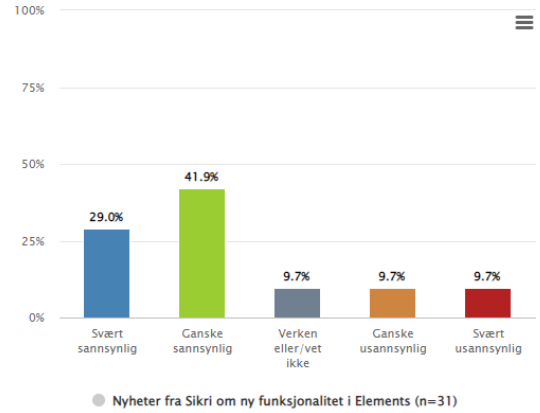
Når det er nye treff i et forhåndsdefinert søk

etter testsvar



Nyheter fra Sikri om ny funksjonalitet i Elements

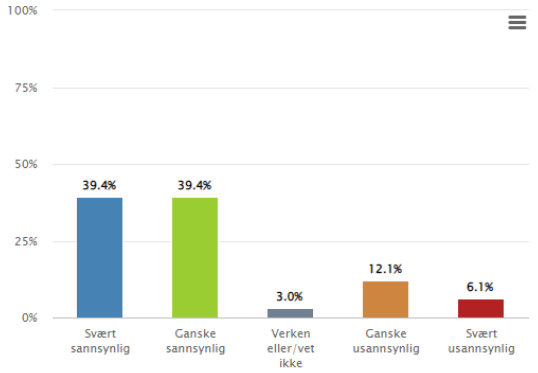
etter testsvar



Nyheter fra Sikri om ny funksjonalitet i Elements (n=31)

Viktig informasjon om Elements (feilrettinger og lignende)

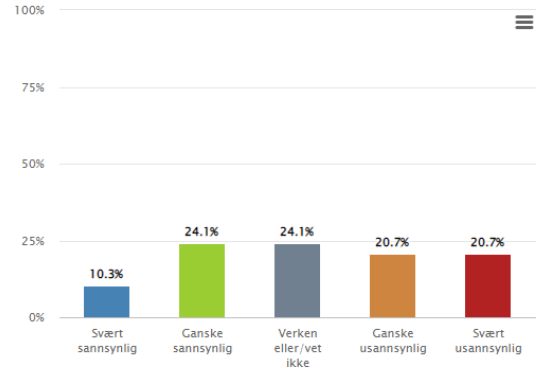
etter testsvar



Viktig informasjon om Elements (feilrettinger og lignende) (n=33)

Om nye saker og journalposter som tildeles noen i min avdeling

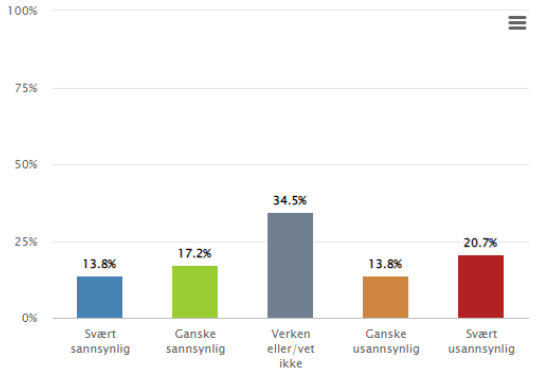
etter testsvar



Om nye saker og journalposter som tildeles noen i min avdeling (n=29)

Når oppgaver eller journalposter tilhørende noen i min avdeling har passert forfallsdato

etter testsvar



Når oppgaver eller journalposter tilhørende noen i min avdeling har passert...

Appendix 8: Definition of Done (DoD)

Definition of Done

A PBI may be accepted as done if all the following expectations are met.

PBI requirements:

- Acceptance criteria from user stories (Given/When/Then)
- Needs to be estimated
- Additional information if necessary

Requirement to create Pull Request:

- Write/run tests
- Backend – Linting
- Specify if still in progress
- Pull down main

Pull Request reviewal:

- Read and look for code smells
- Run and read (review) tests
- Manual testing when applicable

Pull Request Completion:

- Minimum two approvals
- Squash commits
    - Tag work item in commit
- Delete on completion (should be on)
- Work item completion should not be on