NTNU
Norwegian University of
Science and Technology

ACLgen: adapting legacy applications to the web

Author(s)

Sigmund Granaas Sandring
Karl Labrador
Ilona Podliashanyk

Bachelor in Software Engineering
20 ECTS
Department of Computer Science
Norwegian University of Science and Technology,

20.05.2022

Supervisor Donn Morrison

Summary of Graduate Project

Title:	**ACLgen: adapting legacy applications to the web**
Date:	20.05.2022
Authors:	Sigmund Granaas Sandring Karl Labrador Ilona Podliashanyk
Supervisor:	Donn Morrison
Employer:	Sikt - Kunnskapssektorens Tjenesteleverandør
Contact Person:	Vidar Faltinsen
Keywords:	Thesis, ACL, IDI
Pages:	48
Attachments:	4
Availability:	Open

Abstract

This bachelor thesis concerns the development of the web application ACLgen. ACLgen is a system for managing and generating access control lists for firewalls. The reason behind the project is expressed as a need for a modern take on an already existing application that is out-of-date and no longer being maintained, to improve the day-to-day workflow for a network engineer. The goal is to build the foundation for a web application that meets this need, with further development in mind.

With ACLgen, a network engineer can generate rules and create abstract objects such as hosts and services for re-usability through a web-based user interface. Changes are saved locally and are visually tracked in the user interface, so that the user may see new additions and changes before committing them to the server. The system supports storing rules along with their related objects in a database, to persist the data. It also allows for managing multiple networks by creating additional repositories and making separate sets of firewall rules.

The team takes advantage of modern technologies and development methods such as Next.js, Django REST framework, and Scrum for agile development. Using these technologies has allowed the team to build expandable core systems and a user interface with solid interaction mechanisms to prevent creating invalid rules and objects.

This report focuses on exploring the possibilities and problems related to adapting and improving a native legacy application into a web-based solution utilizing a modern software stack without compromising on functionality. Research and development methods were adapted in accordance with the environment constraints and a small user base. As a result, the team has come up with solutions and suggestions on the uncovered challenges of the legacy software adaptation.

Preface

This report is the outcome of the system development project in connection with the final bachelor thesis in the study Bachelor of Engineering in Computer Science, at the Department of Computer Science at the Norwegian University of Science and Technology in Trondheim. The project is an assignment given by Sikt (formerly known as Uninett), a service provider for the education sector headquartered in the city of Trondheim.

The students reached out to Sikt in late 2021 regarding potential system development projects for our bachelor thesis. The company invited the students to discuss their initial thoughts and vision of the project, and found the project to be interesting and match the students' field of knowledge.

Working with this project has given the team a lot of insight into the field of networking on a larger scale, and provided new experiences with technologies the team had no prior experience with from before.

We would like to thank Vidar Faltinsen for cheering us on and organizing workspaces for the team in their Trondheim office. We would also like to give our thanks to the engineers Morten Brekkevold, Vidar Stokke, Knut-Helge Vindheim and Tom Ivar Myren for actively taking an interest in our project, and providing useful feedback on user experience and networking knowledge.

Finally, we want to thank Donn Morrison, our bachelor thesis supervisor, for useful feedback and advice.

Trondheim, May 20, 2022

Sigmund Granaas Sandring

Karl Klykken Labrador

Ilona Podliashanyk

Assignment

Original assignment text (in Norwegian)

Uninett tilbyr CNaaS – Campus Network as a Service – til våre kunder. Med CNaaS designer, bygger, videreutvikler og driver Uninett campusnettet til interesserte kunder. For at dette skal skalere godt er det nødvendig å automatisere tidkrevende prosesser og drive rasjonelt. Vi er også avhengig av et godt samarbeid med kunden, der de lokalt kan gjøre fysisk montering/oppkobling, men også enklere, manuelle konfigurasjonsendringer i det lokale nettverket.

Relevant i denne sammenheng er konfigurasjon og vedlikehold av brannmurregler. Med brannmurregler forstås enten aksesslister (eller ACLs = access control lists) på rutere eller regler i en brannmur. Det skal i denne oppgaven utvikles et verktøy (ACLgen) for effektiv og brukervennlig administrasjon av brannmurregler. Det er et poeng i seg selv at verktøyet må være generisk og ikke spesifikt til et bestemt produkt (ruterfabrikat / brannnmurtype). Videre må brukergrensesnittet være intuitivt og lettfattelig, da kompetansenivået til de som skal bruke verktøyet vil variere.

Det bør innledningsvis gjøres en studie av hva som finnes av open source løsninger der ute og om det er noe man kan bygge videre på.

Et viktig krav er at ACLgen evner å abstrahere filterparametre som IP-adresser, IP prefiks, TCP/UDP porter m.m. til objekter som igjen kan brukes i bygging av regelsettene. På den måten kan brannmuradministrator bygge regler mer abstrahert og kan enkelt gjenbruke byggeklosser i ulike deler av regelsettene og også på tvers av rutere/brannmurer.

Et interessant prosjekt det kan tas utgangspunkt er Google sitt Capirca prosjekt: https://github.com/google/capirca. Capirca er en tekstbasert aksesslistegenerator som støtter de viktigste plattformene og som har denne evnen til å bygge objekter. Capirca har imidlertid ikke noe grafisk brukergrensesnitt. Dersom Capirca-sporet forfølges, skal det her ses på hvordan man kan lage et brukervennlig webbasert grensesnitt. Mange nettverksadministratorer i UH-sektoren har etterspurt et slikt grensesnitt (noen bruker fwbuilder: http://fwbuilder.sourceforge.net/index.shtml), men prosjektet har vært dødt i mange år og er ikke tilrådelig å satse på).

ACLgen må ha mekanismer for å teste konsistens i regelsettet som bygges og det må bygges inn logikk som hindrer at alvorlige brukerfeil blir rullet ut som aktive regler. Det kan også være aktuelt å automatisere prosessen utover at konfigurasjonen blir generert, dvs inkludere å pushe konfigurasjonen til ruter/brannmur. Det skal i oppgaven fokuseres på Juniper rutere og Juniper brannmurer og her skal man utnytte Junipers mekanismer for rollback på commits. En aktuell tilleggsoppgave kan være å lage et forenklet webskjema som CNaaS-kundene kan benytte for å melde inn ønsker om åpninger i brannmurreglene. Skjemaet skal være «idiotsikkert» slik at vi sikrer at kunden oppgir all nødvendig informasjon på en korrekt måte. Dette er et problem i dag der innmelding ofte skjer som fritekst epost og er upresis, noe som medfører ekstraarbeid ved at Uninett må tilbake til kunden for å avklare. Output fra dette skjemaet skal kunne automatisk/semiautomatisk tas inn i aktuelle ACLgen regelsett.

Changes

The scope of the project has changed during the process to focus on building a solid foundation for further development of the web application. This means that the team will focus on core systems that would be easy to expand in the future, as well as solid interaction mechanisms, to make the tool efficient, and prevent the user from creating invalid elements.

Contents

List of Figures

List of Tables

1 Introduction

Configuring network rules is a day-to-day task for network engineers at Sikt. This process involves the configuration and maintenance of Access Control Lists (ACLs) in the routers and firewalls. ACL can be regarded as a set of permit-deny-reject rules to and from a router or a firewall that is meant to limit unauthorized access to the network. Management of ACLs is a crucial network security procedure and requires an advanced level of knowledge about networks and access rules (router setup, IP addresses, transmission protocols, in/out interfaces, and more). ACLs are applied to the network after a filter file is deployed to the appropriate router (or switch). Filter files differ in format (both syntax and contents), depending on the hardware's vendor and model.

Sikt has years used a tool called FWbuilder [1], which provides a graphical user interface for configuring access rules and retrieving a filter file that could be pushed to the router for ACLs to be active. FWbuilder is a tool for clustering network objects and configuring firewall rules (including ACLs). FWbuilder has become an outdated system and is no longer updated. The main problems with FWbuilder, as expressed by network engineers at Sikt, are:

1. Tedious workflow: extensive amount of steps (mouse clicks) are required to perform common actions, application crashes due to user errors, application is run as a standalone local instance.
2. Confusing and little intuitive UI.
3. Application is no longer under development.

Therefore there is a need for a tool that maintains FWbuilder's functionality for configuring ACLs on the firewalls, but is also up-to-date regarding the development stack, has user-friendly UI and workflow, and is vendor-agnostic when creating filter files.

Sikt is developing a product set called CNaaS (Campus Network as a Service) [2], and the need for an updated tool for managing ACLs becomes more and more acute. The rules are often replicated across devices and locations and should ideally support a wide range of hardware.

Sikt is looking for a tool they can continue to develop in-house. This means that there is a need for software that would be easy to maintain by Sikt and is up-to-date regarding the development stack.

The purpose of ACLgen, as a replacement for FWbuilder, is to optimize user's workflow when configuring access rules for firewalls, by developing a new, up-to-date product that will assist in user's day-to-day process of managing networking rules (ACLs). The result of the project will give long-term value to the user in the form of a time-saving tool.

The main purpose of ACLgen in the long term can generally be described as sustainability. ACLgen would allow for the following benefits when managing ACLs:

- save resources otherwise used on manual administration of access-control lists.
- reduce the rate of errors in network configuration.
- be operable by less experienced administrators with a more intuitive UI.

In the process of developing ACLgen, the team has discovered several interesting issues and system development challenges. This report provides insight on how to replace legacy software with a web-based cloud system while maintaining relevant functionality and providing comfortable and modern UI and workflow.

1.1 Problem Statement

Transferring old solutions to new platforms can increase the lifespan of otherwise dead solutions. This process does, however, come with some downsides. The legacy application might use features that are hard to implement, or outright impossible. Features like seamless drag and drop and working with an offline system, might be a challenge on the web. Transitioning from a working tool to another one is a difficult process, as the new product isn't a viable alternative until it has achieved at least one of two things: feature parity, or a significantly better user experience. Although some features might be harder to implement on the web, it does have a lot of advantages. The tool-chain for web-based applications has matured significantly, making development and maintenance significantly easier than before. Accessibility is also expected to improve significantly as the project is taking a web-based approach.

Two of the primary methods for interacting with the legacy software is by leveraging context-aware right-clicking and using draggable element in the application. These actions are possible to implement in the browser but are not supported as native methods of interaction for browsers. Exploring ways of enhancing user interactivity in the browser, is key to matching the features of the old application, but might also present challenges due to different browser implementations of certain features. Straying away from using native features in browsers might bring stability issues, but is also a necessity for maintaining existing features users has grown accustomed to.

The following problem statement will be the focus of this project:

"*Exploring the possibilities and problems related to adapting and improving a native legacy application into a web-based solution utilizing a modern software stack without compromising on functionality.*"

2 Theory

2.1 Keywords and Concepts

This section will give a short description of keywords and concepts.

2.1.1 Scrum

Scrum is an iterative agile development strategy, using time boxed development cycles[3]. After every iteration, the team should present the potentially shippable product the the stakeholders for review. By using scrum, the team is able to focus on some features, and to try bring them to completion by the end of the sprint. By focusing on new unique features every sprint, the team can slowly but surely aim towards feature parity with the legacy software, one iteration at the time.

2.1.2 Last responsible moment

Postponing big decisions until the last possible moment. Doing this will allow for more time to gather information. If the team is uncertain in regards to a choice, a plan can be made to organize the information that is needed to be gathered before making a decision.

2.1.3 User Stories

A user story is not an index card that described what must be implemented in the software [4]. The point of a user story is to tell a story. Stories are easier to handle than just referring to a requirement. It should be more than just simple text and post-it notes. The purpose of a user story is to create a story that described a scenario for the software developers and product owners. This approach ensures that both parties understand the scenario of how an end-user will be using the feature from the user's point of view. Creating user stories will resolve any misunderstandings between the software developers and product owners.

2.1.4 Shared understanding

Shared understanding is often used with user stories to describe scenarios where the software developers and product owners have the same vision and understanding of why users need software. A common understanding between the two parties is necessary to be built. A simple requirement for the software is not enough for both parties to understand why the property of the software exists. As such, user stories are used to gain a shared understanding.

2.1.5 User story maps

There will always be more ideas than available time and resources. The project members may want to create a visual overview of all the possibilities and feature requests for the software. Creating a map of all potential user stories will allow for putting together stories and organizing a plan with details on how properties are related to one another, and in what order to work on an issue. Putting together user stories will make it easier to see relationships between stories, and potentially spot missing prerequisites to produce a user story with software.

2.1.6 MVP

An MVP is the minimum viable product that meets a specific goal. The definition of an MVP is quite subjective. For an MVP to be useful, the requirements must be concrete. Developers may not have the available time or resources to validate all implementations. Many features have no concrete solution, and it is up to the software developers to try possible solutions. Assumptions that bear a high risk can be mapped out. By using MVP's, the team can confirm or reject any arised assumptions. This is a repeatable process until the team is certain that the available time and resources are spent on developing useful software. This will also mean that the team does not spend unnecessary hours on building unnecessary software.

2.1.7 Prototyping to learn

Prototypes are created with clear milestones to learn from them. The work with prototypes ends as soon as relevant knowledge and insight is uncovered. The point of prototypes is to learn more about the problem that needs to be solved. This can involve making prototypes of user interfaces to learn more about what users prefer of appearance in the software. Prototypes like these can give insight and feedback from the users.

2.1.8 Build to learn

When starting to build software, the goal does not necessarily have to be delivering complete software. Most importantly is building enough software that users may find useful. The software does not necessarily have to bear great quality, but enough to be able to collect feedback on its usefulness from users that will actually benefit from the software.

2.1.9 Validated learning

The software that is being built is based on assumptions on what the user needs, and the assumption that the software actually solves the user's problem. Validating an assumption is not as easy as performing a survey against the target user base. The development is based on the users also having an assumption on what they need. The team may not exactly know the user's needs before the team provides working software that the user can use. This approach fits well into the Scrum process, where software can be delivered after each sprint, to confirm if the software is useful or not for the users. The team can take notes of existing knowledge, and what assumptions the solutions are built on.

2.1.10 Definition of done

For efficient collaboration and planning, it is important that all the members of the team have a common understanding of when a task is complete. The definition of complete in a piece of software may vary, depending on the goals for the project and the individuals that work on it. All the members must agree on what a completed task means, which is important in order to plan and organize the tasks for the team. This means that the team needs to add checklists to break down tasks and components, to easily identify completed tasks. Items in the checklist may include code quality, complete integration in the software or unit tests. For all the tasks, the team must agree on the definition of completed task.

2.1.11 Velocity

Velocity is the term for how much progress the team can do for every sprint. This can be measured using points, or with an hour value to define how much is involved in a task. This

approach is used when planning sprints, to see what user stories the team can take on in a sprint period based on capacity.

2.2 Technology

2.2.1 REST

REST is an acronym for Representational State Transfer, which is an architectural principle and style for managing state information. The architectural style is mainly used in web applications and was introduced by Roy Fielding in his doctoral dissertation in 2000 [5].

The architectural style allows for independent development and implementation of the client and the server. This means that a team can make changes to either the client or the server without necessarily affecting the operation of its counterpart. As the client and the server are separated, they may be developed and evolve independently. [6].

REST and its architectural principles come with constraints. Key constraints will be mentioned, to describe how the architectural style is put into practice.

As mentioned earlier, the client and the server are independent, and as such their concerns are separated. By doing this, the team can improve the portability of the client and user interfaces across multiple platforms. Separation allows the two components to evolve and scale independently [5].

Communication from the client to the server must be stateless. This means that every request the client sends to the server, must contain all the necessary information for the server to understand the request [5]. As communication is stateless, every request is new and independent from each other, and the state is handled by the client instead.

Clients send requests to retrieve or modify resources to the server. The server will then reply with responses to the requests. Resources are identified using Uniform Resource Identifiers (URIs). A client request generally consists of an HTTP verb, a header, a path to a resource, and optionally, a message body that contains data if the client wishes to send any to the server [6].

In a request, the client can specify what operation it wants to do by using HTTP verbs. The four basic and common verbs that can be seen in a REST-style API are GET, POST, PUT and DELETE. GET is used to retrieve a resource, POST is used to create a resource, PUT is used to update a resource, and DELETE is used to destroy a resource. The HTTP verb PATCH may also be used to for example update only a specific part of the resource.

For this project, the team is taking advantage of the REST architectural principles for developing the server backend for our application. The purpose of the server in the project use case is to store information related to the data objects the client application will be working with.

2.2.2 Client-side frameworks

Client-side frameworks are toolkits for building scalable and interactive web applications and are an essential part of building modern frontends when developing a web application [7].

As mentioned in the REST section, the client and the server are separate concerns, and those two components may be developed independently. Client-side frameworks allow for further separation of concerns by breaking down the client-side web application pieces into components, using the tools provided in the framework.

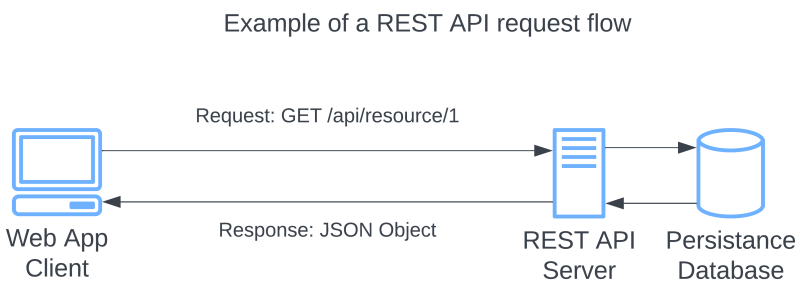


Figure 1: Example of a REST API request/response flow. The client sends a GET request to the server, and returns a JSON object in the response.

For this project, the team are taking advantage of modern client-side JavaScript frameworks that will allow for structured development of the client-side of the web application.

2.3 Development Process

2.3.1 Wireframes for UI mockups and User Testing

A wireframe assists designers and software developers to communicate the structure of a web application that is being built. It serves as a schematic and blueprint for the surface appearance and design of a frontend web application. Using tools for wireframing, a design can be tested against user experience engineers and the target userbase, and be revised further based on their feedback before writing any code. Drafting a wireframe before writing any code will save the frontend developers time, and will be useful to avoid spending unnecessary engineering resources on later adjustments [8]. Making adjustments on a wireframe is easier and more time-saving than doing adjustments with code. A wireframe is always up for discussion and does not necessarily represent a final design. Structure details weigh more than coloring the interface elements, for example.

For this project, the team is using wireframing tools to create prototypes of a user interface for the web application's front end. This approach will communicate the expectations on a user experience level based on feedback from the end-users.

2.3.2 Version Control

Version Control Systems are systems that are responsible for tracking and managing changes to software code. These systems track every modification made to the code base in the form of commits to the central code repository of a project. Applying this practice will help software developers manage code changes that happen over time. With version control, multiple software developers can work on the same code base [9]. Developers can download a local copy of the code base, and make changes in form of commits locally and independently of the main codebase. Code commits can later be pushed to the main code repository.

With multiple developers working on the same code base, conflicts in code changes may occur. Version Control Systems solve this by providing tools to safely merge code changes and solve conflicting changes, before being pushed to the main codebase. As all changes are tracked, it also allows for rolling back to an older version of the codebase. The branching feature also allows for developers to push code to the main repository, but on its own version

of the codebase [9].

For this project, the team is using a distributed version control system to handle and track code changes of the client and the server of the web application in separate code base repositories. By actively using version control with branches, multiple team members can contribute to both application components by making changes to their local codebase.

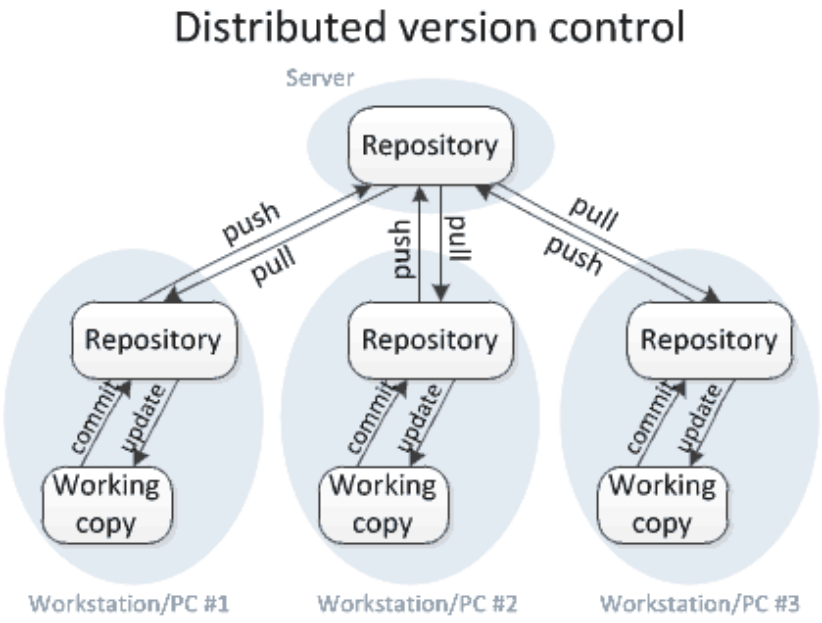


Figure 2: Workflow of a distributed version control system. Developers clone a repository locally, and make commits and updates to their local repository. Changes are later pushed or pulled from the central repository [10]

3 Choice of Technology and Method

3.1 Research methods

3.1.1 Background for the choice of research methods

Choice of research methods has a considerable impact on the answer to the problem statements. When selecting research methods, there is a number of considerations to make in order to ensure that the results are credible, valid, and reliable. Problem statement of this project is tied to software development. Therefore the team has adopted Design Science Research (DSR) framework as a basis for research. DSR [11] is a framework that is meant to reduce the gap between research methods and software development methods. This is achieved by considering software development as a step in a broader research method. This way development becomes a link in producing knowledge about the technology that was applied, which reflects the conditions of this project's problem statement.

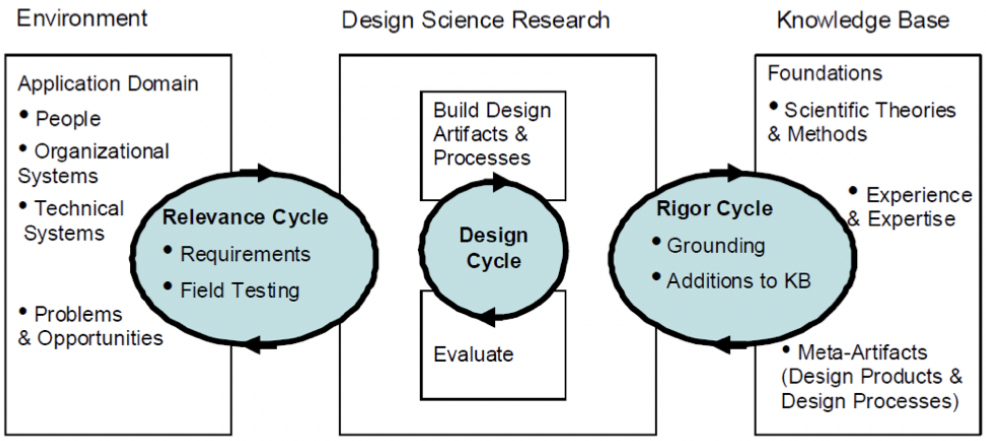


Figure 3: Cycles of the Design Science Research framework ([11], Fig.1)

When developing ACLgen, the team has focused on two cycles of DSR - Relevance and Design, as both are important given the development constraints and the environment. Iterating over the Relevance cycle is done in order for the application to be correct and relevant, based on the domain. Iterating over the Design cycle ensures that technology stack of the application is solid and effective.

There are several development conditions that are important to consider in the scope of this project:

1. ACLgen is a replacement of FWbuilder. This introduces the need to study the existing software, both the functionality and how it is used.
2. ACLgen is not meant to be finished in the scope of this project, and will further be developed by the client.

3. ACLgen has a very narrow and specific user base. Application is expected to be used internally by the client. User base consists of 2 network engineers. The client has expressed the intention of deploying ACLgen as a product for the external use, but it is a long term goal. This condition introduces challenges with user tests.

3.2 Project phases

The project is split in two primary phases: research and development. The estimated available work amount for the whole project is 500hrs per person. Splitting the project into distinct phases makes it possible for the team members to adapt their work method and focus, to better suit the time and resources available, to help the project reach its goals.

3.2.1 Research phase

The goal for the research phase was to gather sufficient information about how the old solution was used, and increase our knowledge about the problem domain to start the development of an application to manage access control lists. There were a lot of questions regarding the functionality of the application, which could not be tested in isolation. This meant that some core components of the system had to be built before the viability of different solutions could be tested in conjunction with each other. The results for this phase were not intended to create a complete picture of how the application would work when finished, but rather a solid foundation where the team could quickly start development and user testing.

Results from discussing with users and shareholders from Sikt

Throughout the research phase, several meetings were conducted to discuss potential solutions, discuss design decisions, and help deepen the team's understanding of the problem domain, and understand the users' needs in the new application.

The primary userof the applcation are experienced engineers over at Sikt and less experienced user at the help center. All users have experience with central networking concepts. While discussing the targeted users, it became clear that the application would initially only be used by a handful of people. Creating a distributed application with users outside of Sikt could be an option in the future, but this project was only intended for internal use.

3.2.2 Results from UI mock

Creating a visual representation of the application will make the discussions about required features easier to understand for both parties. Creating shared understanding with the customer can be difficult, and the team spent a lot of effort visualizing their ideas to make it easier to discuss and present them to the stakeholders. The stakeholder were generally pleased with with the look of the application, but had concerns regarding the layout of the network devices and and objects, as well as technical details regarding the functionality of the application, which were outside the scope of the meeting, but useful feedback.

3.2.3 Results from prototypes

To form a decision for which technologies would help us develop the the application, the team decided perform some tests and create prototypes for central parts of the software stack. These tests would reveal how certain frameworks functioned premade scenarios. The team decided to do practical tests of the software instead of basing their decisions on documentation and specifications for relevant frameworks. Doing practical tests would enable to

team to explore sample scenarios the application would be designed to solve, and leverage these prototypes to learn something new about the problem domain.

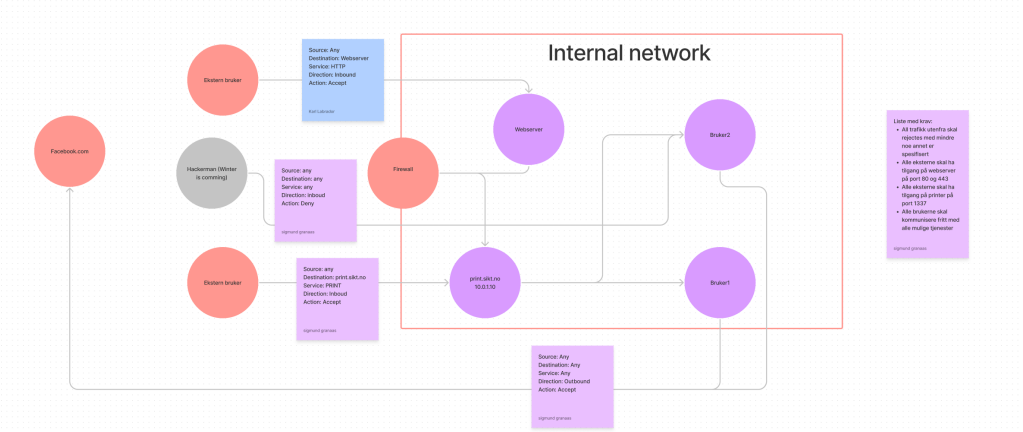


Figure 4: An example scenario used to create early prototypes before development started

The goal of the application is to handle and manage large amounts of data. Handling this data in web applications requires the use of state management tools, like context or Redux. A simple test setting up some of the data intended to be handled by the application quickly revealed that using the context API is extremely quick and easy to set up, but would require a lot of effort to create a system for handling complex state scenarios.

Several different UI frameworks were discussed, like Bootstrap and Material UI, but the team realized they would not help the team create new ways of managing user inputs. Tailwind was then chosen because it gives the developer lower lever control over components. This would allow the team to develop components from scratch, and create components that behaved predictably, even though the method of input was different.

The result from the prototyping stages was that the team could start planning the first sprint with features in mind, instead of getting to know the framework which was chosen. The team had already built components using these frameworks and could use them to jump-start the development of actual features for the sprint. This approach made it possible for the team to have a working prototype of the application within a few days of starting development.

3.2.4 Testing existing legacy solution

To get a better understanding of the needs of the end-user, the team was given access to an instance of an existing instance of FWBuilder. This allowed the team to explore the software currently in use to perform access control list management, which allowed the team to gain a basic understanding of the end-user workflow scenario.

3.2.5 User story maps

To be able to create an overview of how the functionality of the application would be composed of several different user stories, the team created a User Story Map[4]. The goal of the User Story Map is the create an overview of how user stories are connected, and how these features can be bundled together to make it possible for users to complete some scenarios. By creating a visual map of which user stories work in conjunction with each other. This makes it

easier to start planning sprints, as groups of functionality can be targeted for a single sprint.

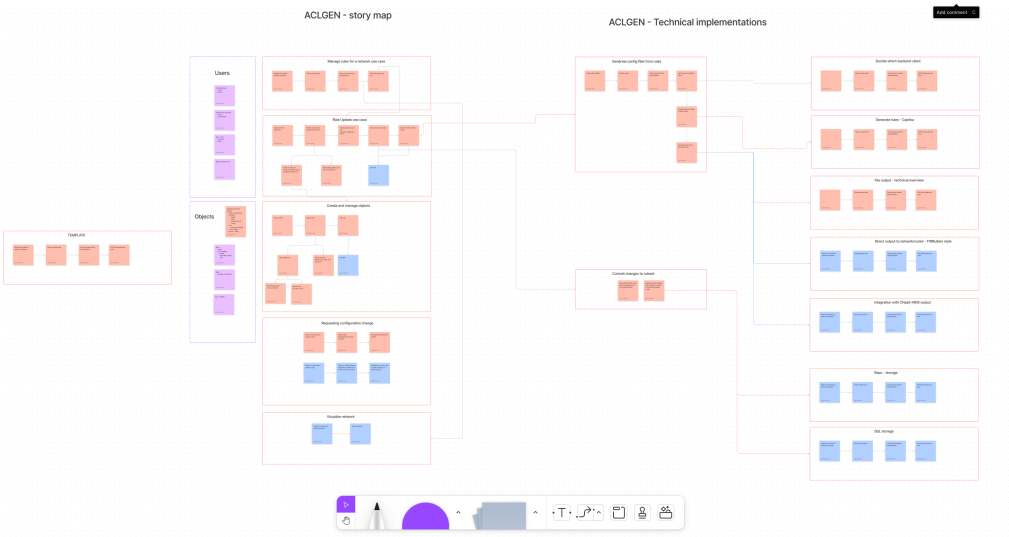


Figure 5: User story map covering most of the functionality discussed in meetings with the stakeholders.

3.2.6 Development

The development phase is dedicated to developing the solution. During this period, the team will be able to work full weeks. Through the entire period of 21. March to 20. of May. Having a period like this available allows the team to adopt more efficient development strategies, like Scrum. To capitalize on the available time, the team will complete four sprints, of two weeks each.

3.2.7 Agile methodology

Scrum was chosen as the development method of choice. Scrum focuses on time boxed iteration where feature complete software should be delivered. Other methodologies could have been chosen as development strategies, most notably Extreme Programming(XP). One of the main differences between scrum and XP, is how XP focuses on continuous refactoring and delivery, as well prioritizing simple solutions. XP works really well for maintaining a stable solution, by focusing on continuous integration, and trying to refactor existing code to make it more flexible and more stable. If ACLGen was a mature project already being used in production, this approach would make more sense. ACLGen is a new development project, and the most important factor is to test whether features can be implemented, and how well they work. This means that a lot of the work with creating testing routines, a build and release pipeline and the necessary frameworks for this would be largely wasted in early parts of this project.

Kanban's strength lies in management. By increasing the visibility of work by putting tasks on a kanban board, you can easily see where resources are being spent, and where there are potential bottlenecks. This allows teams to prioritize work by how much it is needed by the rest of the team. This approach also ensures that no member or team takes on unreasonable workloads, as there are strict rules for how much work is allowed to be scheduled. This approach works better in an environment where there are several tasks which depend on

each other, and making sure there are no bottlenecks in the system. As the development team is small, spending too much time managing team members would be counter productive to the overall objective of the project.

3.2.8 Sprint planning

During the start of every sprint, the team members along with the stakeholders, will create a plan for which user stories should be completed by the end of the this sprint. By continually measuring how many user stories the team were able to complete every sprint, the team can get a better picture of how much time certain tasks takes to complete, as well as how much capacity the team has for delivering software every sprint.

3.2.9 Daily stand-ups

Scrum is driven by self-organized teams. This means that the team members should be able to plan and execute the agreed-upon goals for the sprints by themselves. To achieve this level of self-organization by the team, they will utilize the daily stand-up as a planning tool within the team. The team uses this information to plan out the day's tasks, and orient themselves about what the other members are doing. Most of the tasks planned in the sprint will be interconnected, which means that daily planning is needed to utilize the time available to accomplish the goals of the sprint. Daily updates also give a lot of insight into the well the stories have been measured.

3.2.10 Sprint review

By the end of each sprint, there is a sprint review. At the review, the agreed-upon stories are presented to the stakeholders, and the team can test the features that have been under development during the sprint. The review would include the stakeholders from Sikt as well as the development team. As the users for the software were present at this meeting, the team could use this meeting to conduct the user tests for the features in development. During this session, the team would discuss how well the features performed, and which features needs further development next week, as well as which features needed changes or more work before they were finished. The sprint review is the team's opportunity to test software with its intended users. This feedback would form the base of any decision regarding how features should be developed.

3.2.11 Sprint retrospective

After every sprint, the team would discuss how they performed. This meeting is important for highlighting what went well, and what went wrong during every sprint. By discussing how the team can improve every sprint, potential recurring problems can be solved as soon as they are discovered. In contrast, elements that contribute positively during the sprint can be highlighted, to make sure the team is not missing out on any potential for improving how the team works.

3.2.12 Using scrum as a method for gathering information

Developing software gives the developer an overwhelming amount of possibilities for how to implement the functionality. A well-planned feature increases the chances of success but guarantees nothing. Placing features in the hands of the users, allows the developers to test how an idea works in practice. An untested idea or feature carries a huge risk, as there is no way to know if the feature works as intended before it is used in production. Increasing the

frequency of testing allows developers to gather more information about the features they are developing. Although testing software frequently allows developers to gather more information, there is also a point of diminishing returns. Decisions have to be made concerning every single part of the application, but not every feature is created equal. Some aspects of the application, like ordering of text elements, font size, and colors are easy to change and can be tested with a higher frequency, although the results of the tests, might be less significant. Some features, like the Drag and Drop functionality, are not easily changeable and require significant development time. Frequent tests on big features are unlikely to yield any interesting information and might hinder development if the team is doing development and testing concurrently.

Balancing the frequency of tests with how the team develops an application, creates an opportunity for the developers to plan their development around which assumptions can be tested, as well as how vital this information would be for the project. A planned feature for an application is nothing more than an assumption of how it will be used in the real world if indeed, it is used. The time available for development is limited. Spending time developing a feature has the potential to be wasted time until it has proved its potential for being useful. The available time for development is extremely limited for this project. This means that the team will have to prioritize which features are the most important for the success of the application, and follow a development strategy for developing and testing features in order of importance to the project, and what the individual features depend on.

Using Scrum as the base for our development strategy allows us to use each sprint to formulate a hypothesis, for what assumptions the project depends on, and what the team should develop to either prove or disprove the assumptions. The user stories chosen for each sprint would be a reflection of what information the team wants to gather to continue iterating over the product. Using this strategy, many of the decisions regarding the project can be made at the last responsible moment. This means that the team will defer making a finalized decision about how features should be implemented, until enough information has been gathered, and their dependencies have been worked out. This allows the team to be flexible when discussing and testing the product with the stakeholders, because features can be planned in the context of how the project is doing, as opposed to how the project was envisioned in the beginning. This method also allows the team to make changes to how features are implemented, as the team are planning to receive feedback during the features' development. Flexibility during development is important. Envisioning a product is important to form a shared understanding between the shareholder and the developers, and to create a base on which to start development.

The vision for the project is simply a hypothesis, waiting to be either proved or disproved during development. Managing the development based on two-week time-boxed iterations, allows the developers to test the vision at the end of every sprint, and discuss with the shareholder what needs to change for the project to reach its intended goals. Implementing new features and making changes to the original vision, is a vital part of the development process, as it means the team is making progress in forming a shared understanding of the problem they are trying to solve, as well as taking steps to make sure the project is heading in the right direction based on the information the team can gather during the development.

3.3 Technical decisions

3.4 Web development methods

The basis for all methods used for developing the application, was made during the research phase. This phase gave us the results needed to make decisions about which frameworks and solutions would be most fit for development.

3.4.1 Making it easier to create correct Access Control Lists

FWBuilder is a desktop application with a local database. This is a major issue preventing more than one user from using the software at a time. The current solution is to virtualize a desktop and connect to it when connected to the right VPN. This works, but is prone to crashing, it is hard to access, and will never allow more than one user at a time. Developing a desktop application could make a lot of features easier to implement, like leveraging native drag and drop support from the OS, and being able to build a monolithic application instead of splitting it into server and client. This would probably increase developing efficiency, but could face the same issue as FWBuilder concerning the accessibility of the application. In addition to this, the application would have to work cross-platform, which could erase some of the benefits gained from relying on native API's like drag and drop. Creating a web-based application would solve the accessibility issue, and would work seamlessly across different platforms. Expanding the scope of the web application to include user authentication and access control would also be possible. The main problem with developing a web application is having to rely on different libraries and custom technologies for implementing several core features. These libraries, like state-management, drag and drop functionality and virtualization of elements work well independently, but might create issues when composing an application based on these technologies. There are no standards for how these elements are supposed to work, and no guarantees for if they work together at all before it has been tested.

3.4.2 Developing two different ways of interacting with the application

During early meetings and planning with the stakeholders and users, it was made clear that some functionality would be essential for making practical use of the software. FWBuilder users were already accustomed to using drag and drop functionality for composing and editing rules. In addition to this, some users preferred using their keyboards for interacting with software if possible. To address these issues, the development team decided to focus on creating components that could be interacted with by using a mouse and keyboard. Managing rules and objects was the main goal of the application and this is where development was focused.

To make interacting with rules easier with a keyboard, all inputs would be created as searchable components. This would allow the user to search through available elements when focusing on inputs, and manage the results with the keyboard as well. Tabbing through inputs would allow users to navigate through the rules. If an element did not exist, the user would be prompted to create the element when searching for it.

Interacting with the application by dragging elements where they are wanted is very intuitive, as this is how humans are used to interacting with elements in the real world. This method might not be the most efficient approach but is very important as this is the primary method of interaction with the legacy application, and would be the way the intended users

would be familiar. Most elements in the application that could be created and edited should also be draggable to their desired location. Dragging a service into the service input of a rule should apply this service to the rule-set of the rule. Reordering elements by dragging should also be possible.

3.4.3 NextJs as application framework

When choosing a front-end development framework, the team had two major concerns, development efficiency, and future development for Sikt. Vue and React-based frameworks were considered as the team had experience using both, and utilizing a virtual DOM, gives the developers a way to write declarative components with similar functionality. The decision to choose React was done because some engineers at Sikt already have some experience with React. This would make it easier for Sikt to continue development after the project is handed over.

React is only a library for rendering components using a virtual DOM. It provides no assistance or preference for other functions like navigation, theming, request handling, and so on. NextJs is an opinionated react framework including most features for front-end application development. The biggest feature provided by Next is the support for Server Side Rendering. Rendering the application on the server would allow the application to fetch all the data sets from the back-end before the application is loaded on the client. Because the client application is not relying on client authentication before the information is made available, all of this data could be made available when the user requests the website instead of fetching this data after the application has been transferred to the client's browser. NextJs includes components and solutions for dealing with issues like project layout, pages, and images, but these features were not considered essential and played no role in the decision to use this framework.

3.4.4 Using Redux for managing and sorting state

The goal of the project is to make it easy to create and manage rules and the elements required to compose effective rules. As the lists of rules grows, so does the data needed to manage it all. To make the system accessible and easy to use, rules for how elements can interact with each other needs to be enforced by the system to make it easier for the users to create a valid configuration. Managing and enforcing these rules can be a complex task, and requires a robust system for handling it efficiently. Using Redux makes it possible to handle state operations and logic independently of the UI components. This makes it possible to define logic for how certain data should be modified, as well as create actions which define different ways of interacting with the state. By decoupling components and global state, it becomes a lot easier to reuse application logic in different components, as well as restricting modifications to state to predefined actions, making it impossible to perform dirty changes to state.

3.4.5 Using Tailwind to create custom components

Tailwind was chosen based on the need for creating custom input components. Tailwind is a lower-level abstraction for building UI's with CSS. Tailwind does not provide any premade components but gives the developers a rich set of tools for creating their own components. In addition to this, Tailwind can be included only as a dev-dependency, meaning it will purge all unused CSS when the application is being built, leading to a reduced bundle size. Creating

custom input components would be necessary for creating unified input methods. Text inputs and object inputs should look and feel the same. This is not possible to do with frameworks like bootstrap and Material-UI without interacting directly with custom CSS and creating overrides. This would involve spending significant effort on recreating input components that would work and look like the native components from the respective frameworks. As custom components would be a necessity, Tailwind was chosen because it specializes in doing exactly this.

3.4.6 DnD-Kit for drag and drop functionality

The browser has native support for dragging and dropping elements making it possible to upload files by dragging them into the browser window. This API is provided by the HTML5 spec and works well for some applications, but is also a controversial API, as it is based on Microsoft's drag and drop implementation for IE in 1999[12][13]. Libraries are required to utilize this functionality unless you have the resources available to develop a custom solution yourself. React-DnD was originally tested when building prototypes, but had lacking support for touch input, limited options for displaying dragged elements, and was difficult to extend to multiple different dragging operations, like reordering, dragging into fields, and collapsing elements all at the same time. The switch was made to DnD-kit[14], which does not utilize the HTML5 drag and drop API. This means the application cannot drag elements from the desktop, but this is not an intended feature of the application. DnD-kit makes it possible to define several zones, and targets, which support dragging and dropping elements into rules and reordering rules at the same time. This library gives greater control over how elements are rendered when they are dragged. This makes it possible to indicate to the user which drags are allowed, and which are impossible.

3.4.7 Using Typescript to reduce the risk for run-time errors

JavaScript uses dynamic types. This has drawbacks and benefits. Dynamic types make it easy to get started with development and make code less verbose, as there is no need to specify type information. The downside is that you cannot use tools to help you check the system for basic type errors, as well as lacking documentation for how objects are supposed to look. This is a common issue in the web development ecosystem. React has a built-in way of managing types for components, prop-types [15], but this solution only supports React components, and would make it necessary to adopt yet another library for type check other parts of the application. Typescript is a popular super-set of JavaScript designed to solve this problem. Typescript allows developers to write type-safe code, which can be checked by the compiler whenever desired. Setting up types is a time investment, which makes code less error-prone by warning the developers when they write invalid code. Using typescript also makes it easier to refactor code, as the compiler can check which attributes are missing in affected components making it easier to track down affected components. Typescript transpiles its code into native JavaScript making it compatible with everything that supports JavaScript.

3.4.8 Virtualizing lists for stable performance

Access control lists could be hundreds of lines long. Rendering large lists of data on modern computers is possible, but the experience will deteriorate eventually. Early tests using static data revealed that rendering hundreds of elements with static data was unproblematic, but the situation was completely different when adding support for dynamic input fields support-

ing drag and drop as well as searchable components. The rule elements would need to act as a drop target for at least four different elements. This means that every single rule would have child components fire events for every single input when trying to drag items. Rendering and reordering large lists of elements with dynamic inputs would overload the application quickly. This would make the application impossible to use for larger networks. To solve this problem the team implemented virtual lists. Virtualized lists only render lists that are visible in the window, this makes it possible to only render a subset of the available data regardless of the list size. Virtualized lists are not a native part of React or most other frameworks and require the use of open-source libraries. Many popular libraries are deprecated and no longer maintained by their creators. The decision was made to use React Virtuoso because it is possible to integrate this library with sortable drag and drop components from DnD-kit, although they are not designed to work together. This solution makes it possible to create almost infinitely long lists that can be reordered while maintaining solid performance. Only the drop target that is visible and accessible to the user would be rendered.

3.5 Backend

The application required a back-end solution to persist data related to Rules, Services, Objects and other necessary data related to a firewall, such as Devices. To support multiple networks, all the data is tied to a parent Repository object. The backend system is required to validate incoming data, and expose the data through an API service. The team found it best to develop a backend based on REST principles, allowing a client to consume an API to interact with resource objects.

3.5.1 Django with Django REST framework

To create an API for the frontend to consume, it was important to choose a framework that could provide tools to develop a REST API.

Django is a high-level Python-based web framework that allows for a robust and quickly developed backend driven by a database. It encourages rapid development and clean, pragmatic design [16]. The framework offers tools such as Object Relational Mapping with data models, which is one of the key features for data persistence. It also has a large community and ecosystem that developers can take advantage of. The framework handles all database-related operations, which allows for changing the underlying database technology. For local testing and development, an SQLite database was used, while using PostgreSQL on our public testing instance.

To easily develop a consumable API, the team took advantage of the Django REST framework. The framework is a toolkit built on top of the Django web framework and reduces the amount of code necessary to write the REST API [17].

A few important key factors were included in the decision-making, before ultimately choosing Django. As our application also revolved around the Capirca library to generate access control lists, which is a Python-based library, it was natural to develop the backend using Python as the programming language. Going forward with Python would give the developers the possibility to take advantage of the Capirca library without any workarounds.

A potential workaround would be to develop a backend using JavaScript, which the team collectively has experience with from previous projects. However, to use the Capirca tool, it would be necessary to run access control list generation through the tool command-line

instead, with operations executed by the JavaScript code.

Other Python-based frameworks such as Flask were also considered. Flask compared to Django seemed to have a gradual learning curve, which may be due to the minimalistic nature of Flask. However, the team took into consideration that the developers at Sikt, are mainly familiar with the Django framework. This will strengthen further development in the long term, after the bachelor project has ended.

3.5.2 Pytest for testing

For unit testing, Pytest was chosen to test API endpoints and other backend components. Pytest is a testing framework that makes it easy to write small and readable tests for applications [18].

Django has its toolkit for writing tests, which is included in the Django web framework. Pytest however has features such as increased speed as it takes advantage of running a test suite in multiple processes and managing dependencies with data fixtures. Another key feature is less boilerplate, which makes tests easier to write and read [19].

As part of a Continuous Integration effort, tests are being run when pushing new code commits to the GitHub repository. Testing actions are triggered using the GitHub Actions feature.

3.5.3 Docker for deployment

Both NextJs and Python support building and deployment to barebones services without any supporting framework. This is nice when setting them up, but makes it impossible to reproduce and manage any given setup of these applications. Docker is the standard containerization platform and makes it possible to build containers for these applications to run. The project comes with Dockerfiles used to build the projects, which can be used in conjunction with Docker Compose. A Docker Compose configuration creates a collection of the services and makes it possible to run and build them together using a single command.

4 Results

4.1 Sprint results

The Gant chart created in the start of the project formed the outline of how the team would conduct its sprints. In total, four sprints were planned over the period of two months where the team had the opportunity to work in Sikt's office. The team did not set exact goals for what should be achieved at the end of the last two sprints, as it was strictly dependent on the results from the first two sprints. Leaving the two last sprints open gave the team the opportunity to adapt the direction of the project according to the feedback given by the stakeholders. This made it possible to spend more time reworking components to make them more flexible, instead of trying to adhere to a plan for when each individual component should be finished. The project was imagined to include a wider range of features like potential integration with other services, authentication system and a way of handling changes made to the rules, like a diff editor. These features were not requirements, but would be nice features to develop if the team had more time. These features were quickly discarded during the first sprints, as it became clear the developers would not have time to focus on anything other than essential functionality.

4.1.1 Roles and focus under development

The roles for each team member was originally planned to be decided by which user story they were responsible for managing. This would ensure that every team member would oversee or develop features across the stack, managing both frontend and backend code. During the first sprint, this relationship quickly deteriorated as each individual team member had to focus on one part of the stack, frontend, backend and Capirka. As soon as the team were under pressure to deliver the user stories for the first sprint, little time was left for each member to spend the required time to explore the different parts of the application, and each developer had to focus on their own domain to be able to finish their desired user story before the sprint would end.

4.1.2 Sprint 1

Sprint planning

The goals for this sprint were to be able to create basic rules in the system, as well as create a very basic configuration using Capirca. The use cases required for this behavior were created, and tasks were delegated within the team. Notable features: Creating UI baseline, creating state structure, creating endpoints for the API, filling the application with dummy data, start work on Capirca.

The sprint board was set up, containing all of the relevant user stories, which would be moved to the finished column as the development progressed.

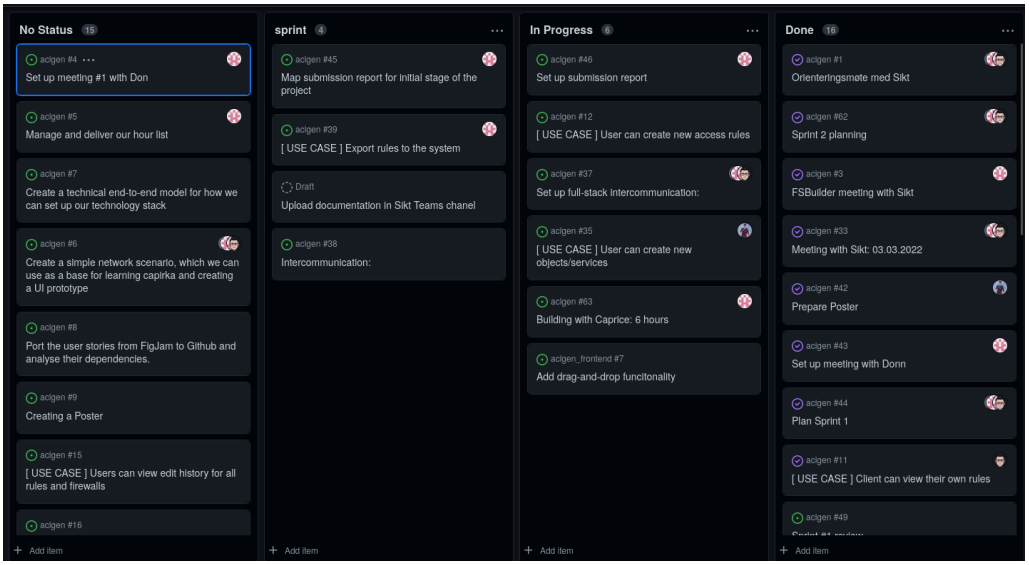


Figure 6: Issue board for sprint 1

Sprint review

The team had been unable to fully complete the main tasks of the sprint, partly due to the absence of a team member as well as having to spend time creating and presenting the poster.

Feedback: The rule objects were too big, and too much of the screen real estate was being used by rules. Services needed a distinction between source and destination in case engineers wanted to create rules targeting a special port on the target. Dragging and dropping services and objects was required before the project could progress further, and more work needed to be done before the application could be connected to the backend.

Sprint retrospective

The development team is a fairly small group. The team worked closely together on most features and took part in the planning of sprints and features. Before the reviews, the team has a meeting to discuss the week's performance, and how they should present the features to the stakeholders, this lead to the retrospective being largely redundant, as the team had already discussed the issued present in the sprint.

4.1.3 Sprint 2

Sprint planning

The goal for sprint 2 was to finish the functionality left over from last week and look into how elements could be dragged and dropped into input fields.

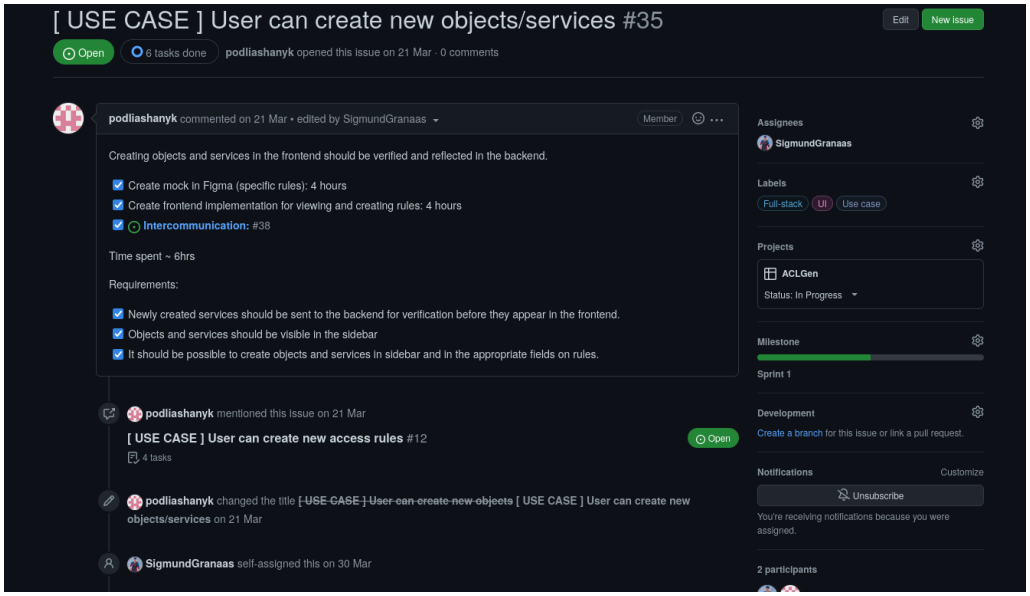


Figure 7: A use case indented to be completed by the first week, but was delayed because of it did not meet the specified requirements.

Sprint review

The second sprint of review for the development phase was discontinued, because of the Easter holidays. During this period the team worked on various tasks originally planned to be finished in the first sprint. Several stakeholders had availability issues due to the Easter holidays for the sprint review. This made it hard to schedule a proper sprint review, and the time available for development was cut short because of the holiday. The decision was made to continue working on client-server communication, which was a major source of issues.

Notable features: Working on handling state changes on the front-end, to make it possible to "save" what has been worked on. Connecting backend and frontend, using backend to load all data about rules, objects, repositories, and services.

Sprint retrospective

Sprint retrospective was not officially held, as the team had already discussed what needed to improve during a meeting before, and would be hard to complete due to the fact that the sprint didn't really complete as it was supposed to. A lot of the issues from last sprint were hard to mark as completed because they relied in some small feature that took longer than anticipated to complete. The retrospective were deemed redundant, as the team discussed issued as they occurred within the team. Spending time holding retrospectives did not uncover new information.

4.1.4 Sprint 3

Sprint planning

When planning this sprint, the team acknowledged that biggest flaw of the system was the inputs. Without a proper way of creating objects, and place them inside rules, it would be impossible to evaluate the state of the application. The focus of this sprint was to create a

solid way for user to compose rules of different objects and services.

Notable features developed: Implementing a system for dragging and dropping services and objects into rules. Creating searchable input fields. Working on creating a system for handling the creation of different objects, services and rules. Setting up a website, and containerizing the system.

Sprint review

Users wanted to be able to minimize and expand the rules, to make more room. Being able to conduct user tests using the public demo made testing a lot more efficient, as all stakeholders had the opportunity to test the application. It was also clear that there were scaling issues present for users which used increased zoom levels in the browser.

4.1.5 Sprint 4

Sprint planning

As the last opportunity to develop features for the application, sprint four would include the most important features which would be possible to implement. The team created a prioritized list of features with the importance ranging from highest to lowest. The development team knew that the time estimates gathered from earlier sprints were unreliable, and decided to start development on features and see how far they got.

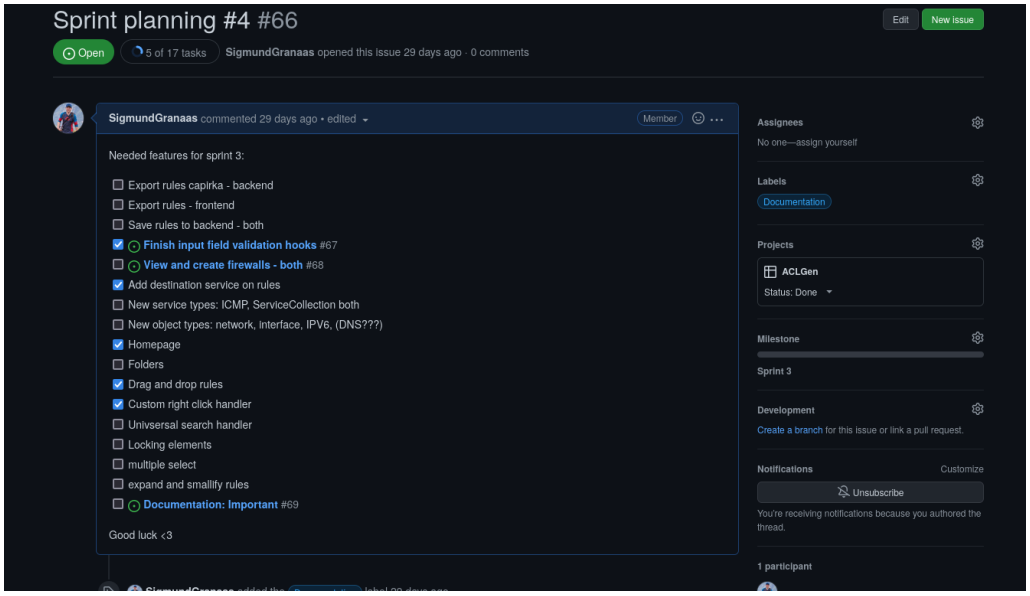


Figure 8: Issue containing all relevant features which could be included int the last sprint.

Sprint review

The team had an early sprint meeting during the first week of the sprint. The last review fell away due to the team having to focus on reports. During this meeting, the stakeholders and users would test the reworked inputs as well as voice their opinions on which features were essential to focus on during the last week of the sprint.

End of sprint results

Notable features: Drag and drop rules, right-click handler, saving rules, copying rules, Finishing the system for creating objects and services. Working on support for folders, making rules expandable.

4.2 Project results

The goal of the project was to create a vendor-agnostic application for managing and creating Access Control Lists. The project was not intended to be finished by the time the project was done, but rather to be a base that would see continued development in the future. The goals for the project can be broken into two parts, creating an accessible interface for creating and managing rules and creating a system for exporting these rules to a vendor-specific format. At the end of the development phase, the team had created a working application that had been used during the sprint review to test functionality. The application was used to demo the application from the first sprint review, this allowed the team to gather feedback regarding which stories was needed to focus on to make the application viable as an alternative to the legacy system. The iterative approach to development has to lead to some core components undergoing a lot of change, in response to feedback from the users during development. The results of this approach have led to the creation of components that support a variety of inputs, all of which are designed to reduce the possibility of making logical and input errors. The resulting application has implemented many of the features the team has recognized as being vital for using the application for its intended purpose.

The application allows the users to manage and create rules, objects, and services, as well as manage the rules by different devices. Most of the core features have been implemented, but there are a few missing components that would have to be developed before the system could be used to create access control lists for the a real system.

A lot of work has been spent researching integration with Capirca, but more work is required before the integration can be completed.

4.2.1 Functional Requirements

The table below shows an overview of functional requirements that are completed and not completed, with notes on the uncompleted requirements. The subsections below the table will describe how some of these features work through the user interface.

Functional requirement	Completed	Not Completed	Comment
Create a Repository	x		
Create, Update & Delete Network Objects	x		
Create, Update & Delete Service Objects	x		
Exporting to different formats		x	Could not complete due to time-constraints
Folder System		x	Downprioritized due to time-constraints
Showing Difference ("diffing") on changes	x		
Drag & Drop for elements	x		
Global Repository for pre-made objects		x	Downprioritized by the team
User System for staff		x	Low priority
User System for customers		x	Low priority
User Interface	x		
Search		x	Search is not implemen -ted on a site-wide scale
Request Form		x	Downprioritized by the team

Table 1: Overview of completed and not completed functional requirements

Application overview, and core functionality

When users first access the website, they are presented by a list of repositories they can choose. Repositories are modeled after Git's repository, and exist to create a separation between workspaces or projects. Repositories is a concept that see little practical use in the application, but would be very important if the scope of the application would be expanded. Repositories are designed to make it possible to manage access to different project and workspaces to different users or clients.

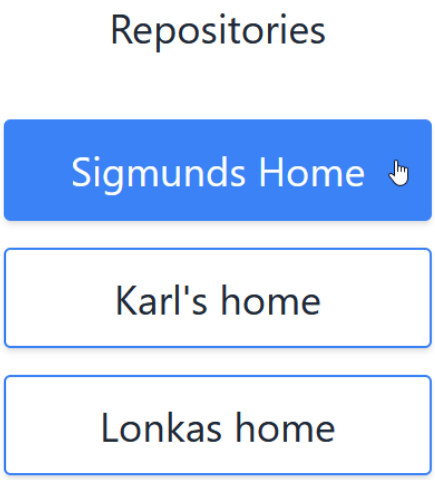


Figure 9: Repository selection. A repository contains all related devices, rules, objects and services. When first entering the application, the user is given this choice

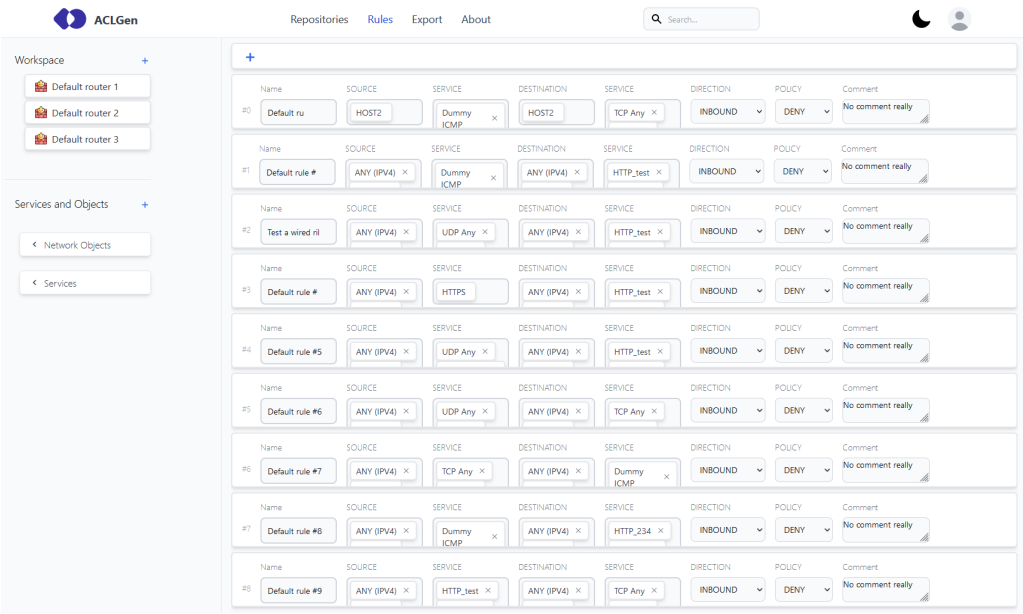


Figure 10: After selecting a repository, an overview of the repository is displayed. The overview lets the user quickly manage repository objects

Rules are the primary focus of the application, and the space available is centered around providing the most amount of information about the rules. Results from user tests revealed that the users preferred smaller rules, and would like the redundant description of fields replaced by a single descriptive row at the top. This issue was not prioritized, as increasing the number of visible elements also had a performance impact on slower systems from rendering more input fields.

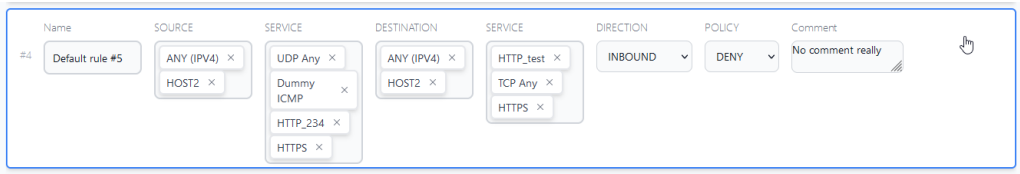


Figure 11: A rule is normally collapsed to save space and declutter the overview. Clicking a rule expands the box, so that it displays all related information

Comparing the initial design mocks for the application reveals a lot about the development of the application. The application has almost been developed identically to the design mocks. Spending time creating a good design foundation has helped the team focus on building useful features for the application instead of having to spend time during the development cycle refactoring code. Some elements have been removed or reworked due to feedback from the stakeholders. Most notably is the addition of several inputs, as well as placing elements inside the inputs, instead of text. This makes it obvious that there is an element inside the input, as opposed to a piece of text. Some other elements are not present in the final application, because the development of these features has not been prioritized.

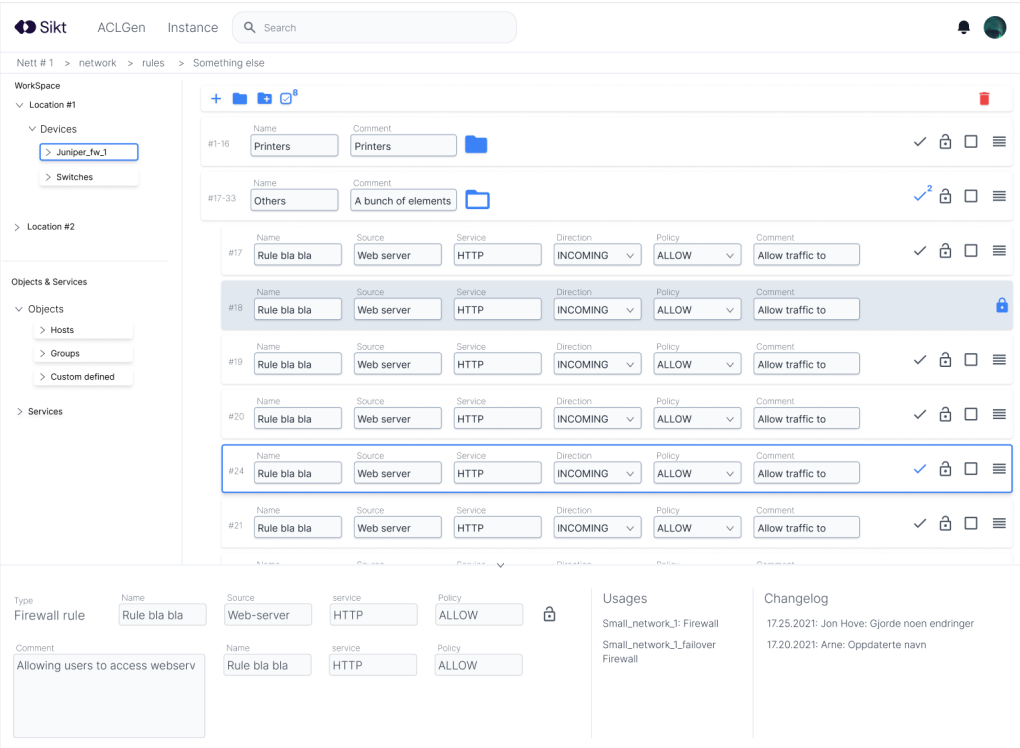


Figure 12: Overview of the design mock before the development began

Creating and editing new services and objects

Creating new services and object can primarily be done clicking the plus icon next to the "Services and Object" menu. This menu will prompt the user to create new services or objects.

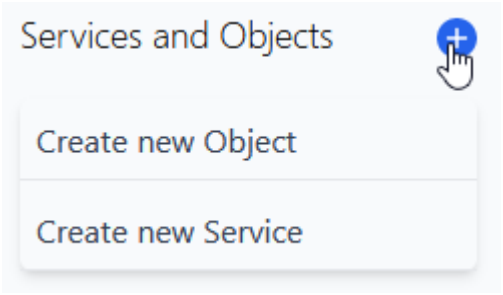


Figure 13: creating a new IPV4 object

The user can manage services and network objects from the left pane while inside a repository overview in the web application. Pressing the blue plus icon gives the user the choice to create either a network or a service object. Creating a Rule also has this approach, in another area of the user interface.

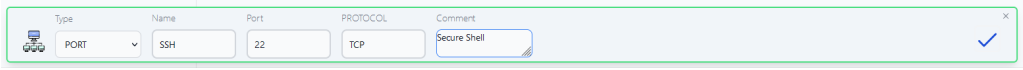


Figure 14: Creating a new Port service

There are different services present in the application. By clicking the drop-down menu next to the name of the element, you will be prompted to choose the type of service. When switching service types, all applicable data will be transferred to the new service. Some services have shortcuts available when typing. Pressing the hyphen key while writing the port number of a port service will turn it into a port range, and copy all relevant data to the new service.

To save a service or an element, click the blue check button on the right side of the pop-up.

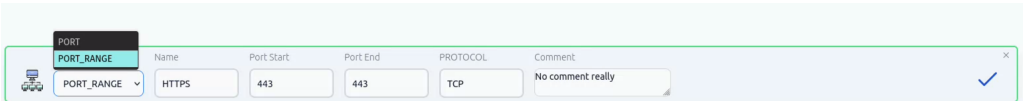


Figure 15: Using the drop-down menu to switch types

Editing objects and services are possible by clicking them in the left-hand menu. A gear icon will be highlighted when hovering these elements. The same menu is used for editing and creating elements. If the user inputs invalid data, this input field will turn red, and a response message will be presented. If any invalid input is detected, the check button will be disabled, and the user will be prevented from making this change.

When editing elements, the user will also have the option to delete the element by clicking the trash can next to the submit button. This option is only available when making changes to objects, as there is nothing to delete when creating an object.

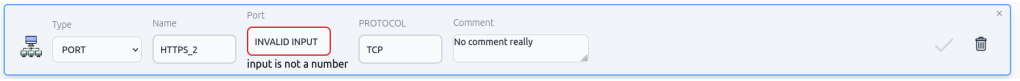


Figure 16: Example of invalid input

Network objects utilize the same input pop-up and functionality as the services but have unique inputs and types.

Figure 17: Same approach as when creating a Service object. The user selects the type of object and enters relevant information

Editing rules by drag and drop

Because the rules are in focus in the main panel, they can be expanded and edited without having to be clicked on. Editing rules is as simple as changing the text fields or dragging elements into accepted inputs. Objects can be added to a rules destination and source, while services can be placed in services. While dragging elements, invalid inputs will be greyed out. When dropping elements into a rule, the rule will expand by itself, making it possible to see the entire contents of the service. When dropping an element that is already present into a field, nothing will happen.

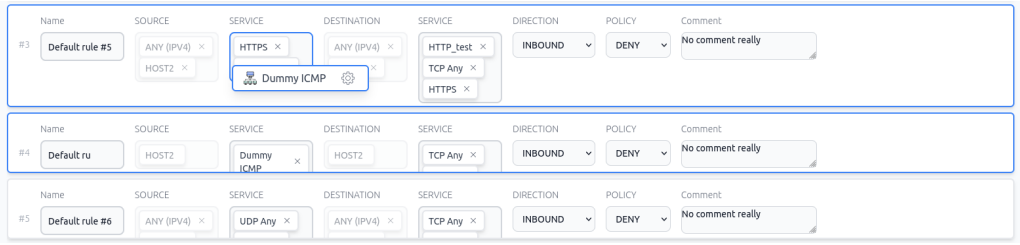


Figure 18: Dropping a service into the service field

Using the search functionality inside the input components

A faster way of editing rules is by using the built-in support for searching in the input field. By clicking or focusing on the input, a popup will appear allowing the user to search for valid objects. This menu allows you to add or remove elements that correspond to your search by clicking on them. The user will not be able to remove the last element inside an input, as this would make the rule invalid. This component searches as the user types, which makes it easier to get an overview of which elements are available, and makes the searching process faster for the user.

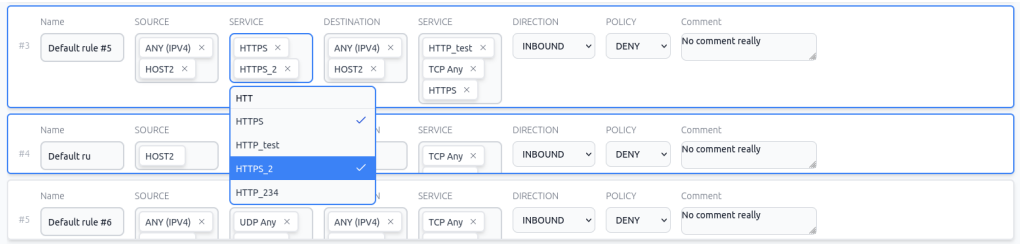


Figure 19: Searching for services

If no elements are matching the users search, there will only be one option present, which will allow the user to create a new service matching the name the user are searching for. Clicking this element will trigger the new service/object action described earlier.

Keyboard support

All of the inputs for rules are fully keyboard traversable. If a rule is focused, hitting the tab button will cycle through the input, and automatically show the search input popup for supported inputs. The search menu can be navigated with arrow keys, and pressing the enter key will add the focused elements. If the focused element is already present in the input, it will be removed. By tabbing when inside a searchable input, you will move to the next input and expand it. When the end of the rule is reached, it will cycle down the list highlighting the inputs for the next rule in the list. This will continue until you hit the end of the list.

Creating rules

The primary way for creating new rules is by utilizing the rule action bar. The plus icon will trigger an action for creating a new rule. The bar provides actions that can be used in the present environment. Only save and new rule actions are available in the application. Rules can also be created from the right-click menu, allowing new rules to be created from anywhere the application.

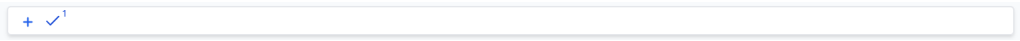


Figure 20: Rule action bar

Creating rules is done almost the same ways as services and objects. The main difference is that rules cannot be created without having filled its service and destination fields.

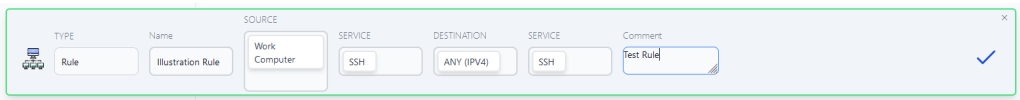


Figure 21: Rule creation popup

Right click functionality

Another way of quickly creating elements is by utilizing the right-click action menu. This menu will give the user quick access to some useful features, like creating new objects, ser-

vices, and rules. The right-click menu is available everywhere in the application.

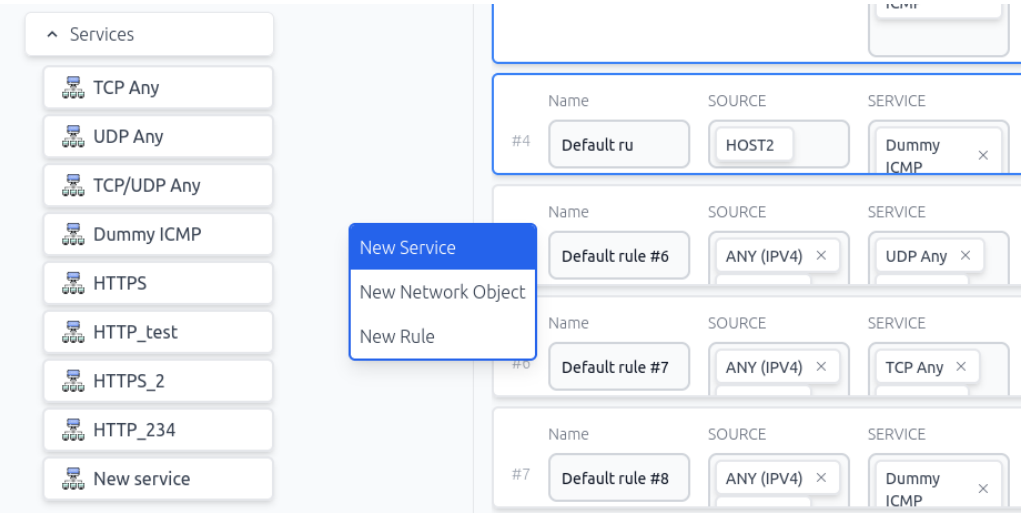


Figure 22: Right click menu

The right-click menu is also context-aware. By right-clicking on a specific element, the menu will give the user options for actions specific to the element which has been right-clicked. When right-clicking elements, the user will have to option to copy this element, which will copy all of the attributes into a new object of the same type. The user can also delete right-clicked elements, but this option has been moved to the bottom to prevent users from accidentally clicking this option.

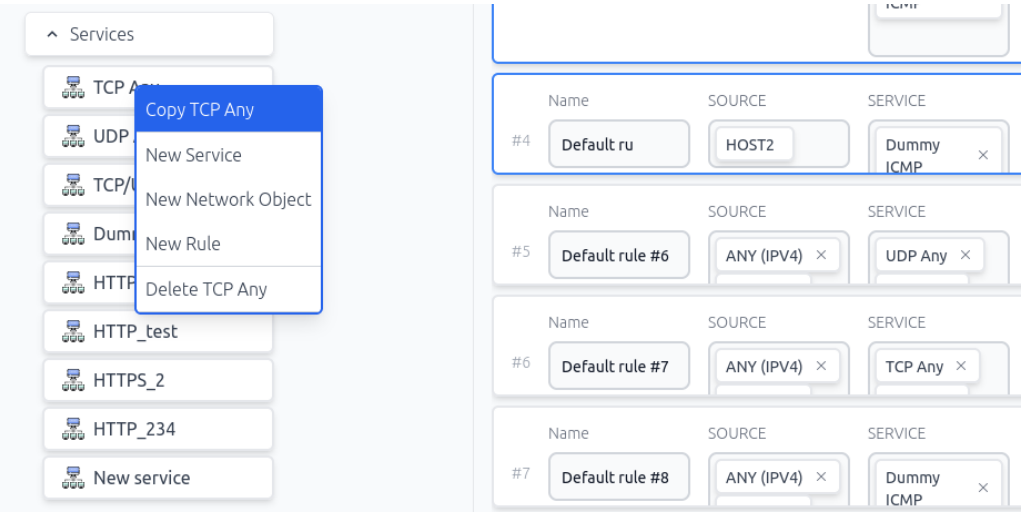


Figure 23: Right click with context

Building reusable, flexible components

To make the system as flexible as possible, a lot of development effort has been put into making core systems work with a variety of different types that share some common traits.

There are three types of objects presented thus far, a service, an object, and a rule. All of these elements share a common type, which is an EditableElement and has some core attributes. All of these elements have a name, a unique id, and a status. The status can be either "source", "new", "deleted" or "modified". These attributes are used throughout the system to enable common functionality. All new elements in the system have a green outline, this is a visual difference from the edited status, which will create a blue outline around elements. Elements with the deleted status will simply not be visible to the user.

By utilizing and sharing common types, it has been possible to create versatile components that know very little of the element they represent. This makes it possible for all new services and objects that are added to be searchable and droppable in the application without having to make any changes to the system. The only part of the application which would have to change when adding support for new objects and services is the pop-up menus where the user can specify type-specific attributes.

Keeping track and persisting changes

Preventing the user from making errors is a core functionality of the application. Providing a clear visual indication of which changes have been made to the network configuration is important for making the application more accessible for new users. A robust system for tracking and comparing modified elements is implemented by layering the entire state of the application and slicing the core components of a repository into editable zones where the user can make changes. All changes and edits made in the application are made locally on the client. When an element has been modified or created, a check icon will appear next to it. Clicking this icon will persist the element to the server.

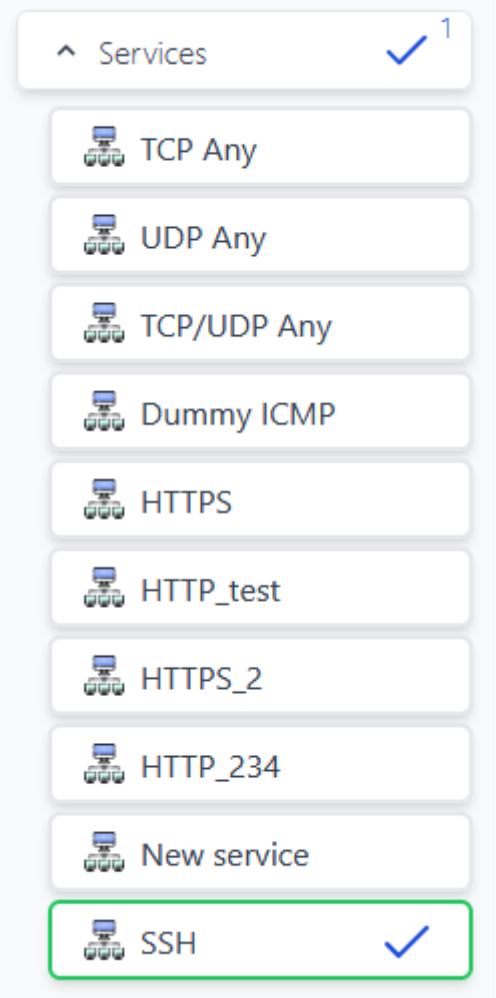


Figure 24: Changes made to rules, objects and services are made locally on the client.

A copy of the repository is always kept by the client. Whenever the user triggers an action that will save data to the server, a full refresh of the repository will also be client-side. When the refreshed repository is fetched from the back-end is received a merge operation is started. This operation will compare every single element received from the server to the element present on the client. If the client contains a modified element that is present on the refreshed data from the back-end, the modified element will be kept, preventing the client from clearing all modified data on the client. This system makes it possible to save all elements individually and make incremental changes to the state of the application on the server. This will also incorporate changes made by other users to the system every time a save operation is made. By clicking the counted check button, you will save all changes made in the relevant group of elements.

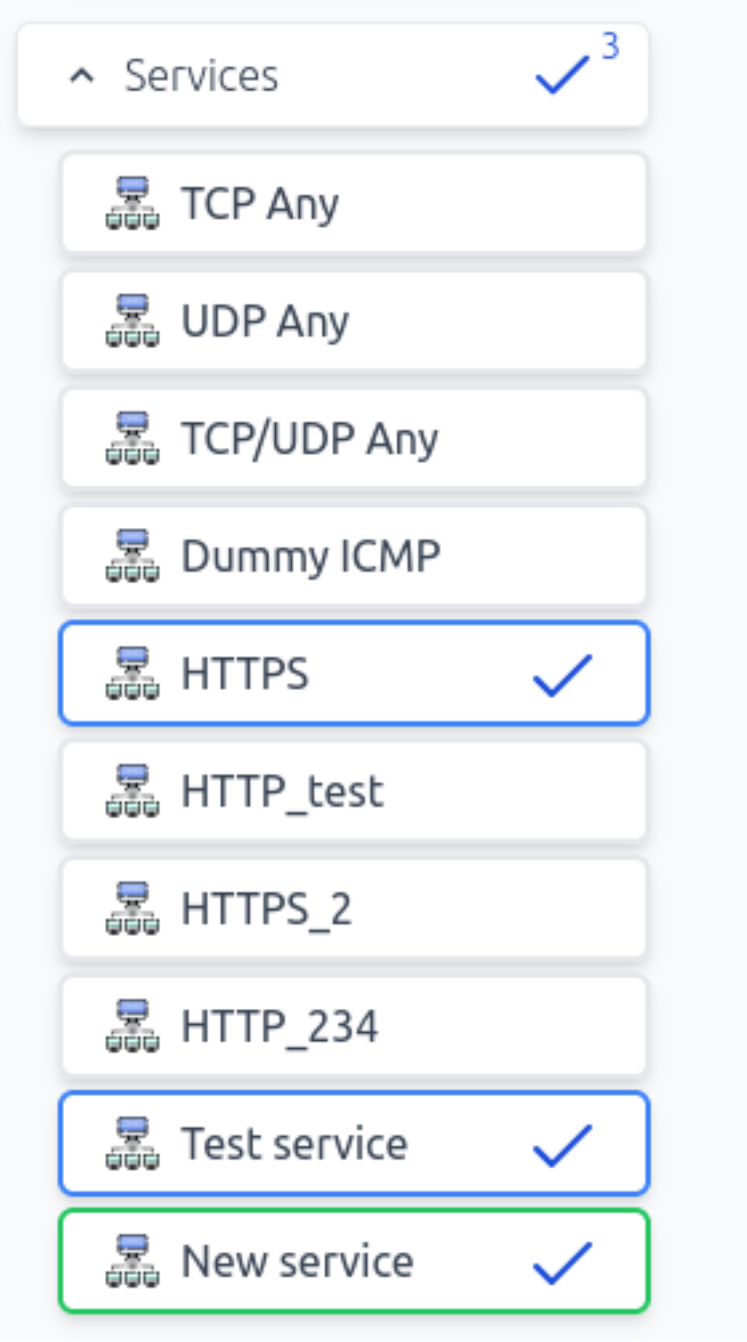


Figure 25: The changes made will be tracked and counted

This system was also intended to power a feature for making it possible to restrict editing by certain users. This was envisioned to work similarly to how pull requests work in popular Git services. A design mock was created for this feature, but never planned in the development roadmap, as it would be beyond the scope of this project.

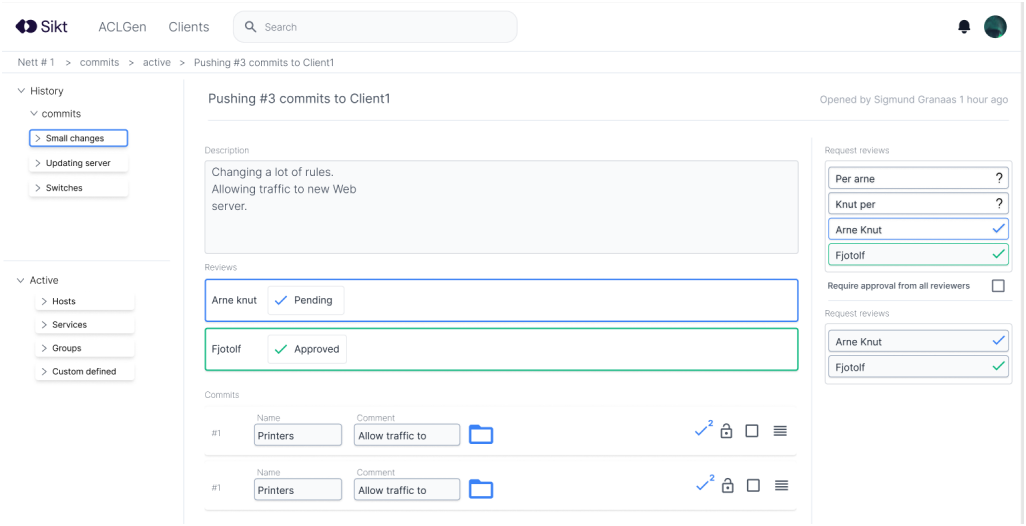


Figure 26: Design mock for how users with less privileges would be able to make changes to the Access Control Lists

Switching between Firewalls

The list of rules visible to user is tied to a single device. The application supports several devices. By selecting different devices, the user can view different lists of rules. Changes made to a device will be stored on the client and kept when switching between devices.

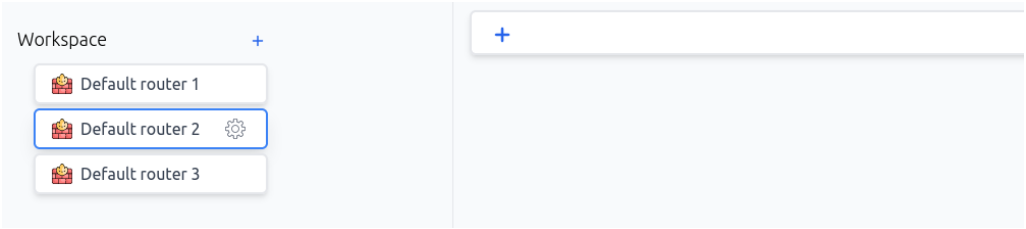


Figure 27: A firewall with an empty ruleset

Deployment

To properly test the application, a public server was set up using docker Traefik and HTTPS. This would allow the team to conduct tests where users could play around using their machines. The public demo is available at https://aclgen.com and works as an interactive demo for anyone to play around with. Setting up a public-facing application early made user testing a lot easier, as well as making the team fix issues related to setting up production services, making the application easy to set up for anyone wanting to try it out.

Missing features

Due to time constraints, some features has not been implemented.

Creating firewalls, vlans, folders, generating firewall configurations.

4.2.2 Non-functional requirements

Documentation is the key non-functional requirement defined in the vision document. This requirement is fulfilled by writing unit tests and commenting the code on both the client and server component of the web application.

Backend - Model Relationships

Below is a Entity Relationship Diagram to visualize the relationships between the models in the backend.

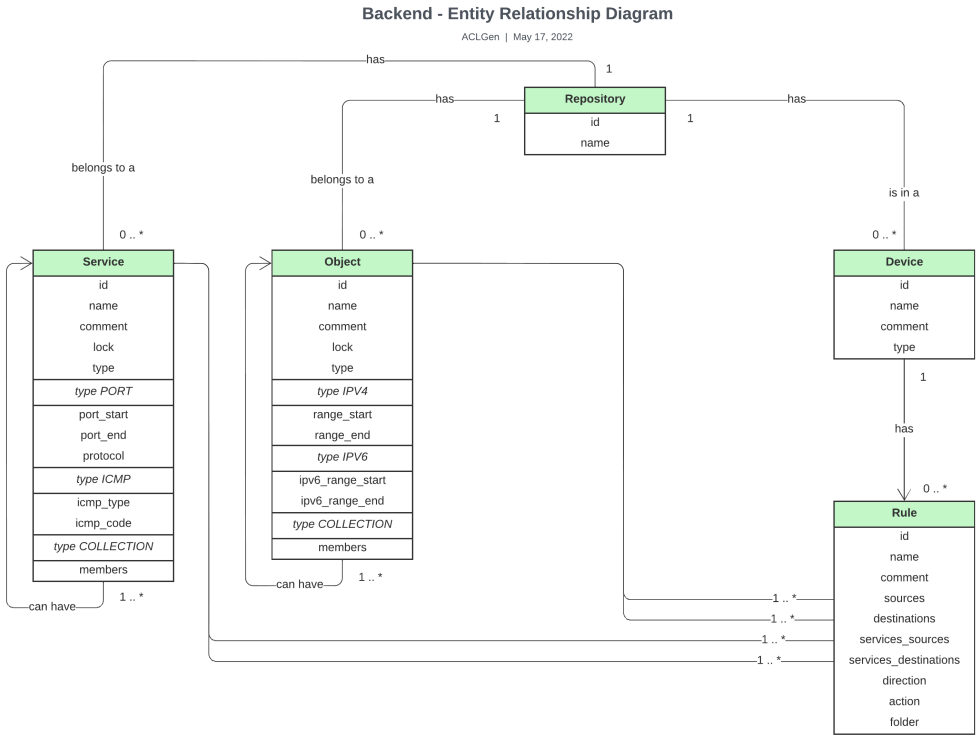


Figure 28: Entity Relationship Diagram. The figure shows the relationships between the models that exist in the REST API

4.3 Administrative results

4.3.1 Time savings due to choice of technology

Some of the technologies chosen for development, like Typescript, provides zero functionality to the project. Typescript is a tool aimed towards making it easier to write correct code, and making it easier to refactor code. The time benefits associated with Typescript is hard to measure, but it certainly has an up-front cost when it comes to creating types for every single object and function you create. When building the front-end application, types would always be checked for correctness. This made it impossible to build the application unless all type checks passed. Validating and writing types takes a lot of time, but was greatly appreciated when the team started expanding the footprint of central objects, and made functionality changes, like adding folders, and a lock status. By expanding the types, the build system

would automatically find and highlight errors in the code, which saves the team a lot of debugging when making changes to core types in the system. The time saved during development due to automatic type checking cannot be measured directly, but it is estimated to be equal to the time spend implementing and maintaining the types. This result is not impressive, but the positive impact of types would only continue as development continued.

4.3.2 Time sheets

The team continuously logged project-related working hours on timesheets. The timesheets document time in hours spent by categories such as research, project development, and daily and weekly meetings to mention a few. The team has also documented in short detail what each team member has been working on, every week. The figures below give an overview of time spent on the project as a whole, including time spent by each team member by category, and the total time spend by category. The complete timesheets with weekly status reports are included in the project handbook as an appendix.

IDATT2900 - Bachelor Project 47
Accumulated time in hours for the ACLGen bachelor project

Sigmund Granaas Sandring	530,5 hours
Karl Klykken Labrador	522 hours
Ilona Podliashanyk	519,5 hours
Total, all students	1572 hours

Started:	10.01.2022
Ended:	20.05.2022

Figure 29: Overview of accumulated time in hours for all team members

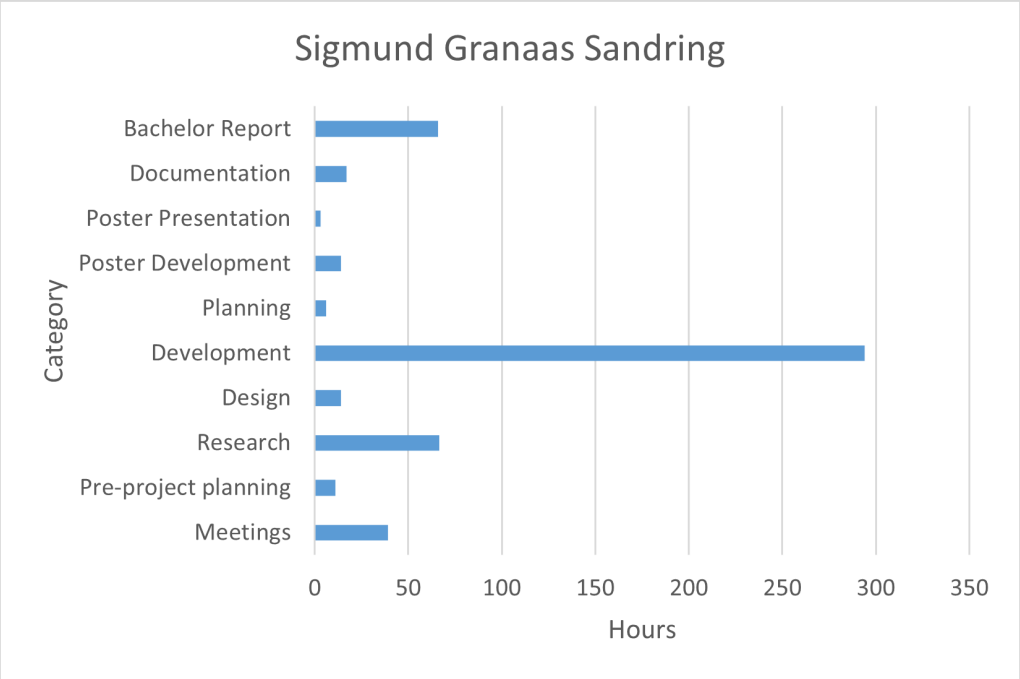


Figure 30: Accumulated time spent per category in hours by Sigmund Granaas Sandring

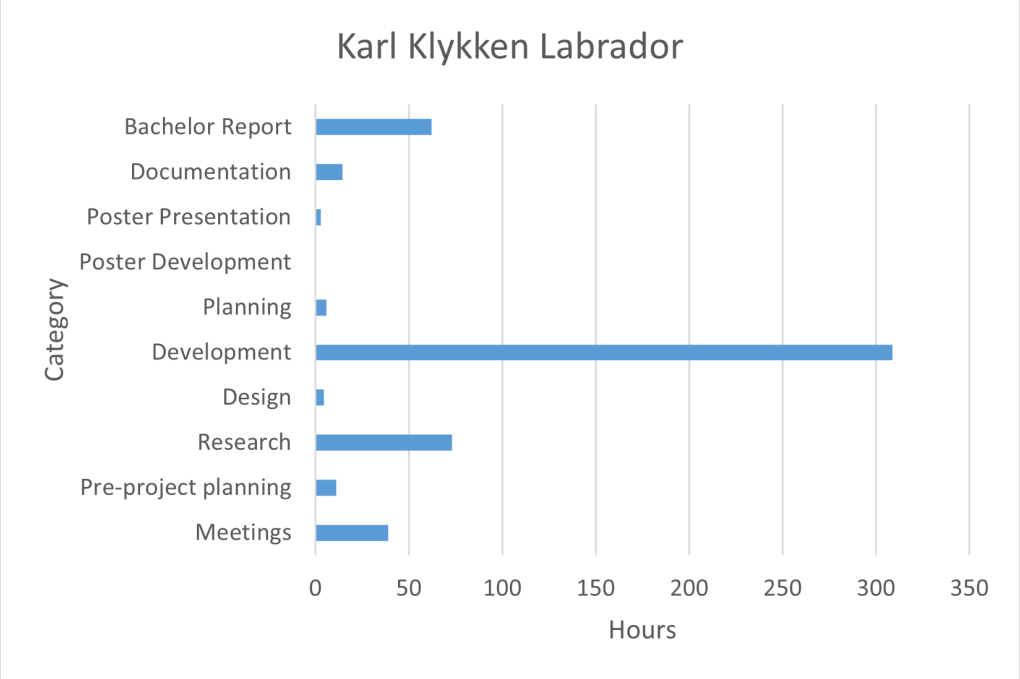


Figure 31: Accumulated time spent per category in hours by Karl Klykken Labrador

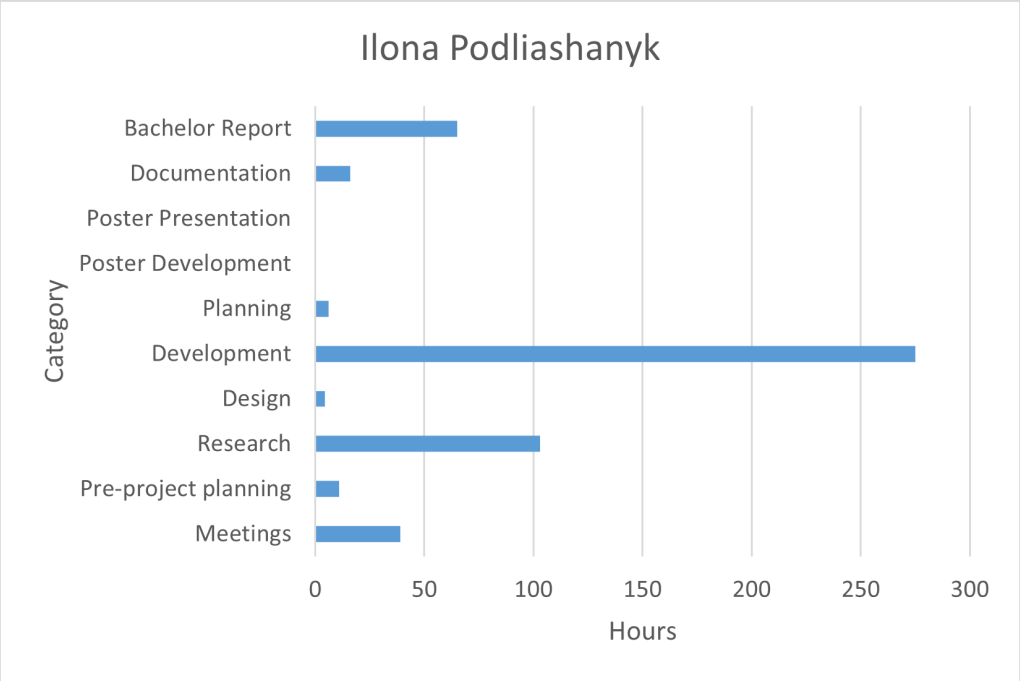


Figure 32: Accumulated time spent per category in hours by Ilona Podliashanyk

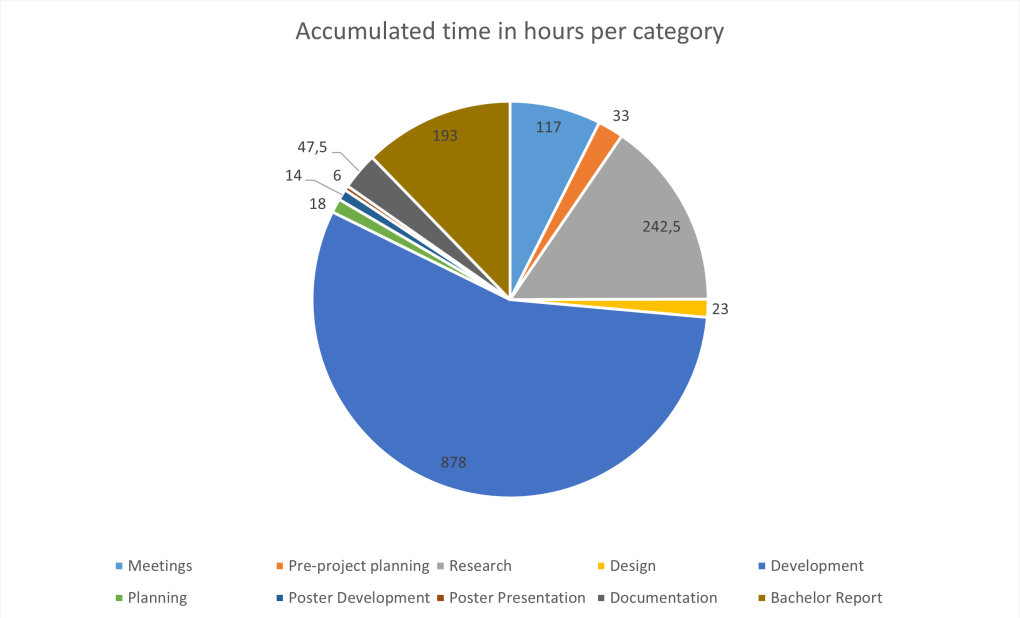


Figure 33: Accumulated time spent in hours by all team members per category

5 Discussion

5.1 Project Results

Results from the previous chapter are discussed in this section.

5.1.1 Functional requirements

Implemented features and the overall status of the project were presented in the results chapter. It shows that only half of the functional requirements presented in the table were completed. Developing some of the features took more time than initially anticipated. Most of the technology chosen for this project were new technologies with which the team had no prior experience. This introduced a learning curve and affected the pace of the feature development process. During the development period, some features were down-prioritized and some features were dropped due to complexity and time constraints for a single feature. However, the team is satisfied with the current implementation of the completed features.

Between the sprint periods, it was advised by the employer to not focus on a few features, notably the user authentication and permission system. As the project is initially a stepping-stone and a base concept for future development, the team did not expect to completely fulfill the functional requirements. It was important for the team to deliver well-documented and polished features and a foundation on which to continue development.

5.1.2 Non-functional requirements

As the project result is the foundation of further development, it is important to have documented code and features in the web application. This has been fulfilled by writing unit tests for endpoints for the REST API and documenting the code in both the client and server components of the web application. The team would have liked to implement Swagger, which is an API endpoint documentation tool for REST APIs, but due to time constraints, this was not completed. The Django REST Framework has a built-in interface to test endpoints via the web browser. The team found this feature of the framework to satisfy the needs during development and deemed it more beneficial to focus on completing features. Models and endpoints in the server component are subject to change in the future.

5.2 Research

The accuracy of the research being put into the users for an application scales well with the number of users the team can conduct tests on. Developing an application intended to be used by only a couple of people presents some interesting challenges in regards to gathering reliable data from users. The main benefit of this situation is that the targeted users can be interacted with directly, this allows the development team to interact with the users and watch them interact with the software during sprint reviews. Receiving direct concrete feedback from the user during every sprint makes it easy to prioritize which features need to be developed, as the users can describe in detail how they need features to work before they can use the tool to create Access Control Lists.

The downside to having such a small userbase is that the application could be harder to

expand to a wider audience. When the application is built with only a small set of users, the feedback will only reflect some user's needs, and might cause the team to implement features in a way that could hinder further growth of the application.

5.2.1 Scrum / agile

Scrum as a development process was used at the start of the development phase but was utilized less and less as the development progressed. Using scrum can give a team a lot of insight into how much time different tasks take, as well as creating a predictable schedule for testing and approving features in development with the stakeholders. Utilizing time-boxed sprints was very useful for coordinating testing with stakeholders and prioritizing which use cases would need to be developed first. Being able to iterate on feedback from stakeholders during every sprint, was extremely helpful for the development team as issues and improvements were being discussed as soon as they were developed. As team sizes grow, more time has to be spent managing team members to make sure the development process is as efficient as it can be. Investing time in team management makes a lot of sense when development stability and schedule are important.

Results from the early sprints showed that the time estimations related to stories were off by half of their real amount, as well as small issues which had not been accounted for like bugs and unforeseen dependencies in the system took half of the entire development time. Time measurements for use cases and features that depended on systems or components that needed development, like Drag and Drop and searchable input components, were completely unreliable. Applications in the prototyping stages are hard to measure, as there are a lot of unknowns. Some of the processes from scrum were discontinued quickly, as they took time, and introduced a lot of management overhead, which were quite redundant for a team of three people, especially as the project was nearing an end, and development was ramped up to be able to finish desired features before the project ended.

Utilizing scrum on a mature development project would be ideal, as it would provide tools for the team to create predictable sprints, ensure consistent releases and create maintainable work habits for extended periods of development. This project did not share the same characteristics and might have fared better by trying to implement some concepts from XP, without putting too much effort into CI/CD.

5.2.2 Frontend Technology

Using a mature framework like NextJs has helped development immensely. Utilizing opinionated designs makes development faster, as there are fewer decisions that have to be made. NextJs is packed full of features, but very few had an impact on the final application. Using SSR would make loading times faster, but was not implemented in a way that made any difference in the application. Enabling support for this would be when the project continues development, but it is hard to imagine it would be prioritized as the application is very likely to be deployed on-premise. This means that the framework chosen for development was arbitrary, and could be replaced by any other framework using react as a rendering library. Still, there are plenty of quality-of-life features in NextJs that simply make the framework pleasant to use. The surprising result is that developer convenience played a bigger role than features for the project.

While the choice of using NextJs had a surprisingly little effect on the project, using Redux had a surprisingly big effect on the project. Actions and reducers it not an exclusive concept

for Redux, as Redux internally used React's context API under the hood. Having an opinionated state framework like Redux was a key factor in creating a complex state tree supporting a huge amount of data. Being able to slice pieces of the state into chunks that could be worked on independently and merged with the root state in a predictable pattern using actions, was crucial to making it possible to render all components efficiently and support the behavior for merging data coming from the server with modified data on the client.

Drag and Drop is an incredibly useful tool for the application, but it has some severe drawbacks. The application suffers from performance issues on older hardware. These are issues that can be fixed, but requires a lot of work to set up correctly. In addition to this, it can be hard to create an application where components are not directly coupled and dependent on the drag and drop library. Managing interactions between drag and drop handlers is hard and prone to causing issues breaking the entire application. Implementing features like drag and drop makes the project significantly more complex and harder to work with, but also serves an incredibly useful function in the software. Reworking the drag and drop implementation would make it possible to integrate with more advanced features like nesting folders by dragging elements inside each other, but adding these features would have to be considered carefully, as the time costs for this would be huge.

The biggest issue plaguing the application is issues relating to creating custom input components. All major browsers support most of the same specifications. Creating applications that implement the normal inputs and operations work well across browsers. A lot of problems arise when trying to create custom input components, and issues vary from browser to browser. Different browsers handle the application differently, and a lot of time was spent debugging functionality that worked perfectly fine in one browser but was useless in another. Spending time writing patches for specific browsers take a lot of time and makes the code harder to understand. Straying away from the well-supported standards will make development significantly harder, but might also be necessary for the application to serve its intended purpose. targeting a single browser might make developing applications like this a lot easier, at the cost of limiting the users' freedom. At the current state of the project, dealing with browser-specific oddities is manageable, but it might be a major point of concern during the future development of the application, and targeting a single browser should be considered with the discovery of more browser-specific bugs.

5.2.3 Backend Technology

Django and Django REST Framework initially has a steep learning curve, as there are many elements that go into creating a single task. It is a powerful framework and toolkit for building REST APIs. The team had many thoughts regarding features and what could be developed using the framework with the assumption that past experience with other frameworks in another programming language (particularly Java, and Spring Boot) could also apply to Django with Python. The team had challenges developing complex backend solutions supporting a variety of object types tied to the same endpoint, such as services and network objects. The team had to settle with a less cleaner approach by supporting multiple types in the same model.

5.2.4 Teamwork

The basis for joining together to form the team was that the members have previously worked together on previous development projects related to the field of study. Past experiences on

development projects have proven the dynamics between the team members to be good.

Initially, the team worked closely and organized sprint milestones. As the project progressed, it was obvious that how much work was being done was closely related to how effective the team could communicate. When working from home during the research part of the project, the team was not able to work efficiently, partly due to fragmented time schedules, and having to use digital tools for communication and collaboration. This became apparent when the team could start active development in Sikt's offices, as the results from development were far exceeded expectations when physically present, as opposed to when the team had to work from home. There is a huge difference between working hours and effective working hours, which cannot be deduced by looking at the reported hours worked per week.

Some team members were directly affected by the war in Ukraine. Some events are more important than a bachelor project, and there are problems that cannot be solved by simply changing methodologies. In situations like these, predictable results cannot be expected, and the team is grateful for all the effort that has been put into the project by all team members during this period.

5.3 Effects

A completed web application to manage access control lists may have positive effects on network security and managing networks in general. A wanted effect from the software is increased accessibility to contribute to safe and fast changes to access control lists, with built-in validation tools for a more reliable self-service approach in the form of self-validating request forms.

6 Conclusion and Future Work

The problem statement that was introduced in the introduction was the following: "*Exploring the possibilities and problems related to adapting and improving a native legacy application into a web-based solution utilizing a modern software stack without compromising on functionality*".

In this report, technology choices and the development process for the web application have been discussed. The team has focused on exploring the possibilities related to the problem statement.

6.1 Conclusion

The original assignment was focused on building a functional application with a complete feature set. As the development progressed, it became apparent that the application would have to sacrifice some core features and have challenges fulfilling its goals given the constraints of most web-based solutions. A significant effort would have to be invested into the development of the applications to avoid having to sacrifice core features, like dragging and dropping elements to compose rules. The goal of the project was changed to focus on creating solid features which could be developed further and expanded upon by Sikt, as a solution that sacrificed core functionality would not prove useful compared to the legacy application.

6.1.1 Adapting native features

Creating web applications has reached a point where it is so simple to set up and start development, that it would be the go-to standard for creating simple applications. Web applications dominate the market because of their accessibility and ease of development, but they cannot beat native applications in every category. Native applications have access to some well-built and polished APIs. Making efficient use of native support for dragging and dropping elements can make some applications very useful for certain tasks. Adapting this functionality to the web has some challenges. Web applications work well across different browser vendors when following official specifications. When an application requires expanded functionality beyond what is provided by officially supported APIs, problems emerge. Because most browsers are implemented differently, their response to components and events in the DOM might be different. This increases development time and makes it harder to test applications. As long as the problems related to browser irregularities can be worked around and patched, the results from developing advanced features can match native implementations. In addition to this, the accessibility provided on the web is truly unmatched, making it clear that if an application can be developed for the web, it probably should be.

6.1.2 Performance

Emulating native functionality has some drawbacks to running in the browser. Recreating large data sets, performing mapping and filtering functions, and handling thousands of events triggered when elements are being dragged creates performance issues as the scale of the application increases. These issues can be worked around, by utilizing virtualized components, creating asynchronous handlers for doing heavy operations, and making less desirable design

decisions to mask certain performance issues, but it is not necessarily something a developer would need to worry about when developing a native application. Web applications will perform perfectly fine when doing most tasks, but require optimization when pushed to the edge of their capabilities. Some problems like excessive re-rendering can be solved by clever structuring of components, but other issues, like latency related to communicating with a backend, is an issue caused by the client-server architecture and distance to the targeted server. As the project's intended userbase is quite small it is possible that developing a native application could help the project reach more of its goals within the given time frame. This approach would limit the opportunities the application would have for expanding to more users and potentially external clients, which would make it a less appealing option if more resources are planned for the application's development.

6.1.3 Missing features

This project requires more development before it would be able to replace the legacy software. Most of the development time has been focused on researching core needs for the application and developing solid methods for handling a wide variety of elements and inputs. There are a wide variety of protocols and network objects, spending time implementing as many of these as possible would quickly eat up all time available for development. The project's success lies in the long-term strategy for continued development. More time is needed to finish implementing the core system for translating the rules into configurations for specific routers and devices, which is the biggest limitation concerning the viability of the application.

6.2 Future Work

There are some core features missing from this application that needs to be implemented before it can be used to fulfill its goal as an application for creating and managing Access Control Lists. The following points provide a starting point for which features should be prioritized to continue testing the viability of the application in more challenging scenarios.

6.2.1 Exporting access list configurations with Capirca

The work related to implementing Capirca has just begun and is in the early documentation phase. By taking advantage of the Django backend, it is possible to use Capirca's libraries to create a translation layer between rules generated by the system, and device-specific configurations provided by Capirca. In addition to this, generated lists can be tested using Capirca's built-in tools for validating rule sets. This would make it possible to run simple tests against generated configurations for heightened security and error checking. Further work includes developing software to build term blocks that are compatible with Capirca's configuration syntax.

6.2.2 Creating and managing devices

Devices are a core component of a network, and the system would not work properly without being able to configure network devices according to the environment it is supposed to represent. Development effort needs to be put into leveraging the system for creating Services and Objects for adding support creating and editing network devices. The framework for creating and editing elements is already developed but requires more integration than creating a new type of service. Support for creating Vlans under devices and managing each

Vlan's rules independently would also be an important feature to make it easier to manage rules for a specific device.

6.2.3 Folders

The application can benefit from having a system for folders. This is helpful for organizing rules, objects and services when a repository increases in size. In the current state, building a large set of rules may clutter the user interface and make the user experience less efficient. An implementation of this feature was attempted, but it was dropped due to the complexity and time constraints on the backend. A folder must be able to contain different types of objects. This may be solved by developing an implementation using the GenericForeignKey feature that is available in the Django framework.

6.2.4 Request Form

A request form is an initially requested feature, but it was not prioritized by the team to focus on core functionality. The application will later benefit from such a form when the application is more complete and taken into use by the networking engineers. The purpose of this feature is to allow customers to request changes to firewall rules with values that may be directly fetched from the user interface to create or update a rule.

6.2.5 Extending object and service types

Only basic types for objects and services have been implemented. Adding support for different objects and services like IPV6, object collection, and DNS objects is important to extend the functionality of the application. The framework for handling different objects like this is already developed and the effort of creating these objects can be focused on the specific attributes which define the object and how to validate its properties.

6.2.6 Global Repository

The users of the application can benefit from a global repository. The purpose of a global repository is to store template rules, objects, and services that are commonly used across all networking devices. Such a feature will enable networking engineers that make additions or changes to rule sets to quickly import a copy of common objects.

6.2.7 Search

It may be beneficial for the user experience to have a search feature. This can be helpful when managing large access control lists and will improve the user experience by letting the user navigate through objects by naming conventions, instead of having to scroll and dig through files and collections of elements.

6.2.8 Extending testing suite

As the application only has support for core functionality, more testing is needed to validate that this functionality does not break when adding new features. As the project has experienced severe time pressure, testing has been not been prioritized, as there are still core features missing from the application, making it less viable for use in real environments.

6.2.9 Rule Ordering

Rules are dependent on an order position in the configuration file, as firewall configurations are sensitive to what order the rules are defined. Finding a satisfying solution to solve this

problem has not been successful, due to complexity and time constraints. A potential solution could be allowing the user interface to set the orders through a separate endpoint, or implementing a Linked List data structure into the Rule model.

References

[1] Uninett. fwbuilder. https://github.com/Uninett/fwbuilder, 2015.

[2] Uninett. Lokalnett fra uninett - cnaas. https://www.uninett.no/lokalnett. (Visited 20 May 2022).

[3] Andrew Stellman. *Learning Agile: Understanding Scrum, XP, Lean and Kanban*. O'Reilly, 1 edition, 2014.

[4] Jeff Patton. *User Story Mapping*. O'Reilly, 1 edition, 2014.

[5] Roy Fielding. Fielding Dissertation: Chapter 5: Representational State Transfer (REST). https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm, 2000. (Visited 17 May 2022).

[6] Codecademy Team. What is REST? https://www.codecademy.com/article/what-is-rest. (Visited 17 May 2022).

[7] MDN contributors. Understanding client-side JavaScript frameworks. https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks, 2022. (Visited 18 May 2022).

[8] Peldi Guilizzoni. What are Wireframes? - balsamiq. https://balsamiq.com/learn/articles/what-are-wireframes/. (Visited 18 May 2022).

[9] Atlassian. What is version control? https://www.atlassian.com/git/tutorials/what-is-version-control. (Visited 18 May 2022).

[10] Michael Ernst. Version control concepts and best practices. https://homes.cs.washington.edu/~mernst/advice/version-control.html, 2022. (Visited 18 May 2022).

[11] A. R. Hevner. The three cycle view of design science research. *Scandinavian Journal of Information Systems*, 19(2):87–92, 2007.

[12] Hickson Ian. Beneath the surface. http://ln.hixie.ch/?start=1115899732&count=1, 2005. (Visited 18 May 2022).

[13] Hunt Dean. Html5's drag and drop problem. https://www.inkling.com/blog/2013/10/html5s-drag-and-drop-problem, 2013. (Visited 18 May 2022).

[14] Claudéric Demers. Dndkit. https://dndkit.com, 2021. (Visited 18 May 2022).

[15] React. Type checking with proptypes. https://reactjs.org/docs/typechecking-with-proptypes.html, 2022. (Visited 18 May 2022).

[16] Django Software Foundation. The web framework for perfectionists with deadlines. https://www.djangoproject.com/. (Visited 14 May 2022).

[17] Christopher Trudeau. Building with Django REST Framework. https://realpython.com/lessons/building-drf-overview/. (Visited 15 May 2022).

[18] Holger Krekel and pytest-dev team. pytest: helps you write better programs. https://docs.pytest.org/en/7.1.x/. (Visited 14 May 2022).

[19] Andreas Pelme and contributors. pytest-django Documentation. https://pytest-django.readthedocs.io/en/latest/. (Visited 14 May 2022).

<Group 47>

<ACLgen> Vision

Version <0.2>

Revision History

Date	Version	Description	Author
<24.01.2022>	<0.1>	<Basis is done, more details are needed >	<Sigmund, Karl, Ilona>
<25.03.2022>	<0.2>	<Revisioned product position statement, Provided more details>	< Sigmund, Karl, Ilona >

Table of Contents

Vision

1. Introduction

1.1 Purpose

This document provides insight on goals of the ACLgen project for the Bachelor thesis. The project consists of developing a web-based application for managing firewall access lists for Sikt. creating an application that allows the user to effortlessly store, sort and search images. The project stage necessary for the Bachelor thesis will be completed by the 3rd year Software Engineering students within the 20th of may 2022, but the application will be further developed by Sikt.

1.2 Definitions, Acronyms, and Abbreviations None yet

1.3 References

None yet

1.4 Overview

The vision document will contain information to best explain the different goals, parts and processes we wish to implement in the project and how we will go about completing the project to the best of our ability. The document is organized in 12 sections, each covering their own subject.

2. Positioning

2.1 Business Opportunity

FWbuilder is an application that is currently utilized by Sikt for managing firewall ACLs. FWbuilder is outdated and is no longer under development. Sikt is in need for a modern solution that maintains the core functionality.

2.2 Problem Statement

Problem of	FWbuilder, while being necessary and useful, is also outdated, error prone, not user-friendly. Application crashes due to bugs, which are no longer fixed. Application is run as a standalone instance and concurrent network configurations by different users may introduce errors.
Impact of which is	ACLs management and network configuration are tedious, and are performed by very few people who both have great competence in networks, and know how to use FWbuilder..
Successful solution would be	A web-based system with cloud storage, modern UI, intuitive workflow that maintains core functionality of ACLs configuration.

2.3 Product Position Statement

For	Network engineers at Sikt, and other authorized users with intermediate computer networking knowledge.
Who	Have the need to easily configure ACLs of the network.
The application	Lets the user configure access rules of the firewall, create objects and services (building blocks of access rules configuration), and obtain a filter file that can be transferred to the router/switch in question.
Compared to	Acomplishing same goals but in a tedious and not intuitive way.
The product	Will be easier and more pleasant to use, also providing an opportunity for broadening the user base.

3. Project goals

The goals are divided into efficiency-, result- and process goals. This is for making the goals more clear and defined.

3.1 Efficiency goals

We would like our product to:

1. Have a modern UI and development stack
2. Be user friendly and intuitive
3. Be easy to maintain, update and developed further by Sikt

3.2 Result goals

We would like to:

1. Start replacing the existing tool for ACL configuration, meaning develop an application that has some of the core features implemented.
2. Make our product as bug-free as possible.
3. Make our product as user-friendly as possible.
4. Make our product technologically up-to-date.
5. Make the code base easy to maintain and update.

3.3 Process goals

We would like to:

1. Research pros and cons of the existing tool, get a good understanding of the domain and user needs.
2. Create a solid application base, which will be further developed by developers at Sikt.
3. Focus on quality of the developed features, rather than quantity.
4. Improve and deepen our knowledge of computer networking and system development.

5. Expand and improve our skills in team-work, get familiar with working environment and culture that awaits us after graduation.

4. Stakeholder and User Descriptions

4.1 Market Demographics

The target audience are tech employees who have at least intermediate knowledge of computer networking and need to configure ACLs on the network. More precisely, the users are the network engineers at Sikt. With further development, user base can be expanded to other employees with at least intermediate knowledge of computer networking, and/or other users whom Sikt would authorize for application usage.

4.2 Stakeholder Summary

	Description	**Responsibilities**
Client	Sikt employees who play a role as supervisors, mentors.	- Ensures the requirements for our product are well defined - Provides knowledge and mentorship in computer networking and system development - Provides resources and guidance based on our need to research the current situation and the domain - Gives us feedback
Developers	Students at NTNU IDI writing their Bachelor thesis	- Ensures the requirements for the product are met - Required to ask client for guidance, clarifications, resources when needed - Required to schedule meetings with the client

4.3 User Summary

	Description	**Responsibilities**	**Stakeholder**
Basic user	Typically, a service center worker at Sikt who has an intermediate computer networking knowledge and has a need to occasionally make basic updates to the existing firewall configurations, due to customer requests.	• None, as given user group is out of scope of this project	End user (at the late stages of ACLgen development)

Experienced user	A Sikt employee that has not configured firewall ACLs vie graphical interface before and got a need to make firewall configurations. Has advanced computer networking knowledge, is proficient in the domain of networks that are maintained by Sikt.	• None, as given user group is out of scope of this project	End user (at the late stages of ACLgen development)
Professional user	A network engineer at Sikt that has used FWbuilder before and will be the pioneer in utilizing ACLgen for configuring firewall ACLs. Has advanced computer networking knowledge, is proficient in the domain of networks that are maintained by Sikt.	• Feedback • User tests	End user (the only one for the early versions of ACLgen)

4.4 User Environment

Our target user will:

1. Be a network engineer at Sikt.
2. Be able to create object and services (necessary building blocks of ACL configuration).
3. Be able to configure firewall ACLs.

4.6 User Profiles

This section is an overview of the different user profiles.

4.6.1 <Basic user>

Representative	Basic users
Description	The basic user has intermediate computer networking knowledge and performs updates to the existing firewall configurations using network objects and services that are already stored in the system.

Type	The basic user only uses the application occasionally, so the workflow and UI has to be intuitive and error prone. The basic users have limited knowledge in computer networking, so the handling of firewall access misconfiguration is important.
Involvement	The basic users are out of scope of this project.

4.6.2 <Experienced user>

Representative	Experienced users
Description	The experienced user has advanced computer networking knowledge, is proficient in the domain of networks that are maintained by Sikt. The experienced user performs a wide range of firewall configurations, updates/creates network objects and services.
Type	The experienced user uses application occasionally and has little to no prior experience in configuring ACLs via graphical interface, so the workflow and UI has to be intuitive and error prone.
Involvement	The experienced users are out of scope of this project.

4.6.3 <Professional user>

Representative	Professional users
Description	The professional user has advanced computer networking knowledge, is proficient in the domain of networks that are maintained by Sikt. The experienced user performs a wide range of firewall configurations, updates/creates network objects and services.
Type	The professional user uses application regularly, so the workflow and UI has to be efficient, comfortable and error prone. The professional user has prior experience in configuring ACLs via graphical interface (using FWbuilder) and can therefore provide useful feedback and requests regarding product's workflow and functionality. The professional user is the one to come up with requests for new features, expanded functionality, corrections etc.
Involvement	The professional user has a crucial role at the pre-release development. The involvement includes defining the initial workflow and UI, testing the system while at development, provide feedback, requests, corrections.

5. Product Risk Overview

This section is an overview of product assumptions and dependencies and risk analysis.

5.1 Assumptions and Dependencies

Requirements:

- Modern computer with Chrome browser.
- Access to Sikt's network.

5.2 Risk analysis

Problem	**Chance of happening**	**Impact**	**Total risk**
Cooperation issues between students • interpersonal issues between group members • different ambition levels • unfair workload Could lead to distress and slow down the development of the project.	3	7	21
Reasoning: Team members have experience working with each other on development projects. The team members chose to do this project together, based on prior experiences.			

Cooperation issues with other stakeholders • interpersonal issues • mismatch between expectations and results Could lead to distress and slow down the development of the project, or even lead to its termination.	5	8	40
Major inexperience • utilization of unfamiliar technology • Dunning-Kruger effect While minor and intermediate inexperience is to be expected, major inexperience could lead to a considerable slow down in the development project, severe system failures/vulnerabilities.	4	7	28
Reasoning: While the effect of inexperience can be a huge issue, the fact that we have a huge team at Sikt helping us out, severely limits the possibility of this happening.			
Mismanagement • insufficient comprehension of system/process requirements • team members working on overlapping tasks • critical tasks being overlooked • poor planning • poor reflection at the end of each sprint Could lead to unexpected increase in workload, wasting of resources, or even worse - unsustainable, unfinished or vulnerable products.	3	4	12
Reasoning: A small amount of mismanagement is to be expected during development, but our process of iteration and constantly producing working software minimizes the chances of this having a severe effect on our project. In addition to this, the team is small, which makes it easy to communicate.			
Considerable inefficiency of any of the team members • procrastination • struggling with routine while working from home • struggling with noisy surroundings while working from home Could lead to distress and slow down the development of the project.	5	7	35
Sickness/quarantine	5	5	25

6. Product Features

This section is an overview of the products's features that are in the scope of this project.

6.1 Separating rule sets with "repositories"

A feature to create a repository, which will contain all rules and objects needed to create rules. The purpose is to be able to create multiple repositories, for example a repository per customer or network location.

6.2 Network Objects

A feature to create objects. Rules are generally object based. Instead of inputting an IP address directly, a rule should point to an object, to be able to make changes across multiple rules that use the same object.

6.3 Services

A feature to create services. Same principle as with Network Objects. A rule points to a service.

6.4 Exporting to different formats (multiple device manufacturers/vendors)

A feature to export a set of rules to a device, that's understood by most firewall/network device manufacturers/vendors.

6.5 Folder system

The user can create a hierarchy of folders to make it easier to find specific rules or objects, and improve organization experience.

6.6 Difference between old and new changes

When the user makes changes in the system, they want to see what changes they are making before saving the new changes. This is helpful for larger changes.

6.7 Drag & Drop

A feature that allows the user to drag & drop objects into a rule set. This also includes dragging and dropping a rule to change its position.

6.8 Search

A feature to search for rules, and other related objects in the current repository.

6.9 Global Repository for pre-made objects

A feature where the user can pull in pre-made objects from a "Global Repository" into their current workspace/repository. Example use case is making a template for new networks.

6.10 Pretty User Interface

The user interface must be user-friendly.

6.11 User System

A user system that allows a user to login, with a permission system to restrict access to certain repositories.

6.12 Customer User System

A user system with a Customer type to allow customers to see their own networking rules.

6.13 Request Form

A feature where a request form can be sent by a customer to the engineers using the platform

7. Precedence and Priority

1. Separating rule sets with repositories
2. Create, update and delete Network Objects
3. Create, update and delete Service Objects
4. Create, update and delete Rules
5. Diffing
6. Drag & Drop
7. Export ACL to device
8. Search
9. Request Form

8. Non-functional requirements

8.1 Documentation

The application must be documented to prepare for further development of the project.

047

ACLgen
Pre-project plan

Version 1.2

Revision history

Date	Version	Description	Author
25. January, 2022	1.0	Started filling out the document	Sigmund, Karl
26. January, 2022	1.1	Writing about process and planning	Sigmund
28. January 2022	1.2	Rest of the document	Ilona, Karl, Sigmund

Table of contents

1. Goals and constraints

1.1. Why this project

This project was offered by Sikt, as Ilona already has close connections to Sikt due to being an employee. This project was discussed by the team members and was accepted because it would provide an interesting challenge, dealing with technologies we are interested in learning more from. The most important reason for choosing this project was the opportunity to develop software that would be actively used by Sikt in the future. Developing useful software remains the core motivation for this project.

1.2. Thesis, description and goals

The purpose of this project is to create a solution for generating access control lists (ACLgen). Sikt is developing a concept called CNaaS (Campus Network as a Service), and needs new software for managing rules for their firewalls. The rules are often replicated across devices and locations and should ideally support a wide range of hardware. Sikt has been very open about how we should develop these solutions, and left most of the decisions on us to figure out.

Sikt is looking for a tool they can continue to develop in-house. This means that our development efforts are focused on building software that would be easy to maintain by Sikt. A project planned for delivering a complete piece of software with the most complete set of features, would force us to make decisions where completing features is prioritized over maintainability. Due to the development only running for one semester, the maintainability implications are not likely to cause big enough issues for our development during this period. However, the solution would probably suffer greatly if Sikt would ever need to continue the development. For this reason we have agreed to focus on building iterative, maintainable software, at the expense of less important features.

We are starting this project with the assumption that the development of a solution for generating Access-control lists for Sikt is both a software Sikt wants, and one that will save them time. This assumption will not be tested, as the results of this test will not matter. The implications of changing the premise of the entire project will have severe effects on our ability to complete this project as our bachelor thesis. The development of this solution will continually test our assumptions with the goal of determining which solution will prove most useful to Sikt with the given time constraints. The planning of this project revolves around proving or disproving assumptions regarding how or why the software is developed. Our development team is basing these decisions on the assumption that neither Sikt or the development team know exactly what the best solution currently is, and that the majority of the requirements might change, based on the information we are able to gather. It is therefore natural for us to adopt an agile workflow to help us manage a plan that is supposed to undergo major change.

Sikt wants to start using the software as soon as possible in our development cycle. We have decided to adopt scrum during the second part of our development for our team to commit to delivering usable software to Sikt. This will allow us to benefit from the work of creating prototypes, gathering information and creating better measurements for which pace we can develop software, which will be done in the first part of this project. This development cycle will allow us to test whether our software is actually useful, and will allow us to incorporate changes and improvements during the

development of the software as we learn more about our misconceptions and the problem we are trying to solve. This is how we are committing to creating software that will allow Sikt to generate Access Control lists, as well as being a useful and maintainable tool in the future.

The team's ambition is to optimize our client's workflow in regard to their request (given assignment), by developing a new, up-to-date product that will assist in our client's day-to-day process in managing networking rules (ACLs - access-control lists). The result of the project will give long-term value to our client in the form of a time saving tool.

As the product is finalized, the goal is to achieve a measurable difference between our client's previous workflow versus the team's newly developed solution.

1.3 Purpose

Sikt wants a solution that allows them to generate ACLs (access-control lists) for managing their network rules for their CNaaS service (Campus Network as a Service). Their current solution will be replaced with a newly developed and more efficient solution, in order to save resources related to network management in the long term.

The main purpose of ACLgen can generally be described as sustainability. ACLgen will:

- save resources otherwise used on manual administration of access-control lists.
- reduce the rate of errors in network configuration.
- be operable by less experienced administrators with a more intuitive UI.

1.4 Constraints

This project has no material limitations, due to the development only needing hardware we already are in possession of. The development can happen anywhere, but is limited to working from home, as long as there is a threat due to the coronavirus. We are hoping to move development into Sikt's offices if possible during the development phase. During the course of development we will need access to hardware used to test the generated access lists. This will be supplied by Sikt when necessary.

2. Organisation

The only involved parties in this project are Sikt and us. NTNU is involved with Donn Morrison as supervisor.

3. Execution

3.1. Primary activities

We have two courses running in parallel this semester. During the start of the semester, we have to account for the fact that our efforts have to be split between these courses. The development process will also be affected, as the team members will have to split their efforts during the first part of the semester. Each part will run for about two months. To account for this, we have decided to split the development into two phases. While the two courses are running in parallel, we will mainly

gather necessary information which is vital to our decision making, and learn more about the technology we are going to use, by creating prototypes. We will also spend this time measuring how much time we spent doing different tasks. This allows us to improve our measurements, and create realistic plans for each iteration of the solution we are planning to develop. During this phase, we need to develop prototypes using the frameworks and technologies we are planning to use. The prototypes we have created during this phase will form the backbone of our development during the second phase. Following, is a list of which primary activities will be completed during the prototyping and testing part of the development. As a team, we have agreed to spend Thursday and Friday working primarily with this project. Each week we will discuss which work has been completed, who needs help, and what is keeping us from making progress. At this point, we will also decide who is best suited for the upcoming tasks, and which tasks should be done in pairs.

3.1.1. User story map

One of the first activities we wish to complete is to create a user story map. This will partly be done with the team over at Sikt. A User Story Map allows our team to put our thoughts and ideas on a board and create a vision for what the software could be. This can help us manage our ideas, and sort our ideas by what is most valuable for Sikt. This map will then be presented and discussed with people from Sikt, which will help us come up with new ideas, and prioritize which stories are most important for them. The map will be used and updated over the course of the entire development cycle, and should be a visual guide to understand what the software needs to do. The goal of this activity is to create shared understanding between the developers and Sikt, this is important, as it will make sure we are working on features which are the most important to Sikt.

After we have mapped out the core needs of our solution, we can start to make ideas for how we can implement the most important features, and which features we need more testing before we are ready to produce a prototype. We will have a meeting with Sikt, where we will discuss the story map on the 3rd of February. This meeting will help us create a plan for which parts of the system needs to be tested/prototyped first.

3.1.2. Documentation

There are a lot of design choices that are being made during this phase. A big part of development will be spent documenting design choices and gathering information. A large part of this will be documented only for the thesis report, but we hope to focus on documentation that would be useful for Sikt when we hand over the development of our software. Some of the Documents that will be produced during this period includes: User tests, Documents relating to system design, design choices, manuals and other required documents. We will write the documentation when we are either developing a prototype or doing tests, as this is the point where the information is clearest and it will help us avoid excessive amounts of work towards the end of the project.

3.1.3. Testing software and equipment

A core part of this project is to understand the limitations and possibilities of the systems we are going to be working with. These are primarily firewalls, but also Sikt's CNaaS. We will devote time to create tests which will showcase how these systems can be used. Which tests need to be done will be determined by which user stories are deemed most vital. We are planning to only test software and equipment which would be needed to start development when the active development phase

kicks in. A lot of user stories will depend on hardware and software which we currently have no experience with. To make sure we are not wasting our time, we will only test the solutions which are connected to highly prioritized stories, as these stories are most likely to be needed for our first sprint.

3.1.4. Prototyping

To get an effective start at development, it is important for us to create working prototypes which we can use as a baseline for development during our first sprints. Prototypes are used to test or showcase certain functionality which we have decided is necessary for completing one or more of our user stories. Initially we need to create system prototypes used as a foundation for our sprints. These include prototypes for User interfaces implemented in our preferred framework and the services required for generating ACL. Prototypes are used to test if an idea will work for our project, and will provide insight into how much time we will spend working on different frameworks and solutions.

3.1.5. Active development

During the second part of this semester, we will devote all available time to actively developing our software and producing working tools which the team at Sikt can make use of. To make the best possible use of our time, we have decided to adopt scrum to guide our work through this period. A big reason for choosing scum is to force us to deliver working software as quickly as possible.

3.2. Milestones.

Orientation
Dato: 10th January - 28th January
Week: 2-4
This period is reserved orienting the team about information relating to the project and our bachelor thesis. The team has used this period to orient themselves about the formal requirements and to create a plan for how to execute this project. During this period, we will complete our first meeting with Sikt and our supervisor, as well as prepare the pre-project plan.

Prototyping and testing
Dato: 31th January - 18th March
Week: 4-11
During the testing and prototyping phase, we will map out the most important features, and their hardware and software dependencies. We will test systems we are unfamiliar with, and create working prototypes, which will enable us to start building working software as soon as we move over to a scrum-process. During our testing and development phase, we will also be able to estimate how much time we spend on various tasks, which will help us create realistic estimates when it comes to planning how much work we should put in each sprint.

Active development
21. March - 20. May
Week: 12-20

Every two weeks during the active development we commit to delivering working software to Sikt, which they can use, and to provide feedback for how we can improve our software for the next sprint.

Sprint 1:
21. March - 1. April
week: 12 - 13
Sprint 1 is focused on bringing the first MVP to Sikt, and should be the most basic implementation of our system for generating ACL. The first sprint will set the stage for each consecutive sprint, and will decide what our focus needs to be for the next sprints.

The next sprints are left empty, as we need to complete the first sprint to figure out what needs to be done during the next sprints.

Sprint 2:
4th April - 16th April
week: 14 - 15

Sprint 3:
18th April - 29th April
week: 16 - 17

Sprint 4:
2nd May - 13th May
week: 18 - 19

Final week:
The final week will primarily be used to wrap-up the project and finish writing the bachelor report.

4. Follow-up and quality assurance

4.1 Quality assurance.

Our main method of ensuring quality is by choosing a workflow which forces the team to repeatedly produce working software which should be tested and used by our client. We have no guarantee that we will get it right on our first try. The team has committed to working only on features and solutions that are agreed upon by Sikt to be the most useful for them.

4.2 Reporting.

We will report to Donn Morrison (our supervisor), once every two weeks.

5. Risk Assessment

Problem	Chance of happening	Impact	Total risk
Cooperation issues between students ● interpersonal issues between group members ● different ambition levels ● unfair workload Could lead to distress and slow down the development of the project.	3	7	21
Reasoning: Team members have experience working with each other on development projects. The team members chose to do this project together, based on prior experiences.			
Cooperation issues with other stakeholders ● interpersonal issues ● mismatch between expectations and results Could lead to distress and slow down the development of the project, or even lead to its termination.	5	8	40
Major inexperience ● utilization of unfamiliar technology ● Dunning-Kruger effect While minor and intermediate inexperience is to be expected, major inexperience could lead to a considerable slow down in the development project, severe system failures/vulnerabilities.	4	7	28
Reasoning: While the effect of inexperience can be a huge issue, the fact that we have a huge team at Sikt helping us out, severely limits the possibility of this happening.			
Mismanagement ● insufficient comprehension of system/process requirements ● team members working on overlapping tasks ● critical tasks being overlooked	3	4	12

• poor planning • poor reflection at the end of each sprint Could lead to unexpected increase in workload, wasting of resources, or even worse - unsustainable, unfinished or vulnerable products.			
Reasoning: A small amount of mismanagement is to be expected during development, but our process of iteration and constantly producing working software minimizes the chances of this having a severe effect on our project. In addition to this, the team is small, which makes it easy to communicate.			
Considerable inefficiency of any of the team members • procrastination • struggling with routine while working from home • struggling with noisy surroundings while working from home Could lead to distress and slow down the development of the project.	5	7	35
Sickness/quarantine	5	5	25

The above mentioned risk analysis is applicable to the project in general, but not necessarily to separate project's stages/components. Critical stages/components during the development will undergo separate risk evaluation.

6. Attachments

6.1 Schedule

A Gantt-diagram is included as an attachment.

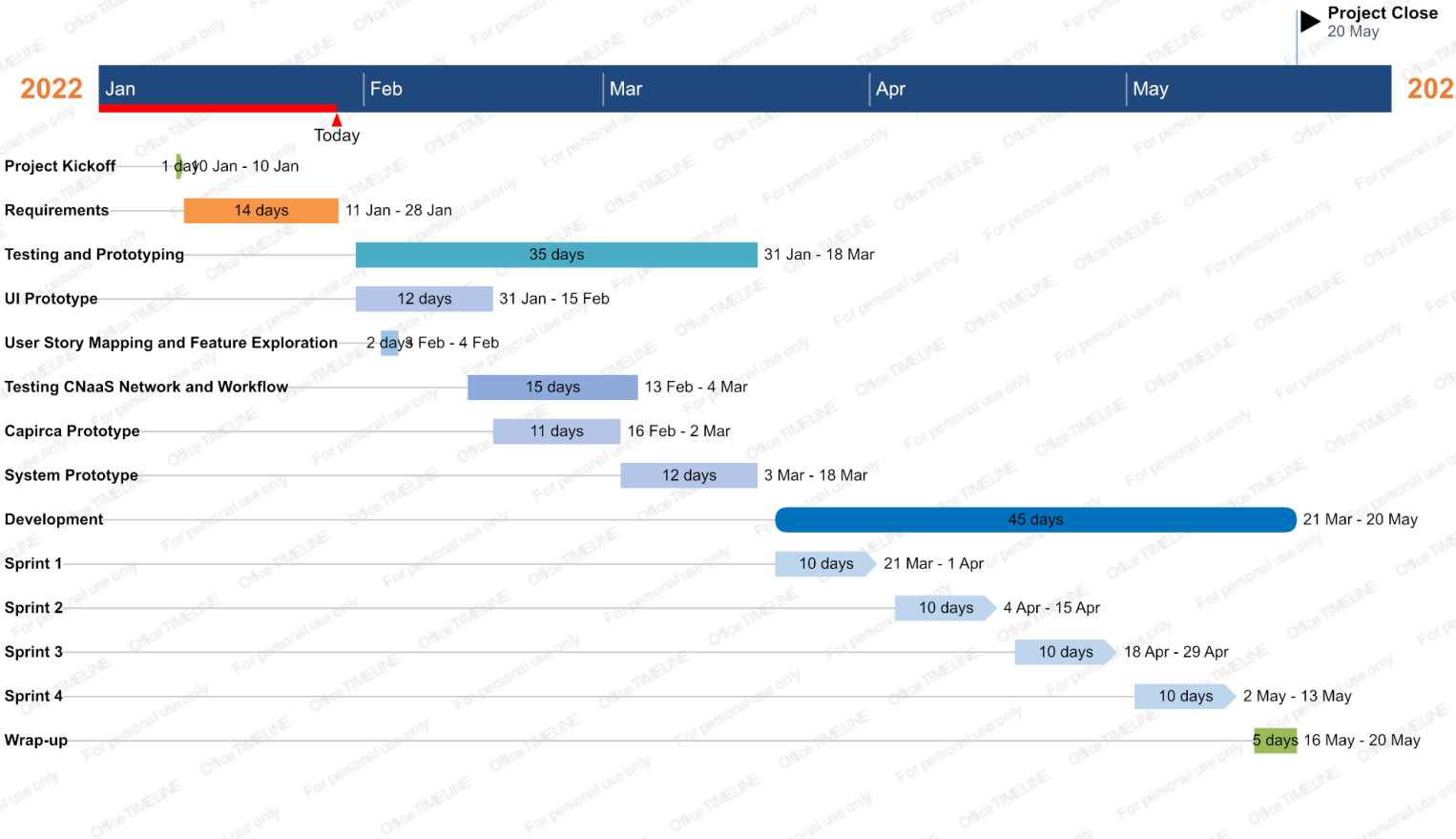
6.2 Collaboration Agreement

The Collaboration Agreement is included as an attachment.

6.3 Three Party Agreement

The Three Party Agreement is included as an attachment.

ACLgen Gantt-diagram



Collaboration Agreement

This Collaboration Agreement is related to the bachelor assignment no. 047 in IDATT2900.

The team consists of the following members:
Ilona Podliashanyk, Sigmund Granaas Sandring, Karl Klykken Labrador.

The agreement establishes goals, procedures and guidelines regarding our assignment. Each member of the team agrees to abide by the terms of this agreement to the best of their ability.

Goals

Purpose

The team's ambition is to optimize our client's workflow in regard to their request (given assignment), by developing a new, up-to-date product that will assist in our client's day-to-day process in managing networking rules (ACLs - access-control lists). The result of the project will give long-term value to our client in the form of a time saving tool.

As the product is finalized, the goal is to achieve a measurable difference between our client's previous workflow versus the team's newly developed solution.

Goals

1. Learn and reflect during the process.
2. Establish and keep up a good client-developer workflow.
3. Gain experience in new technologies
4. Work as a team to create a product we are proud of.

Roles

Each team member has equal responsibility. Responsibilities and tasks are planned on a weekly basis, in order to be flexible.

Procedures

A. *Meeting Invitations*
   Meetings with the team, where the client and/or the supervisor is represented, will have an attached Meeting Invitation, and a Meeting Minutes.

   Internally, the team will regularly meet for project discussions, but not necessarily make a Meeting Invitation.

B. *Notice of Absence & Other Events*
   If a team member cannot attend a meeting or a scheduled event, the member must give a notice in advance in a reasonable and timely manner.

C. *Document Management*
Project Documents, Meeting Invitations and Minutes will be stored in a Teams channel, where our client and supervisor has access.

Other documents will be stored in a shared disk space on Google Drive.

For version control and code storage, the team will use the client's preferred method of version control.

D. *Hand-ins, group tasks, milestones*
The team will to the best of their ability make sure to meet important deadlines related to the project, and other group related mandatory deliveries.

Each team member should push code to the remote repository at the end of a work day.

Interaction

A. *Internal Meetings/Collaboration & Preparations*
The team coordinates internal meetings among themselves, and meet at chosen time and date and preferred location or digital platform.

For internal communication, the team will use a dedicated Discord server for the project, where team members can be available throughout the day.

B. *How to support each other*
Every team member pledges to be a team player and assist each other to progress towards reaching our goals in the project.

C. *Breach of agreement*
The team will to the best of their ability attempt to resolve internal team issues. If a team member, or team members are unable to resolve an issue, the team may seek advice from the supervisor if an issue is not resolved within a timely manner.