

Gjert Michael Torp Homb
Michael Cortes Birkeland
Christian Simoes Isnes
Erlend Husbyn

Design and development of Malware Repository with multi- user access and characteristics aggregation

Bachelor's project in IT-Operations and Information Security
Supervisor: Mohamed Abomhara
May 2021

Gjert Michael Torp Homb
Michael Cortes Birkeland
Christian Simoes Isnes
Erlend Husbyn

Design and development of Malware Repository with multi- user access and characteristics aggregation

Bachelor's project in IT-Operations and Information Security
Supervisor: Mohamed Abomhara
May 2021

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Dept. of Information Security and Communication Technology

Abstract

Title: Design and development of Malware Repository with multi- user access and characteristics aggregation
Date: 20.05.2021
Authors: Gjert Michael Torp Homb, Michael Cortes Birkeland, Christian Simoes Isnes and Erlend Husbyn

Supervisors: Mohamed Abomhara
Employer: NTNU Malware Lab
Contact Person: Andrii Shalaginov

Keywords: Malware, Repository, Development, MEAN, Full stack, Multi- user access
Pages: 73
Attachments: 3
Availability: Open
Studypoints: 22.5

Abstract:

The NTNU Malware Lab wished to develop a platform for analysis and storage of malware and legitimate programs where users can upload and download files, as well as viewing characteristics of these files. This project aims to create a platform where multiple users, depending on their authorization can upload files, see characteristics about these files from various sources, as well as downloading files. The platform was built as a Proof of Concept (PoC) where portability, modularity and simple maintenance was an area of focus. This makes the platform simple to set up on new servers, and easily transferrable to another environments if wanted. Additionally, the platform is built so that adding new sources of analysis is simple. As the main purpose of the platform is to be used for research and education, the data about each file needs to easily accessible and collaborative. Therefore, an API to let users automate external processing of file data is implemented. On the frontend, Feide has been implemented for access control. As NTNU uses Feide for their IT solutions, the already existing user database can be used in user access management and authorization.

Sammendrag

Tittel:	Design og utvikling av plattform for skadevare-oppbevaring med rollestyring og aggregering av kjennetegn
Dato:	20.05.2021
Deltagere:	Gjert Michael Torp Homb, Michael Cortes Birkeland, Christian Simoes Isnes og Erlend Husbyn
Veileder:	Mohamed Abomhara
Arbeidsgiver:	NTNU Malware Lab
Kontaktperson:	Andrii Shalaginov
Nøkkelord:	Skadevare, Database, Utvikling, MEAN, Full stack, Rollestyrt tilgang
Sideantall:	73
Antall vedlegg:	3
Tilgjengelighet:	Åpen
Studiepoeng:	22.5

Sammendrag:

NTNU Malware Lab ønsket å utvikle en plattform for analyse og lagring av skadevare og legitime programmer hvor brukere kunne laste opp og ned filer, samt se karakteristikker fra disse filene. Dette prosjektet gikk ut på å utvikle hele denne plattformen hvor flere brukere med forskjellige rettigheter har mulighet til å laste opp filer, se karakteristikker om disse filene fra forskjellige kilder, samt laste ned filer. Plattformen ble bygd som et Proof of Concept (PoC) hvor portabilitet, modularitet og enkel vedlikehold sto i sentrum. Dette gjør at produktet er lett å sette opp på nye servere og, kan lett flyttes til et annet miljø hvis det er ønskelig. I tillegg er systemet bygget for å lett tilføye flere analysekilder. Denne plattformen skal brukes til forskning og undervisning som betyr at dataene om hver fil må være lett tilgjengelig. I denne sammenheng har det blitt implemenert et API for å lett programmere seg mot plattformen, som lar brukeren automatisere prosessering av fildata. På frontend delen har Feide blitt implemenert til tilgangsstyring. Siden NTNU bruker dette på sine løsninger betyr det at man kan utnytte den allerede eksisterende brukerdatabasen til rollestyring og autorisering.

Preface

During the project there has been multiple people who have helped with the development and progress of the system and thesis. We would like to thank everyone who read through our bachelor thesis and helped with the writing, spelling, structure and general fixes.

First, we would like to express our deep appreciation to our contacts at NTNU Malware Lab and SOC, Andrii Shalaginov, Geir Olav Dyrkolbotn and Christoffer Vargtass Hallstensen for their help and insight into how to create and develop this system that they envisioned, and for providing us with this task.

Second, we also want to offer a special thanks to our supervisor Mohamed Abomhara for all the valuable feedback and constructive comments regarding academic writing and report structure.

Last but not least, we want to thank Håvard Johansen who has given us a lot of guidance during the development, and sparring throughout the project.

Contents

Abstract	iv
Sammendrag	vi
Preface	vii
Contents	ix
Figures	xiii
Tables	xv
Code Listings	xvii
Acronyms	xix
Glossary	xxi
1 Introduction	1
1.1 Problem description	1
1.2 Objectives and Defining the assignment	2
1.3 Motivation and purpose of the thesis	2
1.4 Target audience	3
1.5 Student's backgrounds and qualifications	3
1.6 Additional roles	4
1.7 Project process	4
1.7.1 Development method	4
1.7.2 Progress plan	5
1.7.3 Plan for status meeting and decisions	5
1.7.4 General tools used	5
1.8 Thesis structure	6
2 Requirements	7
2.1 Initial project description	7
2.2 Use-Case diagram	8
2.3 High level Use Cases	8
2.4 Misuse cases	12
2.5 Extended Misuse cases	13
2.6 Functional requirements	14
2.7 Non-functional requirements	15
2.8 External requirements	15
2.9 Secure system development	15
3 Deployment technology	17
3.1 Development stack	17

3.1.1	Chosen stack	17
3.1.2	Alternative stack	18
3.1.3	Critique of chosen stack	18
3.1.4	Frontend	18
3.1.5	Backend / Express	18
3.1.6	Database	18
3.2	Deployment	18
4	Implementation	21
4.1	Frontend	21
4.1.1	Login	22
4.1.2	Listing and viewing files	24
4.1.3	Admin panel	27
4.1.4	Authorization	29
4.1.5	404 and inaccessible pages	30
4.1.6	Uploading	31
4.1.7	Downloading	32
4.1.8	Favorite	33
4.1.9	Filtering	35
4.2	Backend	37
4.2.1	Authentication and Authorization	37
4.2.2	Protecting resources	42
4.2.3	API	44
4.2.4	Upload	48
4.2.5	File Analysis	49
4.2.6	Download	50
4.3	Database design	50
4.3.1	Users	51
4.3.2	Files	52
4.3.3	Uploads	53
4.3.4	Downloads	54
4.3.5	Secure traffic	55
4.4	Storage	55
4.4.1	Metadata	55
4.4.2	File-storage	55
4.5	Logging	56
4.5.1	Winston	57
4.5.2	Winston-daily-rotate	58
4.5.3	What to log	58
4.6	System requirements	58
4.6.1	Frontend	59
4.6.2	Backend	59
4.6.3	Database	59
4.6.4	Testing	60
5	Security and Legal aspects	63

5.1	Security	63
5.2	Secure storage of malicious executables	64
5.3	Legal aspects	64
5.3.1	Avoiding misuse	64
5.3.2	Copyright	65
5.4	Malware research ethics	65
6	Results and going forward	67
6.1	Final product	67
6.2	Choices made during the project	68
6.3	Critique of final product	69
7	Conclusion	71
7.1	Project assessment	71
7.2	Knowledge gained	71
7.3	Limitations and future work	72
7.4	Evaluation of the groups work	73
	Bibliography	75
A	Additional Material	79
A.1	Project agreement	80
A.2	Project description,	83
A.3	Group rules	84
A.4	Projectplan	85
A.5	Meeting Logs	94
A.5.1	With supervisor	94
A.5.2	With employer	98
B	Additional documentation	113
B.1	API Documentation	114
B.2	Worklog	125
C	Code examples	129

Figures

2.1	Use case diagram	8
2.2	Misuse cases	12
3.1	Deployment	19
4.1	Sitemap of the repositories frontend	22
4.2	The login page	23
4.3	Permission denied page	23
4.4	Main user page	24
4.5	The file list	24
4.6	Checkboxes	26
4.7	Pagination on the file list	26
4.8	Detailed file view	27
4.9	Admin panel user list	27
4.10	Edit user dialogues	28
4.11	Delete user dialogue	28
4.12	System logs in admin panel	29
4.13	404-page	30
4.14	Systems sequence diagram of file uploading	31
4.15	Upload status bar	32
4.16	Download multiple files	33
4.17	Download button on detailed view	33
4.18	Favorite button on detailed view	34
4.19	Favorites page	34
4.20	The filtering/search component	35
4.21	The URI crafted by the filtering component	35
4.22	SESSIONID-cookies set by Innsida	37
4.23	The data provided by Dataporten to setup authentication	38
4.24	Button to enable guest users on the Dataporten-dashboard	39
4.25	Overview of file-routes documented with SwaggerHub	45
4.26	Overview of user-routes documented with SwaggerHub	46
4.27	Example of a detailed view of a route shown in Swagger	47
4.28	Docker pipeline for uploading files	49
4.29	How the path for file storage is calculated	56

4.30 Example of info-log file	57
4.31 Example list of daily rotated logs	58
4.32 CPU usage of backend in percent	60
4.33 CPU usage of database in percent	61

Tables

2.1	Use-case: Log in with Feide → Authorize to relevant group	9
2.2	Use-case: Activate/deactivate users	9
2.3	Use-case: View/Download/Upload malware/goodware samples	9
2.4	Use-case: Label samples	10
2.5	Use-case: See user list	10
2.6	Use-case: Classify samples	10
2.7	Use-case: Display samples	10
2.8	Use-case: Display logs	11
2.9	Use-case: Favorite samples	11
2.10	Misuse case: Access admin user	13
2.11	Misuse-case: Download malware	13
4.1	DB structure for users	52
4.2	DB structure for files	53
4.3	DB structure for uploads	54
4.4	DB structure for downloads	55

Code Listings

3.1	NGINX routing/proxy configuration	20
4.1	File sorting algorithm	25
4.2	Definition of routes and guards	30
4.3	How the URL is crafted and set.	36
4.4	Directive getting query parameters from URL.	36
4.5	Function for getting the subjects a user is a member of.	41
4.6	Function for getting the subjects a user is a member of	41
4.7	Protecting resources	42
4.8	Examples of arguments accepted by the middleware	43
4.9	Implementation of authorization middleware	43
4.10	Filefilter removing files unauthorized for users.	43
4.11	Database structure for user objects	51
4.12	Database structure for file objects	52
4.13	Database structure for upload-status objects	53
4.14	Database structure for download objects	54
4.15	Calculate filepath for file storage	56
4.16	Winston errorlogger	57
C.1	The authorization middleware protecting the REST API	129
C.2	Passport strategy handling user-logins	131
C.3	Auth Service Angular	133
C.4	AuthGuard Angular	134
C.5	AdminGuard Angular	135
C.6	LoginGuard Angular	136
C.7	Filefilter to remove files user is unauthorized for	136

Acronyms

APK Android Package file. 7, 14

ELF Executable Linkable Format. 7, 14

JS JavaScript. 17, 18

Mach-O Mach object file format. 7, 14

MEAN MongoDB, Express.js, Angular, Node.js. 3, 17, 18, 68

PE32 Portable Executable 32bit. 7, 14

PoC proof of concept. 1, 2, 7, 25, 37, 67, 71, 72

SOC Security Operations Center. 65

SPA Single-page application. 18

URI Uniform Resource Identifier. xiii, 35

URL Uniform Resource Locator. 35

Glossary

- authentication** The process of verifying the identity of the client. 14
- authorization** The process of specifying access rights to resources. 14
- characteristics** Characteristics in this thesis is related to metadata of files.. 7
- dataset** Structured collection of data. 2
- goodware** Software that does what it's supposed to with relative ease and nothing else.. xv, 1, 2, 7, 9, 13, 15, 20, 56, 67, 71
- malware** Malware is the collective name for a number of malicious software variants, including viruses, ransomware and spyware.. xv, 1–3, 6–10, 13–15, 20, 21, 31, 50, 56, 63, 64, 67, 71
- runtime environment** The environment the application is executed within. 57
- static analysis** Analysis of the source code of a file without executing the file. 2, 49
- TLS** Transport Layer Security. The technology used to encrypt traffic between two endpoints. 19
- transport** A process responsible for transporting the logs to their specific location, e.g. a folder or remote directory. 57
- VirusTotal** An online repository with metadata of many different files [1]. 15

Chapter 1

Introduction

At NTNU, many students and researchers are working with education and research related to malware and goodware. This type of research and study requires some place to store malware and goodware samples in an efficient, structured and secure manner.

A malware is a software designed with the intent to cause damage, disrupt or gain access to a network or computer system[2]. Goodware, on the other hand, is software that does what it is supposed to do with no unexpected or malicious behaviour.¹ A multi-user malware and goodware repository refers to a storage system allowing multiple users with different roles such as researcher or students to interact with files and metadata of the files in the database. These roles allow for different functionality depending on who the user is.

- In this bachelor thesis, firstly, the need for upgraded malware and goodware storage capabilities were discussed with the NTNU Malware Lab.
- Secondly, research on a suitable development stack was discussed within the group.
- Thirdly, the development of a proof of concept (PoC) repository began, with weekly discussion and input from the NTNU Malware Lab. This led to a functioning PoC malware and goodware repository.

1.1 Problem description

The NTNU Malware Lab is an academic collaborative group aiming to increase knowledge about malware in public and private sectors.² Malware and goodware samples are important for research and studies at NTNU. However, the current capabilities for storing malware and goodware samples for research and education at NTNU lack structure and functionality. Therefore, the NTNU Malware Lab wants to develop a better way of storing and interacting with these samples and make collaboration between students and researchers more

¹<https://androidforums.com/threads/what-is-goodware.832075/1>

²<https://www.ntnu.edu/malwarelab>

efficient. As the NTNU Malware Lab is responsible for handing out this thesis, they will be referred to as the employer further down in the report.

The current capabilities consist of multiple datasets containing samples that have been built through out the years, including 400+ archives with about 12TB of samples. Due to the unstructured fashion of this way of storing samples, a new storage solution is wanted by the NTNU Malware Lab. The new structured and sophisticated storage system aims to make fetching samples for use in research and education more efficient, with central storage and by utilizing a graphical user interface.

1.2 Objectives and Defining the assignment

The overall goal of this project is to enhance the research capabilities as well as the operations support for the various groups and departments at NTNU. These groups include the Malware Lab, System Security Group, SOC, Norwegian Cyber Range, etc.

The main goals of this thesis are as following:

- To develop a working proof of concept multi-user malware and Goodware repository with main functionality such as upload/download, capability to analyze files and adding relevant information such as tags to a file.
- To improve the current unstructured and functionality-less system, making research and education within and among various research groups more efficient.
- To make the Multi-user malware and Goodware repository portable and easy to maintain.

To keep the task within a reasonable scope, some limitations are made. Firstly, the finished proof of concept repository shall only be able to perform static analysis on the samples with help from third-party tools such as PEframe and VirusTotal, but may be extended to include more tools in the future. The reasoning behind this is that creating a dynamic analysis platform is of such magnitude that it could be a Bachelor thesis on its own, and this thesis consists the development of a repository, not an analysis platform. Secondly there will not be a major focus on the design of the frontend, however user friendliness is important.

1.3 Motivation and purpose of the thesis

When choosing a task for this thesis, it was agreed that a thesis about malware and viruses would be interesting, and is something that would be exciting to learn more about. Three of the group members have studied malware in *IMT4116 - Reverse Engineering and Malware Analysis*, while the last member studied the course while writing the thesis.

The main purpose of this thesis is to create an opportunity for NTNU Malware Lab to upgrade the current malware storage system, with relevant input from students who has/is taking the malware class. This will lead to more efficient handling of malware samples.

1.4 Target audience

The main audience are users of the NTNU Malware Lab, students, researchers and teachers. Moreover, the report of the thesis is mostly targeting those who wish to gain deeper knowledge about the system and its components. As the target audience is mostly qualified, or at least semi-qualified within the subject, this thesis will be going deep into the specific technologies used and how to implement or how it was implemented. On the other hand the repository is targeting those who conduct malware related research and education at NTNU. Furthermore, as this is a project to make a usable piece of technology, there will be provided a how-to install/use of the tech was developed.

1.5 Student's backgrounds and qualifications

The group consists of four BITSEC students at the Norwegian University of Science and Technology in Gjøvik. During the course of this study, all members should have acquired the knowledge required to complete a project of this size. The members of the group have previously worked together on projects. Although the group inherits a good amount of knowledge from the years at NTNU, there will be topics and technologies that will require deeper study and research. A major part of this self study will be learning how the MEAN³ stack functions and how its different components communicate with each other.

Gjert Michael Torp Homb

Group leader: 23 years old studying BITSEC at NTNU Gjøvik. Have experience with malware and network analysis, and will mostly be implementing analysis and storage of the files uploaded. Working as a security analyst on the side.

Michael Cortes Birkeland

\LaTeX responsible: 21 years old, third year BITSEC student at NTNU in Gjøvik. Has knowledge in the fields of information security and development, and has a lot of experience in using \LaTeX , means that he will be flexible and able to work on multiple parts of the project.

³[https://en.wikipedia.org/w/index.php?title=MEAN_\(solution_stack\)&oldid=1017827543](https://en.wikipedia.org/w/index.php?title=MEAN_(solution_stack)&oldid=1017827543)

Erlend Husbyn

Developer/responsible for conflict resolution: 22 year old student, studying BITSEC in the third year. He has some programming experience, and should be able to contribute to most parts of the project. Currently he is working as a Teachers Assistant beside the Bachelors thesis.

Christian Isnes

Developer responsible for all code, and its quality: 23 years old studying BITSEC at NTNU Gjøvik. Experience with a wide range of programming languages and stacks. He will be able to contribute to all layers in the project. Working as a part-time security analyst at NCSC-NO [3].

1.6 Additional roles

Employer: NTNU Malware Lab
Contact person: Andrii Shalaginov
andrii.shalaginov@ntnu.no

Additional contacts: Geir Olav Dyrkolbotn
geir.dyrkolbotn@ntnu.no

Christoffer Vargtass Hallstensen
christoffer.hallstensen@ntnu.no

Supervisor: Mohamed Abomhara
mohamed.abomhara@ntnu.no

1.7 Project process

1.7.1 Development method

The development method chosen for this project is Kanban⁴. This is a method that all group members have previously experienced and worked with; and find it simple and efficient to use. Adding tasks and requirements from NTNU Malware Lab is also simple with Kanban. Further, this allows for greater freedom to choose when to work on these tasks, compared to e.g. Scrum.

Kanban is a lean development method where tasks are visualized and added to a Kanban board. From here, the members of the development team assigns

⁴[https://en.wikipedia.org/w/index.php?title=Kanban_\(development\)&oldid=1013939902](https://en.wikipedia.org/w/index.php?title=Kanban_(development)&oldid=1013939902)

themselves to a task and completes this within the specified time frame (If there is one). A Kanban board contains various columns saying something about the progress of the task. This makes for simple identification of what tasks is finished, under progress and yet to be started.

1.7.2 Progress plan

To ensure that the project is completed within the required time, a detailed plan was made. A Gantt-chart is created to make sure the group is able to deliver on time. The Gantt-chart is included in Appendix A. May 20. is the absolute deadline for handing in the report, and it should be more or less complete a few days in advance for proofreading. Project presentations will happen on the June 7.

1.7.3 Plan for status meeting and decisions

There were continuous communication and daily meetings between the group members, where decisions are taken. Once per week, the group had a status meeting where each member covered the status on what they are working with. This meeting was generally held on Mondays at 12:00, however this varied depending on the availability of the group members. During this meeting, the group also handled bigger decision-points. Also, the group agreed on a bi-weekly meeting with the supervisor for follow-ups and guidelines. The supervisor was also available in case there was a need for another meeting. During the course of the project there was also weekly status reports and question meetings with NTNU Malware Lab.

1.7.4 General tools used

Throughout the development of this project, various tools are used as listed below. The uses for these tools include storing data and code, monitoring the progress, and communication both within the group and with supervisor and employers.

- GitLab
 - GitLab is the groups chosen Git-repository manager. The reasoning behind choosing GitLab over other code-repository managers (such as: GitHub and BitBucket) is that it provides tools and applications to support the whole development life cycle. This includes CI pipelines, Docker registry and Kanban boards. While not all these applications will be integrated in this project, they're prepared to be integrated later on. To administrate git branches and easier visualize where every member of the group is working, the group is using GitKraken as a Git GUI.
- Trello

- For monitoring and structuring our tasks and progress Trello is used. Trello is a Kanban board manager that is easy to use and manage. It provides a timeline and issue tracking. Overall this is a helpful tool to give the group an understanding of the current progress and what tasks that need to be done.
- Overleaf
 - Overleaf is the groups preferred \LaTeX editor and the main text editing software. Overleaf also provides Git integration which makes creating backups on local machines easy if Overleaf were to go down for an unspecified amount of time.
- Teams
 - Teams is used for maintaining contact and communication with the groups supervisor and employers. A "team" is created between the group and the employers.
- Google Drive
 - Google drive is used to store and share files and documents internally in the group. Examples of documents stored here are meeting reports, figures and the groups work-log.

1.8 Thesis structure

The thesis is structured as following.

- **Chapter 1** introduces and describes the bachelor thesis, and provides the reader with an understanding of the problem area. Additionally it describes the group, as well as the closest people involved in the process.
- **Chapter 2** presents details about the requirements for the finished repository. This includes intended functionality in addition to various functional requirements.
- **Chapter 3** describes the various technologies that has been used in the project.
- **Chapter 4** describes how the technologies used are implemented.
- **Chapter 5** dives into some security related issues and legal aspects of handling large amounts of malware.
- **Chapter 6** describes the achieved results and the future of the repository.
- **Chapter 7** concludes the work that has been done during the course of the Bachelor.

Chapter 2

Requirements

This chapter details the main requirements of the project, as well as the Use case diagrams.

2.1 Initial project description

Identifying the needs and requirements is an integral part of this thesis and is critical to its success. The main benefits of the requirements gathering are to increase productivity and enhance the outcome.

The initial project description of the bachelor thesis outlined certain requirements, as well as some anticipated tasks (see Appendix A.2). While these were the overall indicative requirements for the project, additional features were discussed during the weekly status meetings with NTNU Malware Lab.

The requirements listed in the initial description were the following:

- Development of a repository that allows access to the most recent and relevant malware and goodware samples, to serve as a core component in research and education at NTNU.
- The finished platform should allow for collaboration from multiple directions: uploading and updating datasets, fetching malware samples under defined specifications and cyber threats intelligence.

Based on these requirements, a list of anticipated tasks were given to the group from NTNU Malware Lab. These were the following:

1. Design top-down framework covering: optimal files and characteristics storage, database design, “push” / “pull” API, user access and roles.
2. Define functional and non-functional requirements to handle ELF / PE32 / APK / Mach-O files (see Section 2.6 and Section 2.7).
3. Focus on availability and building up new sources of characteristics.
4. Proof of concept with the focus on: portability, modularity, and complexity of maintenance.

These tasks serve as the base of the malware repository project, in correlation with the initial requirements.

2.2 Use-Case diagram

To describe some of the intended functionality of the repository in a understandable and graphical manner, a Use-case diagram is utilized, as shown in Figure 2.1. The Use-Case diagram details the main actors, both users and systems, as well as the functionality and processes they should be able to perform and handle.

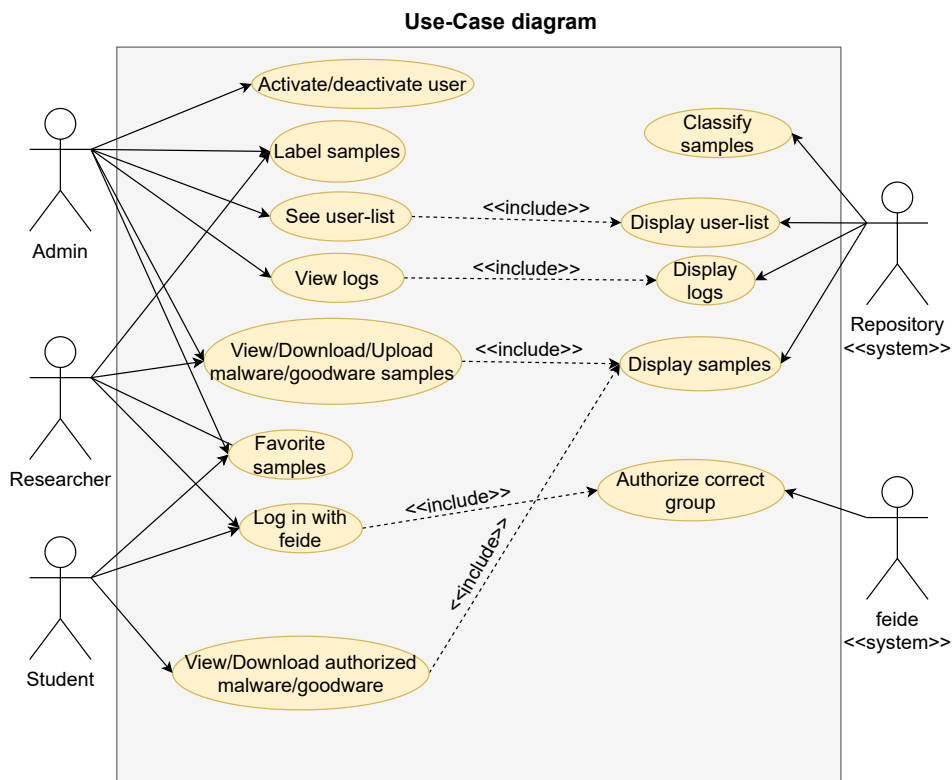


Figure 2.1: Use case diagram

2.3 High level Use Cases

High level Use-Cases based on the Use cases shown in Figure 2.1. These go further into detail than the diagram, and gives a high level understanding of the various interactions with user and system.

Log in with Feide → Authorize to relevant group	
Actor	Admin, researcher, student, Feide «system»
Purpose	Logging in users of the repository and authorizing them to their relevant group
Description	When a user accesses the repository they are met with a Feide login window. When logging in, the Feide system looks at the users profile and authorizes them to their relevant group, with adjoining privileges. Note: Users with admin privileges are hard-coded in a database, as admins and researchers likely belong to the Feide "employee" group.

Table 2.1: Use-case: Log in with Feide → Authorize to relevant group

Activate/deactivate users	
Actor	Admin
Purpose	Admins have the privilege of activating and deactivating users
Description	Users needs to have their account activated before being able to use the repository. This can happen either after a first time login, or before e.g. in a case where a lecturer adds several members of a class to a white list. This also works the other way, where an admin can deactivate a user. This could be necessary e.g. if a user uses downloaded malware with malicious purpose, and therefore needs to be excluded from the system.

Table 2.2: Use-case: Activate/deactivate users

View/Download/Upload malware/goodware samples	
Actor	Admin, researcher, student
Purpose	The users of the system should be able to view, download and upload samples
Description	Being able to view samples, download them, and upload new ones to the repository, is the main functionality of system. Which of these a user can perform is based on their authorization group. E.g. a student is unable to upload samples to the repository, and can also only download certain samples. Researchers and admins can both upload and download.

Table 2.3: Use-case: View/Download/Upload malware/goodware samples

Label samples	
Actor	Admin, researcher
Purpose	Giving labels to samples makes understanding the content easier
Description	An admin can write labels to a sample to get an easier understanding of what the malware contains. Additionally this is useful for the Reverse engineering and Malware analysis class as samples that have been used for assignments and exams can be labeled accordingly.

Table 2.4: Use-case: Label samples

See user list	
Actor	Admin
Purpose	View list of users for user management
Description	The admin has access to a panel listing all users currently having access to the system. This is helpful for either activating or deactivating the accounts of users.

Table 2.5: Use-case: See user list

Classify samples	
Actor	Repository «system»
Purpose	Reveals in-depth information and data about samples
Description	By using third-party tools like VirusTotal and Malware-Bazaar metadata about samples is gathered to give a broader perspective about their contents.

Table 2.6: Use-case: Classify samples

Display samples	
Actor	Repository «system»
Purpose	Give a simple user interface overview over samples on the system
Description	The backend displays the files stored on the system, in a structured manner, with the ability to show expanded info about samples.

Table 2.7: Use-case: Display samples

Display logs	
Actor	Repository «system»
Purpose	Displays logs in admin panel for quick status overview
Description	The admins are able to see relevant logs in the admin panel, without diving into the logs stored on the server.

Table 2.8: Use-case: Display logs

Favorite samples	
Actor	Student, researcher, admin
Purpose	Allows users to favorite samples to make finding them again easier
Description	If a user finds a sample that they find interesting, or know they will work a lot with, they may favorite it. When clicking favorite on a sample, that sample is added to that users favorite panel for later use/fetching.

Table 2.9: Use-case: Favorite samples

2.4 Misuse cases

The misuse case (Figure 2.2) builds upon and is the inverse of a use case diagram shown in Figure 2.1. It specifies the process of executing malicious actions against the system and the actors performing these.

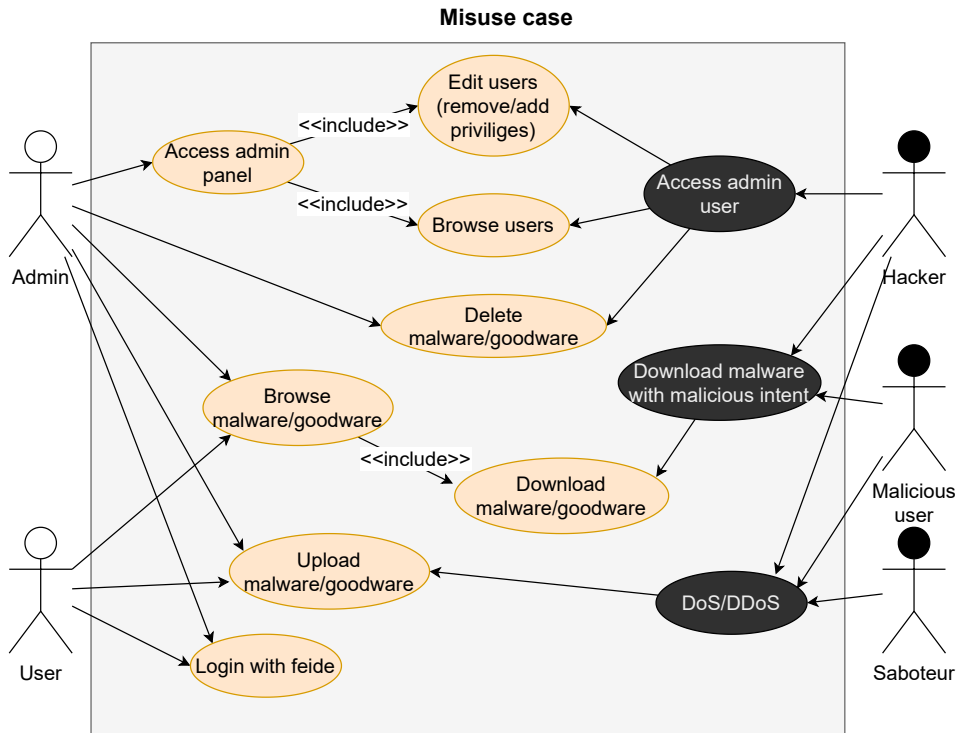


Figure 2.2: Misuse cases

2.5 Extended Misuse cases

Access admin user	
Actors	Hacker
Description	Hacker gains unauthorized access to an admin user. Gaining access to the admin user means that the hacker gains full access to all admin functions and the admin panel. The hacker could see users, change user authorization/privileges and delete malware / goodware from the database
Data (assets)	User data, malware and goodware
Attacks	Social engineering, hacking
Mitigation	Implementing authentication tokens, anti-tampering functionality, Feide
Requirements	Users are required to log in through Feide, and when logged in the admin's token expires after a couple hours

Table 2.10: Misuse case: Access admin user

Download malware with malicious intent	
Actors	Malicious user, hacker
Description	A user with authorization to download malware could download the malware and use them in a malicious manner as the system stores different ransomwares, worms, viruses and similar types of malware.
Data (assets)	malware, computers inside and outside of the NTNU network
Attacks	Malicious use of given files
Mitigation	Logging, authorization
Requirements	Every time a user downloads a malware/goodware sample from the server it gets logged on the admin panel meaning that tracking what files users have access to is easier. Making sure that only the users the admin trusts are given permission to download files.

Table 2.11: Misuse-case: Download malware

DoS / DDoS	
Actors	Saboteur, malicious user, hacker
Description	A user or saboteur performs a DoS/DDoS attack on the service. This could be a DDoS attack trying to overflow the current network bandwidth, or could be a user from the inside uploading .zip files containing Petabytes of files (e.g. Zip bomb). This misuse case covers any attack preventing the service to other users.
Data (Assets)	Database, backend server
Attacks	Denial of Service, Distributed Denial of Service
Mitigation	Docker containers for file uploading
Requirements	To prevent overflowing the uploading service, using a docker container should prevent this. There are few mitigations against a DDoS attack on application level, but they are less likely to happen as the repository is an internal service at NTNU.

To read more about zip bombs¹.

2.6 Functional requirements

The main functional requirements of the system starts with:

1. User login (authentication): User being authenticated via Feide (see Table 2.1).
2. Authorization: Access role to student/researcher and admin (see Table 2.1).

The system needs to be able to authorize users to their correct role. With each role, various functionality follows. A user with student privileges should be able to view and download malware samples with a limit to daily downloads. A user with researcher privileges can download and upload samples to the system as well as labeling the samples.

An admin has access to all previously mentioned functionality as well as the ability to activate and deactivate users (Table 2.2) and observe live logs being streamed from the backend. In addition, a lot of minor functionality should be realized. Users of the system should be able to favorite samples, and samples should be able to be tagged with relevant information. For the backend, logging should be implemented to assist in troubleshooting and investigating unwanted activity.

3. Functional requirements for ELF / PE32 / APK / Mach-O files:

These file types are executables made for the most common operating systems. ELF is made for Linux, PE32 for Windows, APK for Android and Mach-O for Apple based operating systems. These file

¹https://en.wikipedia.org/w/index.php?title=Zip_bomb&oldid=1016214544

types behave differently, meaning that certain precautions need to be taken when implementing support for these types of files. The repository be able to classify different filetypes and start analysis based on the classification (Refer to Table 2.6).

2.7 Non-functional requirements

In regards to non-functional requirements, there were also some guidelines. These mainly refer to the operability of the system. The non-functional requirements of the system are the following:

- **Portability:**
 - The system should be portable to allow for simple implementation on new infrastructures.
- **Modularity/Scalability:**
 - The system should be developed with focus on modularity and scalability, meaning further development and implementations on the system should be simple and uncomplicated.
- **Low complexity of maintenance:**
 - The system should be developed with focus on keeping the maintenance as simple as possible. This is an advantage as it requires less work to keep everything up to date.

With regards to other non-functional requirements such as response time and efficiency, these will be kept in mind during the development phase but is out of scope of this thesis.

2.8 External requirements

As a part of the requirement includes static analysis in the process of uploading samples, third party tools to aid in this sequence is used. While there are many tools out there that has this functionality, VirusTotal is probably the most known platform providing data on many different files, and is therefore used in the malware / goodware repository.

2.9 Secure system development

Since this repository is handling malicious software of all kinds, developing a secure system is crucial. If the system have a exploitable vulnerability, a malware uploaded to the system may, in the worst case, be able to infect the host machine and have full access to the network where the host machine resides.

Accordingly, secure development practices[4] should be utilized while developing this repository. When handling the malicious code, defense in depth [5] is also applied (see Section 4.2.5). This is done to provide several layers of defense in case of vulnerabilities in any of the tools used.

Chapter 3

Deployment technology

Deciding and defining the main building blocks is an important step in every software development process. This chapter describes the technologies behind the parts composing the foundation of the repository, with explanations of why these were chosen.

3.1 Development stack

A development stack, also called a software -or development stack is an ecosystem defining a set of languages, libraries and tools used for application development.¹ To develop a solution for a malware repository, a web development stack is needed which is also referred to as a full-stack. One significant benefit of using a predefined stack is streamlined development using a single language across the entire application.

3.1.1 Chosen stack

The MEAN stack is open source and primarily used to develop cloud-hosted applications due to its flexibility, scalability and extensibility[6]. The name MEAN is an acronym for the four technologies it's built with. MongoDB, Express, Angular and Node.js. The primary language is JavaScript (JS) which has been popular for frontends for a long time due to being flexible, dynamic and easy to use [7]. In recent years, it's started being an option for backend and database development, allowing developers to create applications with JS through the whole stack. By using JS throughout the stack, a smooth flow of information between all parts is ensured, as they have the same structure of objects and data. The stack is by all means not static, and any of the components can be swapped out for another technology if needed [8].

¹<https://www.techopedia.com/definition/27268/software-stack>

3.1.2 Alternative stack

Since the MEAN stack is not static, it is possible to swap out one or more components in the stack. A popular alternative is to change Angular with React². React is a JavaScript library for developing frontend applications, and have in recent times become widely popular.

3.1.3 Critique of chosen stack

Angular is framework designed for big applications. This language does have a steeper learning curve than other frontend libraries, which makes introduction to Angular harder. However, it seemed manageable and was therefore chosen.

3.1.4 Frontend

Angular is Google's Typescript-based frontend framework for developing single-page applications. By using a SPA, building a highly effective website is possible, as the whole page is not reloaded every time, only the section changing will be refreshed[9]. Frontends built on Angular is developed with TypeScript which is an extended version of JS with a few improvements such as variable type declarations [10].

3.1.5 Backend / Express

Node.js is an open source JS framework using asynchronous events to process multiple connections simultaneously[6]. It is the backbone of the MEAN stack and connects all the other parts together and passes communication. Express being the API Router providing endpoints serving different data from the database based on the request format.

3.1.6 Database

MongoDB is the database located at the bottom of the stack. It is a NoSQL database, which has many advantages such as being modular, efficient and able to handle large amounts of data quickly [6]. Additionally it is developer friendly, providing an advantage for this thesis. A NoSQL database is needed in this project as data from different analysis sources are unstructured and unsuitable for a SQL database.

3.2 Deployment

The application need to be deployed on servers to serve its purpose. These servers can be hosted on public clouds, or on dedicated local servers. This

²<https://reactjs.org/>

project is an internal service at NTNU, therefore, local hosting would be best suited. There is primarily two different deployment strategies to be considered[11]:

1. Serving the static files of the Angular application directly with NodeJS/-Express.
2. Use NGINX as a middleware for serving Angular files, and a proxy for the NodeJS/Express application.

The second strategy is being defined as the best practice in terms of performance and ease of use and maintenance. The reason is that NGINX is way faster at serving static raw content[11]. Node/Express is great at executing logic, but not optimized for static serving. NGINX also offers additional capabilities as gzip, load balancing and easier TLS integration.

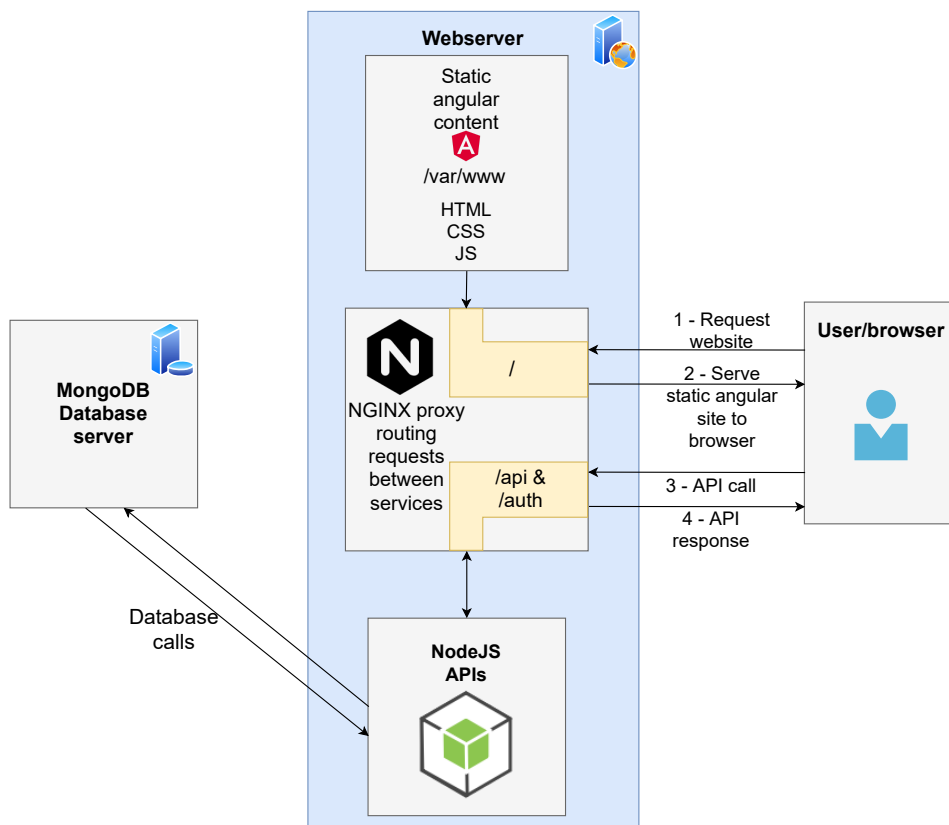


Figure 3.1: Deployment

```
1  location / {
2      # First attempt to serve request as file, then
3      # as directory, then fall back to displaying a 404.
4      try_files $uri /index.html =404;
5  }
6
7  location ~^(api|auth)/ {
8      proxy_pass http://localhost:3000;
9      proxy_http_version 1.1;
10 }
11
12 location /ws {
13     proxy_pass          http://localhost:3000;
14     proxy_http_version 1.1;
15     proxy_set_header   Upgrade $http_upgrade;
16     proxy_set_header   Connection "upgrade";
17     proxy_set_header   Host $host;
18 }
```

Code listing 3.1: NGINX routing/proxy configuration

As shown in Figure 3.1, the NGINX service acts as the first point of contact when accessing the Malware Goodware Repository. It will redirect the request to the correct service running on the server. When a user requests a website on the server on the root path '/', it will serve static content from the `/var/www/` folder, where the Angular frontend files are.

Code listing 3.1 reflects this logic with its routing definitions. That snippet from the NGINX- configuration file shows how requests to root/index will be routed to a static file `index.html` on line 1, which are the Angular static web-files. The block on line 7 defines that requests starting with either `/api` or `/auth` should be passed to a service running on port 3000, which is the Node.js/Express backend. The last block at line 12 will handle WebSocket requests, which also will be passed to the backend on port 3000.

Chapter 4

Implementation

During the course of the development process, various modules and components are used. As many of these work in similar ways, choosing the best fit for the repository is an important task. This chapter describes the various components of the repository from both the front- and backends perspective, and how each of these were included.

4.1 Frontend

Creating the top-down framework of the malware repository includes creating a frontend application with certain requirements, including being able to view and interact with files, having an admin panel with user editing and log view, an upload functionality and other functionality to make the application easier to use. The goal is to make this project a better alternative to what the NTNU Malware Lab already has, which is unstructured data in zipped archives on shared folders. Angular Material Design was chosen as the framework for implementing design elements on the frontend because is one of the most used UI design frameworks in modern web development. Additionally it has been developed by Google, which have developed Angular, leading to a great integration¹. The frontend is hosted as a web-application accessible through a users browser. Figure 4.1 depicts a basic sitemap with the various pages users may interact with depending on their authorization. The sitemap starts with the login page, and diverts to either the access denied page or the dashboard, which contains the main functionality depending on your authorization.

¹https://en.wikipedia.org/w/index.php?title=Material_Design&oldid=1019416314 Accessed: 2021-27-04 15:07

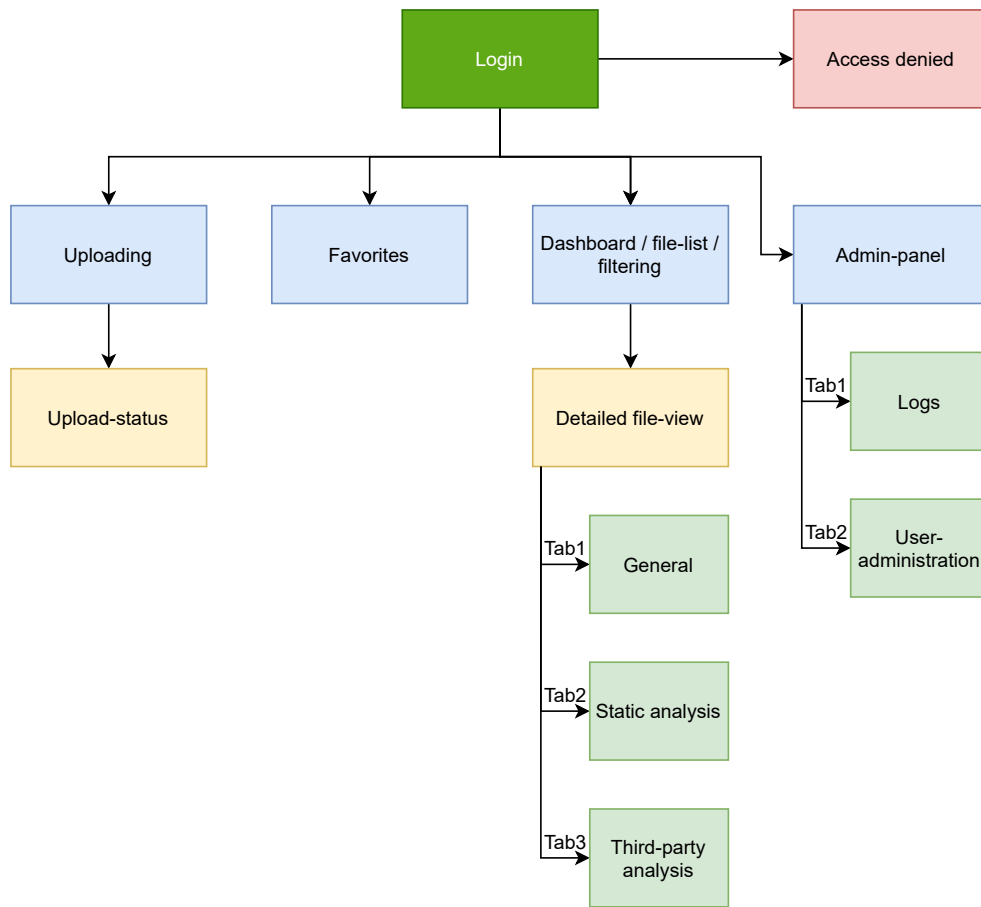


Figure 4.1: Sitemap of the repositories frontend

4.1.1 Login

When accessing the platform and a user is not logged in, a login screen will be shown. The login screen prompts the user to sign in with Feide, as shown in Figure 4.2. The user can then follow the steps and log in with their Feide credentials, that they share with other NTNU services like e.g. Blackboard.

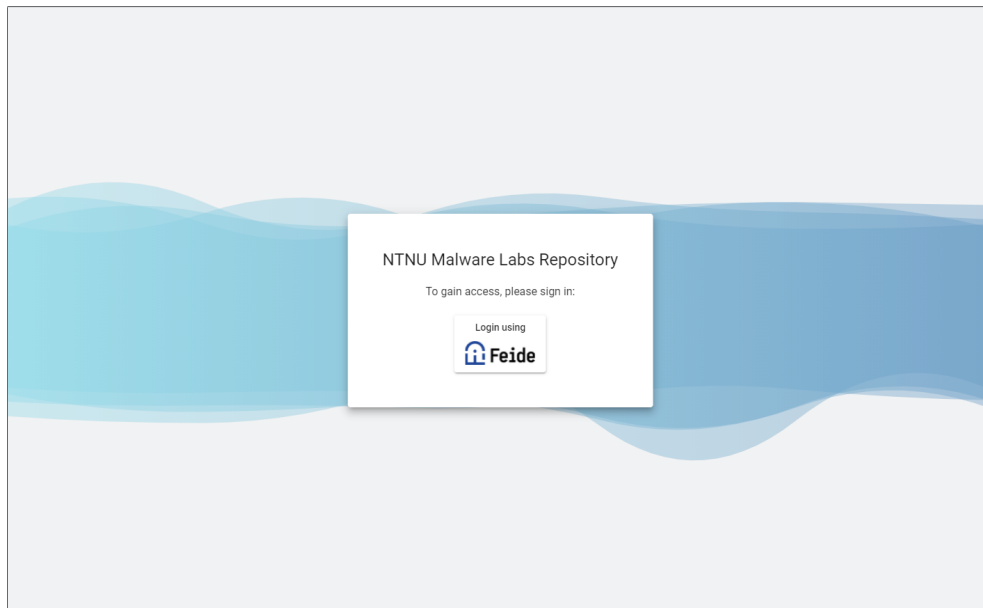


Figure 4.2: The login page

When authenticated with Feide, the server will check if the user is active. If that is not the case, the user will not be able to access anything and will have to contact an admin to be activated. This is default behaviour for new users, or if an admin have deactivated a user. As shown in Figure 4.3, a permission denied page will be shown to the user as the only resource they can access if their account is disabled or not yet activated.

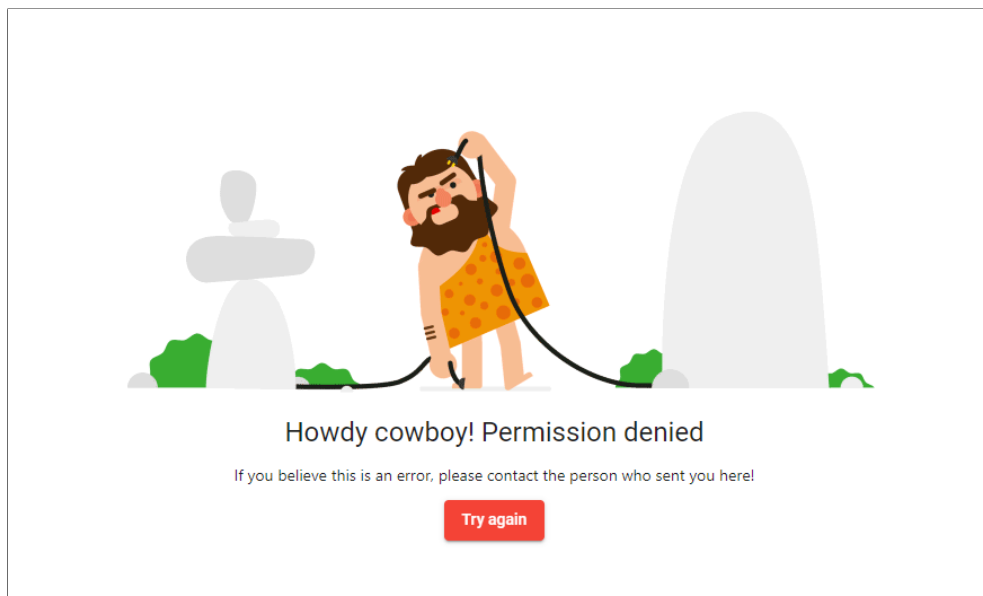


Figure 4.3: Permission denied page

In case the user have successfully logged in, the user will be greeted with the main dashboard as shown in Figure 4.4.

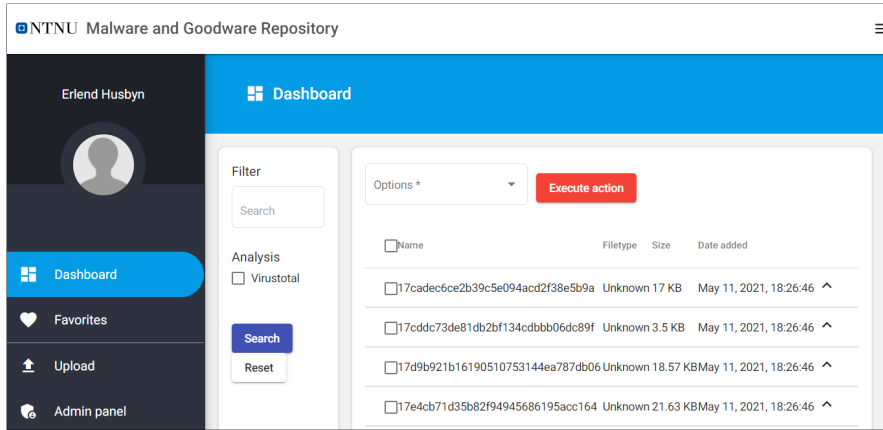


Figure 4.4: Main user page

4.1.2 Listing and viewing files

On the dashboard, there is a table containing a list of files stored in the repository as shown in Figure 4.5. It shows the filename, size and when the file was added to the database. Clicking on a row will expand it and show some more information and display a link to the detailed view of the file. It is possible to sort the rows, and select multiple files with checkboxes.

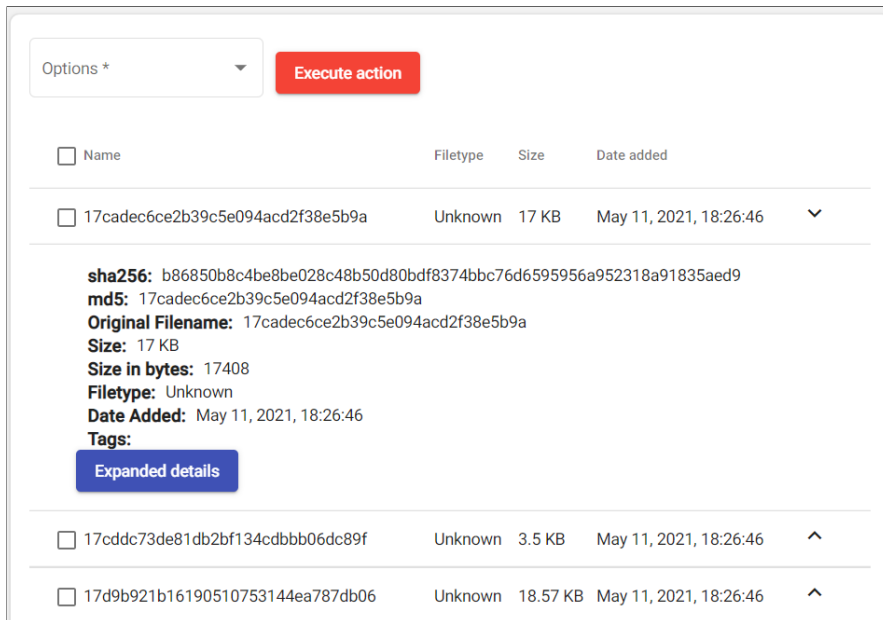


Figure 4.5: The file list

Sorting

The table can also be sorted alphabetically, ascending or descending. The table module supplied by Material Design have struggles with sorting words with different letters cases. To fix this, there was a need to change the sorting algorithm to account for lower- and higher cases when sorting as shown in Code listing 4.1. This was a known issue with the table-sorting package from Material Design. Finding a solution proved to be an easy task as this was already a reported issue on Github ².

```
1 this.dataSource.sortingDataAccessor = (item, property) => {  
2   // If there is no item[property] continue to next drawer in array  
3   if(!item[property]) {  
4     return this.sort.direction === "asc" ? '3' : '1'  
5   }  
6   // item[property] to lower case, which makes mat-sort able to sort it  
7   return '2' + item[property].toLocaleLowerCase()  
8 }
```

Code listing 4.1: File sorting algorithm

Without Code listing 4.1 the sorting will not work as mat-sort is case-sensitive. This code will go through the rows that are going to be sorted, checks if the users want to sort ascending ("asc"), and returns the same data in lower-case.

Checkboxes

The checkboxes enables the possibility to do a mass action, in this case of re-analysing, downloading, favorite and deleting selected files as shown in Figure 4.6. When clicking the button 'Execute action', the system sends an array containing the selected files to the correct API route according to the specified action. In this proof of concept repository, *reanalyze* and *download* are developed and working, as these was considered the most important functions of module

²<https://github.com/angular/components/issues/9205#issuecomment-423995549> Accessed: 2021-4-30 12:00

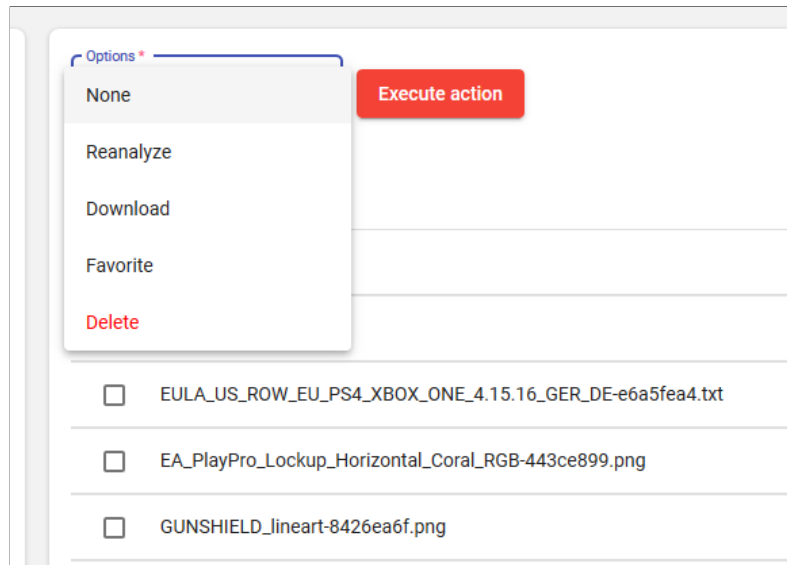


Figure 4.6: Checkboxes

Pagination

The table also supports pagination making traversing and loading the file entries easier as shown in Figure 4.7. When clicking on a new page it sends a new http request to the backend and only loads the files that is needed on the pagination page. This is faster as it now it only needs to load a defined amount instead of the whole database.



Figure 4.7: Pagination on the file list

File view

Clicking on the button "expanded details" in the expanded file on the file list, the user will be redirected to a page containing all information about this specific file, as shown in Figure 4.8. The detailed information includes most of the data stored on the database, such as: hashes, original filename, size, data added, file type, analysed information from 3rd party sites (VirusTotal) and static analysis (PEframe). Both the *sha256* and *md5* sum will for convenience be copied to the users clipboard if clicked on. It is also possible to download the files, add them to favorites or requesting a reanalysis of the file on the detailed view and the file list.

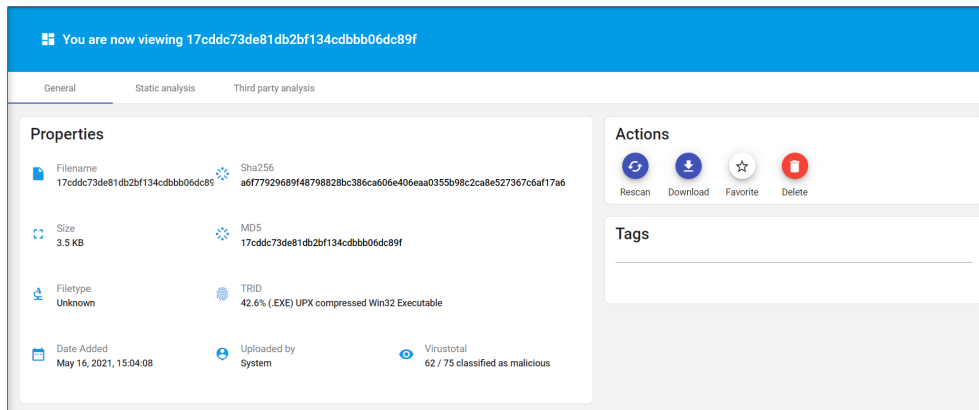


Figure 4.8: Detailed file view

Tagging

All files uploaded can be tagged. These tags are stored in an array in the database as mentioned in Section 4.3. This functionality was specifically requested by the employer as they wanted to be able to tag a file with for instance *exam* to hide it from students, or *private* to only be visible for the user who uploaded the file. These tags can be edited by admins, researchers, and the user who uploaded the file.

4.1.3 Admin panel

The admin panel (Figure 4.9) contains helpful tools for the users with admin privileges. This panel contains some features. First, a list over users where the admin has the ability to activate/deactivate and edit users. This is an important feature as it adds manual control over access to the repository. It is also an efficient way to deactivate a user in the case of malicious behaviour. Spotting this behaviour is related to the second feature of the admin panel which is a simple log view. This shows live logs in a clean format for the admins to see. The edit functionality allows the admins to change the role of a user.

User administration		System logs			
All users					
<input type="checkbox"/>	Name	Role	Last activity	Date Registered	Active
<input type="checkbox"/>	Michael Cortes Birkeland	Admin	Mar 29, 2021, 15:43:59	Mar 15, 2021	true
<input type="checkbox"/>	Gjert Michael Torp Homb	Admin	Mar 30, 2021, 14:30:33	Mar 15, 2021	true

Figure 4.9: Admin panel user list

Edit users

On the admin panels user list (Figure 4.10) the admins are able to edit a users role, status (active or not) and if they are able to upload. As requested by the NTNU Malware Lab, when editing a student there should be a checkbox to enable/disable uploading for specific students, as shown in Figure 4.10b. This is by default disabled.

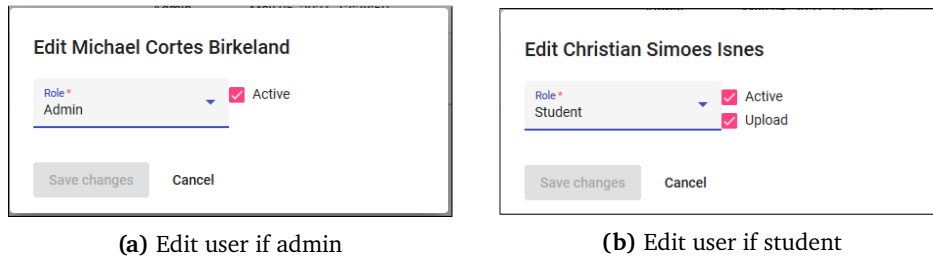


Figure 4.10: Edit user dialogues

Delete users

On the admin panel's user list the admins are also able to delete users. Clicking on the delete user selection users are prompted with a delete confirmation, as shown in Figure 4.11.

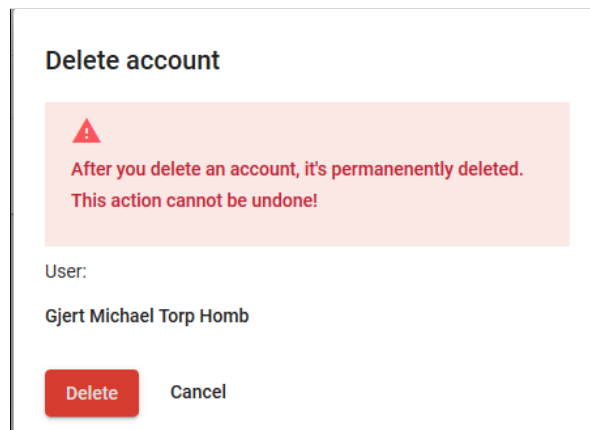


Figure 4.11: Delete user dialogue

Log list

The log-list on the frontend is establishing a websocket connection to the backend to continuously receive logs and display them to the user. The logs are created by Winston on the backend and creates a simple way to view the monitor the system as an admin. Figure 4.12 shows an example of logs in admin panel.

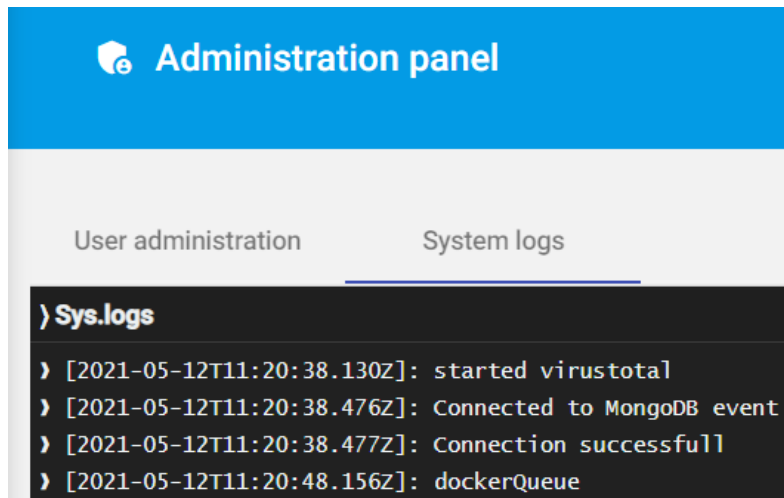


Figure 4.12: System logs in admin panel

4.1.4 Authorization

Handling authorization on the frontend with Angular is done by using a Route Guards, often known as AuthGuard. When a user want to navigate to a different location, it will trigger a call to the Router inside Angular which will handle the request. The Router will check with the AuthGuard whether or not the user should be allowed navigation to the requested route [12]. On the frontend, AuthGuard has been split into 3 different components, each handling a aspect of Route Guarding, these components are:

AuthGuard

The AuthGuard checks if the user that is trying to access the page is authenticated, and if not the user gets redirected to the login page. This guard is enabled on all pages other than the login page itself. And if a users "active" flag is set to false in the database, meaning the user is not allowed into the repository, the user will be redirected to a */accessdenied* page. The code for this is shown in Code listing C.4

AdminGuard

The AdminGuard checks if the user is an admin. If not, the user wont have access to the admin component. This guard protects the admin dashboard located on */admin* (Code listing C.5).

LoggedInGuard

The LoggedInGuard prevents users from accessing the `/login` page if the user already is logged in, and if the user is not logged in, the user stays on the `/login` page (Code listing C.6).

Route and guard definitions

The guards protecting the routes are defined on the frontend in `app-routing-module.ts`, which is keeping the definition of all routes. The routes are defined with their path, which component to render on that path, and which guard to protect this route.

```

1 const routes: Routes = [
2   { path: 'admin', component: AdminComponent, canActivate: [AuthGuard] },
3   { path: 'login', component: LoginComponent, canActivate: [LoggedInGuard] },
4   { path: 'favorites', component: FavoritesComponent, canActivate: [AuthGuard] },
5   { path: 'upload', component: FileUploadComponent, canActivate: [AuthGuard] },
6   { path: '', component: DashboardComponent, canActivate: [AuthGuard], pathMatch: 'full' },
7   { path: 'fileview/:sha256', component: FileViewComponent, canActivate: [AuthGuard] },
8   { path: 'upload/:id', component: UploadStatusComponent, canActivate: [AuthGuard] },
9   { path: 'accessdenied', component: notactiveComponent },
10  { path: '**', component: NotFoundComponent }
11 ];

```

Code listing 4.2: Definition of routes and guards

4.1.5 404 and inaccessible pages

Trying to access either inaccessible pages or pages that does not exist will redirect users to a 404 error page shown in Figure 4.13. The error page differentiates from the access denied page shown in Figure 4.3 by the fact that to get the 404 page the user is already using the website, meaning the account is activated.

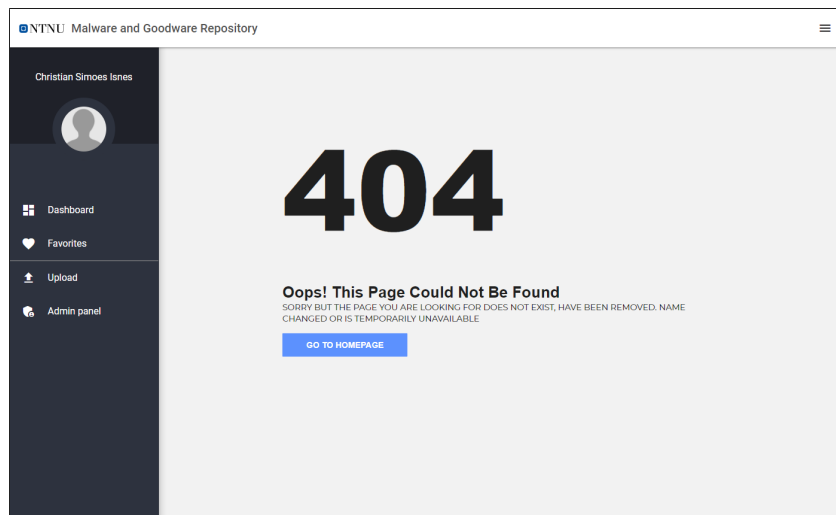


Figure 4.13: 404-page

4.1.6 Uploading

Uploading files is the single most important feature of the platform, to be able to get files into analysis. During the meetings with the employer, they requested that both admins and researchers should have an option to upload, in addition to students if they are given explicit permission through the user-administration tool on the admin-dashboard as seen in Section 4.1.3.

The upload service is accepting multiple files at once. They can either be a non compressed, or compressed files (zip, rar, tar, targz, gz). The file can be compressed without a password, or zipfiles encrypted with a common password used for malware samples such as "infected".

Once the files are sent to the backend via the upload-page, there is a need to see the progress of the files going through analysis after the upload have been tagged and given a unique ID. This could be achieved by the frontend sending a request to the backend on a set interval checking the status. However, this would create unnecessary traffic, and the events would not be entirely live. Instead, the frontend will establish a websocket connection to the backend server, and listening to new changes and showing them to the user in real-time. Figure 4.14 shows the information flow between front- and backend. The figure does not include cases where the user is not authenticated or other general errors.

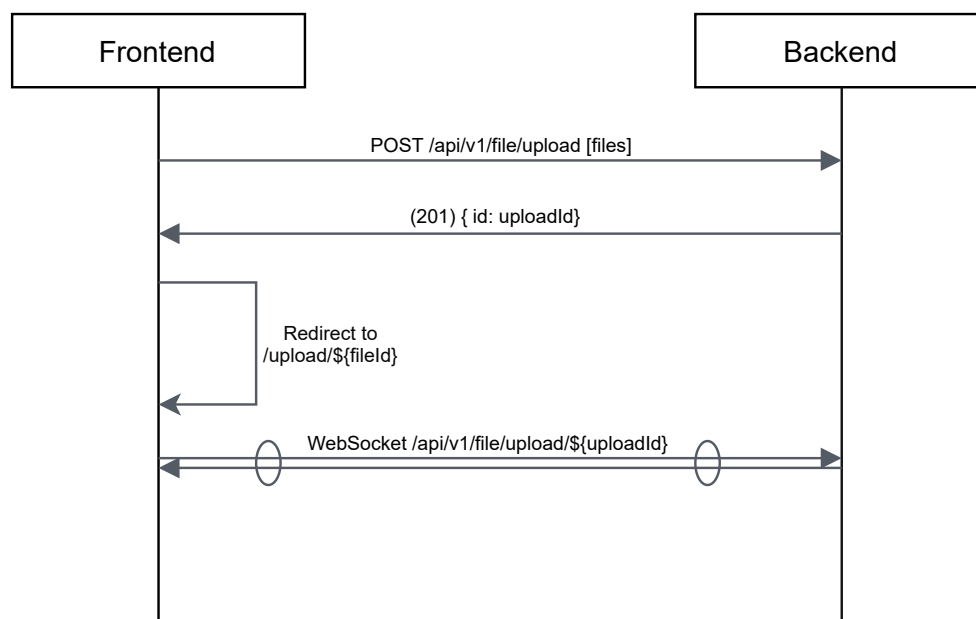


Figure 4.14: Systems sequence diagram of file uploading

Upload status

When uploading one or several files the user is redirected to a new page showing the status of the upload. The URL of this page consists of the fileId of the uploads object covered in Section 4.3.3. The status of the upload is made graphical with a progress bar shown in Figure 4.15. This gives the user a clear indication as to the progress of the upload.

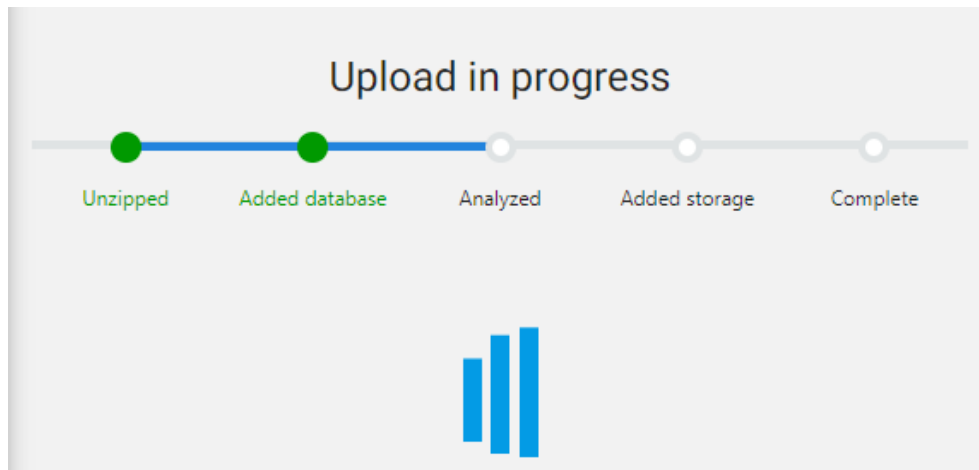


Figure 4.15: Upload status bar

4.1.7 Downloading

The user interface for downloading files allows for both single and multiple file download as shown in Figure 4.16. In the file list, a user can select multiple files and use the multi-function dropdown box to perform a bulk download which will ZIP all the requested files and download them on the users device.

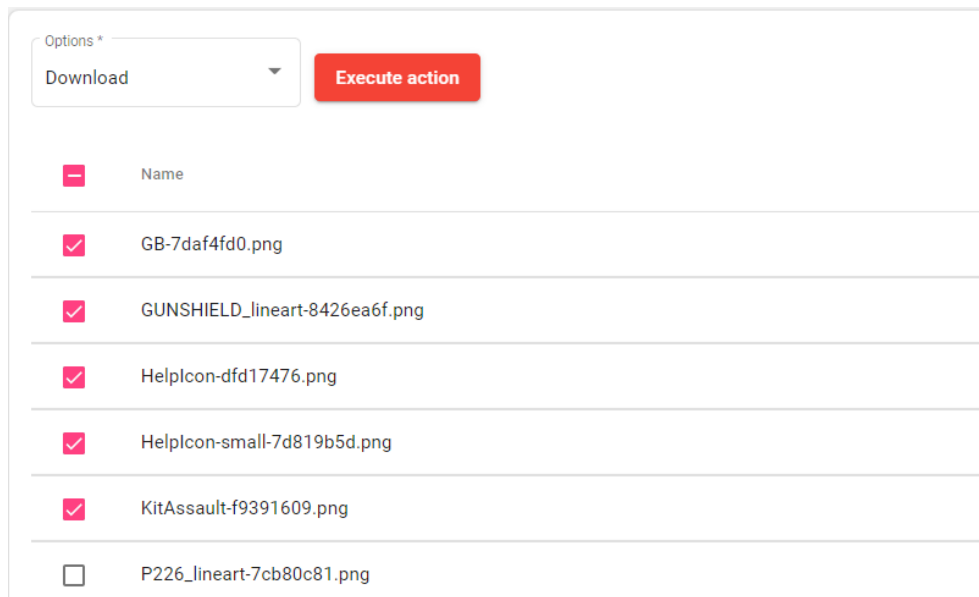


Figure 4.16: Download multiple files

By opening the detailed file view, a download button is present in the upper right corner. This will ZIP the requested file and download it to the users device.

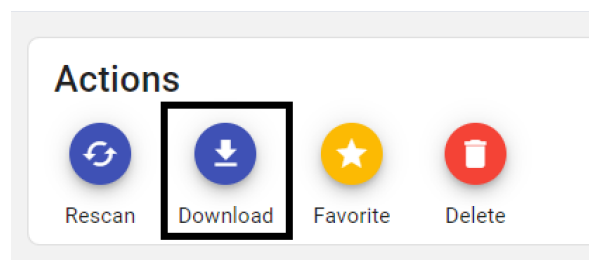


Figure 4.17: Download button on detailed view

4.1.8 Favorite

The functionality to favorite samples is present in the repository. Favoriting samples can be useful if the user want to save a particular file for later access, without needing to filter the whole database to find it again. During meetings with the employer, this was confirmed to be a wanted feature, and was implemented as a part of the repository. A sample can be favorited by either selecting multiple files in the file-list and bulk-adding them to favorites, or by doing it in the file view with clicking on 'favorite' in the actions-box.

The favorited files are accessible through the `/favorite` route, which is one of the primary features linked on the left sidebar of the repository for easy access.

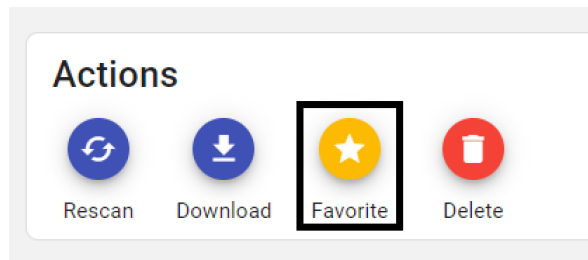


Figure 4.18: Favorite button on detailed view

The favorite page will list up all the files favorited by the user. A user can only see their own favorites, and not other users. The list is also been implemented with pagination in case there is a bigger quantity of files favorited.

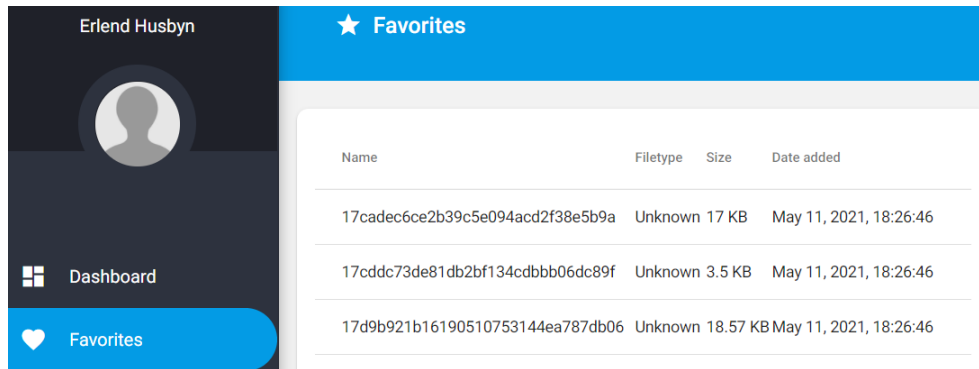


Figure 4.19: Favorites page

4.1.9 Filtering

Basic search and filtering functionality have been implemented to enable sorting. The component can accept text-search, and requirement-buttons for Virus-Total analysis.

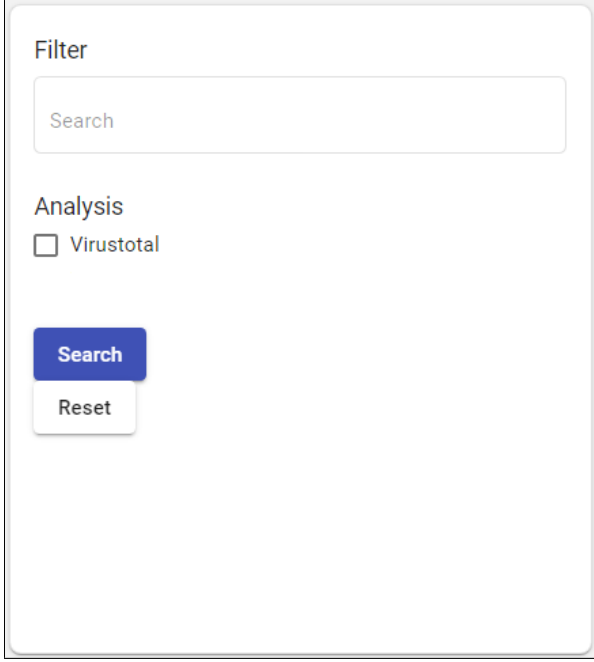


Figure 4.20: The filtering/search component

Once the search button has been pressed, it will craft a unique URI with query parameters matching the options from the filtering. This logic of this functionality is crafted in a way that the URI and the filter-component is two-way-binded, meaning if users change either, the other will reflect that change. This makes it possible to share a full URL with query parameters to another user, and they will receive the same results and the filtering component reflecting those filters to facilitate for further filtering based on the previous query.

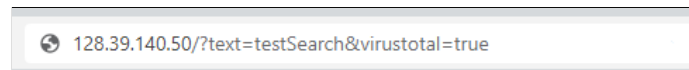


Figure 4.21: The URI crafted by the filtering component

The code in Code listing 4.3 shows how the URL is crafted through the filtering component after clicking "Search". The two filtering values, text and virustotal are assigned to the local variable *data* if they are defined, on line 3. On line 9, the router will navigate the user seamlessly to a route containing the query parameters set in *data*.

```

1 search(form: NgForm) {
2   // Create a JSON object containing the query params
3   this.data = {
4     text: (form.value.text ? form.value.text : undefined),
5     virustotal: (form.value.virustotal ? form.value.virustotal : undefined),
6   }
7   // Set the URL to accomodate for the new query parameters
8   this.router.navigate([''], { queryParams: JSON.parse(JSON.stringify(this.data)) }).then(()
9     => {
10    // Trigger refresh of file-list
11    this.fileservice.filesUpdated.next(true)
12  })
13 }

```

Code listing 4.3: How the URL is crafted and set.

An issue proved to be that undefined selections in the filtering-component (such as a non-selected checkbox for Virustotal), would add 'virustotal=undefined' as a parameter in the URI. This was solved as a part of line 9 in the code above, by taking the JSON object, stringifying it and parsing it back to a JSON object. This removed any undefined keys from the JSON object, and it could be set as queryParams in the router.

The two-way-binding of parameters was achieved by using a directive in Angular to get the query-parameters from the URL. Without the code in Code listing 4.4, the filtering component would not be aware of any parameters already set in the URL.

```

1 ngOnInit() {
2   // Get parameters from URI and save them in queryParams.{}
3   const queryParams = new URLSearchParams(location.search);
4
5   // If query includes form-key, patch it's value from query
6   if (queryParams.has(this.paramKey)) {
7     this.ngControl.control.patchValue(queryParams.get(this.paramKey));
8   }
9 }

```

Code listing 4.4: Directive getting query parameters from URL.

When loading the repository with query parameters set in the URL, they will be sent to the backend which will filter the results before sending them back to the user.

4.2 Backend

The purpose of the backend is to process requests from a user and handle all files in the repository, including their metadata. Some of these requests will require processing and analysis of files, in addition to communicating with the database. The backend will act as a connector between the user and the database and will resolve queries from the user.

4.2.1 Authentication and Authorization

Authentication refers to the process of making sure someone is who they appear to be[2]. Authorization on the the other hand is the process of assigning a user to their corresponding access level[2]. Developing a system where only authorized users are able to access the different components is a main priority. This is because the nature of this system is to handle files that can be harmful if they fall into the wrong hands.

1. Sessions and JWTs

When choosing an authentication method, two of the industry standards and most used method are JWT's (JSON Web Tokens) and Session based authentication. Both solutions are based around storing some data in the users browser, either a token (JWT) or a cookie (Session based). The main difference between these two methods of handling authentication is that the users data is stored on the server when using session based storage; while user data is stored in the browsers local-storage when using JWTs [13].

The decision on which method to implement was set by the argument of reusing the technology NTNU already have implemented in their platforms. For instance the service 'Innsida' is session-based where they are storing a 'JSESSIONID' in the cookie-store on the browser, and the users data on the web-server. Blackboard inherits this functionality. In addition, Uninett is supplying a session-based Node.js backend module, to show a proof of concept of how it can be implemented.

Name	Value	Domain	Path
LFR_SESSION_STATE_10	161494	innsida.ntn...	/
LFR_SESSION_STATE_2554	162020	innsida.ntn...	/
JSESSIONID	6B242DDCE9D03F...innsidaprod02	innsida.ntn...	/innsida-th...
JSESSIONID	7CD3D3095E079A...innsidaprod01	innsida.ntn...	/

Figure 4.22: SESSIONID-cookies set by Innsida

2. Methods

During the start of the development, a system for local authentication was developed. This method used a single username and password for authenticating the user with the data stored in a database. The employer later requested the

integration of Feide as an OAuth³ platform to authenticate the users, and researching Uninett's solutions showed this being possible. Therefore, focus was shifted away from the local authentication solution and over to implementing Feide instead.

3. Feide

Feide, one of the main identity management services used in the Norwegian education, is well integrated with NTNU's systems and is already in use on most platforms educationally integrated with NTNU such as Innsida and Blackboard. Students and employees at NTNU already have an account with Feide; Therefore, account-creation is not needed for users to access the platform. Uninett is running Dataporten.no which is a self-service platform for setting up applications authentication with Feide.

In order to register an application on Dataporten, users need to have an active Feide account. Once the application is registered, Dataporten will provide the necessary data for configuring authentication with OAuth2.0, which is ClientID, Client Secret and Authorization endpoint as shown on Figure 4.23.

The screenshot shows the 'Dataporten Dashboard' for a user named Christian Simoes Isnes. The main heading is 'OAuth 2.0 and OpenID Connect'. Below this, there are sections for 'OAuth Client credentials', 'OAuth Provider', and 'OpenID Connect'. The 'OAuth Client credentials' section includes fields for Client ID, Client Secret, and Redirect URI. The 'OAuth Provider' section includes fields for Authorization endpoint and Token endpoint. The 'OpenID Connect' section includes a note about the required scope and a link for auto-configuration. The 'Authorized Scopes' section lists various scopes like 'groups-other', 'userid', 'userinfo-name', etc.

Figure 4.23: The data provided by Dataporten to setup authentication

Two-factor authentication will add an extra layer of security, and will require users to provide two different authentication factors to verify themselves. A solution for two-factor is currently being launched on Feide, and can be activated from the backend when it's deployed, if needed. Additionally, spring 2021, Feide launched integration with ID-porten for authentication using MinID or BankID. This is a great way to confirming the authenticity of the users, as an alternative to traditional two-factor authentication with Feide.

³<https://oauth.net/2/>

The employer requested the possibility for users with no connection with NTNU to access the repository. Through the dashboard on Dataporten, guest-users can be activated as seen in Figure 4.24. Users can then log in with ordinary email and passwords through Feides IDP portal.

Miscellaneous login providers

<input type="checkbox"/>	HelseID [POC]
<input type="checkbox"/>	eIDAS [PILOT]
<input checked="" type="checkbox"/>	Feide guest users
<input type="checkbox"/>	Feide test users

Figure 4.24: Button to enable guest users on the Dataporten-dashboard

4. OAuth2.0

OAuth is the industry standard protocol for authorization. It is commonly used as a way for users to grant websites or applications access to their information without passing them their password. In this case, Feide supports OAuth2.0, and users will be logging into Feide, which then passes information to the repository about the user. Feide will handle the authentication, and the repository will handle authorization.

5. OpenID Connect

OpenID Connect is an identity layer sitting on top of the OAuth 2.0 protocol. It is supported by Feide as per their documentation [14] and makes it possible to obtain profile information about the end-user in a REST-like manner. User specific information the platform is requesting through the OpenID Connect endpoint is:

- What subjects a user is connected to at NTNU. (e.g. IMT4116), what role they have in that subject such as student and employee, and what faculty they're a part of.
- Full name of the user.
- Profile picture of the user for display purposes.

6. Passport.js

On the backend there is a need to protect the API routes from unauthenticated users. In the backend framework Node.js there are several ways to accomplish

this, one of which is by coding the whole module ourselves. The drawbacks of doing this is that future maintenance and refactoring of code will be more challenging as it is a custom module. To solve this problem, there is a module for Node.js called Passport ⁴ which is the most popular library for authentication in Node/Express based applications[15]. Passport.js is an authentication middleware for Node.js. Meaning it can be placed in the middle of a API route to process and manipulate the request before proceeding to the resource. A common use of this is to append data about the user to the request object.

7. Strategies

Passport.js are using authentication mechanisms called strategies[16] to define how the authentication should take place. As mentioned in Section 4.2.1, Uninett has a public GitHub profile with various repositories, one of which is *passport-dataporten*⁵. This package was depending on OAuth2.0 alone, and was able to authenticate users and show what subjects at NTNU they were a part of. However, as OpenID connect provides a better way with more endpoints to obtain user identity, it was decided that it would be a better and more scalable solution.

Further research showed that Uninett also have a repository called *passport-openid-connect*⁶ on GitHub. This repository successfully integrated OpenID connect, but did not grab which subjects at NTNU the user is a part of. To avoid the package being overwritten in the future by a possible update from the developer, it was manually added to the project without connection to the package manager.

In order to modify the package to suit the repositories needs, several changes had to be made.

First, it had to be able to grab a users subjects through the OpenID Connect endpoint. This was done through a call to the URL supplying user-groups. The function shown in Code listing 4.5 was added as a part the pipeline in the strategy provided by Uninett. It crafts a request-body on line 5, and sends the request on line 13. In return it will receive what groups the user is a part of, and return it on line 16 to the calling function.

⁴<http://www.passportjs.org/> Accessed: 2021-05-15 14:11

⁵<https://github.com/Uninett/passport-dataporten> Accessed: 2021-05-15 14:36

⁶<https://github.com/Uninett/passport-openid-connect> Accessed: 2021-05-15 14:38

```

1 OICStrategy.prototype.loadGroups = function() {
2   var that = this; // Workaround for accessing
3   'this' inside a subfunction, promise.
4   return new Promise(function(resolve, reject) {
5
6     var options = {
7       url: that.groupsUrl,
8       headers: {
9         'User-Agent': 'passport-dataporten',
10        'Authorization': 'Bearer ' + that.tokenSet.access_token
11      }
12    };
13    // console.log("Perforing OAuth 2.0 Request", options);
14    request(options, function (error, response, body) { // Send request to groups API
15      if (!error && response.statusCode == 200) {
16        var data = JSON.parse(body);
17        resolve(data); // Resolve with data
18      }
19      reject(error); // Reject if error
20    });
21  }).then(function(groups) {
22    that.groups = groups;
23    return groups;
24  });
25 };

```

Code listing 4.5: Function for getting the subjects a user is a member of.

The next thing that had to be done was to modify the serialization of users. When a user logs in, the account will be serialized and assigned to a session. It is necessary to create a database records of users using the repository, and to solve this it was added code to register new users in the database when they are logging in. This can be seen in Code listing 4.6 where data of new users will be sent to the database first. This way, all users on the repository can be administrated by an admin, and it is possible to store additional information on a user other than what is provided through Feide.

```

1 OICStrategy.serializeUser = function(user, cb) {
2   // Find the user in the database based on their feide_id
3   userService.findUser(user.data.sub)
4   .then(usr => {
5     // if user exist, serialize the user to a session
6     if (usr) {
7       cb(null, user.serialize());
8     }
9     // if the user does not exist, create it in the
10    // database and serialize the user to a session
11    else {
12      userService.createUser(user)
13      .then(data => {
14        cb(null, user.serialize());
15      })
16    }
17  })
18  .catch(err => {
19    this.fail(err)
20  })
21 }

```

Code listing 4.6: Function for getting the subjects a user is a member of

On line 3 in Code listing 4.6 the database will see if the user already exists. If the user exist, it will assign the user a session on line 7. If a user does not exist, it will create the user on line 12, and once the process is complete the user will

be assigned a session. The full code for the Passport strategy for authentication can be found in Appendix C and Code listing C.2.

4.2.2 Protecting resources

Protecting resources from unauthorized users is crucial to achieve a multi-user repository with access control. There are primarily two levels of authorization which needs to be implemented. The first layer is implemented as a part of the API route when users are requesting a resource, and the second layer is a part of the logic processing the files before sending them back to the user.

1. API middleware

A middleware is a function placed inside the definition of an API route. The purpose of this middleware is to see if a user got the necessary permissions to access that particular route. Different applications have different needs, so a middleware has to be tailored to suit the repositories needs. The middleware is programmed in such a way that it can restrict access for users based on both their role, and if they have certain tags set on their account.

As shown in Code listing 4.7, `exec()` is the main function in the middleware checking if certain functions are resolving as true. First, it is checking if all authorized users should be able to access the resource on line 8. Further, it's checking if the users-role is whitelisted for accessing the resource on line 10. Lastly, it's checking if the user has any tags set on their account which is whitelisted for that access on line 12. If none of these checks resolve with true, it will send a response to the user with HTTP code 401 as unauthorized.

The full authorization middleware can be found in Appendix C and Code listing C.1.

```

1      /**
2       * @desc Authorization-check for backend
3       * @params req, res, args (request, response and arguments)
4       * @res Will return true if authenticated/authorized, and send an immediate
5         401/403 if unauth.
6       */
7      function exec(){
8          if (allowAllAuthenticated()) { // Check if "everyone permitted"
9              is passed
10             return next()
11         } else if (isUserRoleAllowed(req.user.db.role)) { // Check if users groups
12             is in the routes groups
13             return next()
14         } else if (isUserTagAllowed(req.user.data.sub)) { // Check if users id
15             is in the routes whitelisted users
16             return next()
17         } else {
18             return res.status(401).json({ // User not
19                 authorized
20                 message: 'Unauthorized'
21             })
22         }
23     }
24 }

```

Code listing 4.7: Protecting resources

Code listing 4.8 shows the different arguments being passed to the `authorize` middleware. Line 1 would allow all authenticated users to access the API route. Line 2 would only allow admins to enter the route, and line 3 would allow admins and researchers, in addition to everyone else tagged with `studass` (short for student assistant).

```

1 authorize({ all: true })
2 authorize({ role: ["admin"] })
3 authorize({ role: ["admin", "researcher"], tags: ["studass"] })

```

Code listing 4.8: Examples of arguments accepted by the middleware

A live example of the middleware is shown in Code listing 4.9, where the `authorize` middleware has been implemented as a part of the `/upload` route on backend. The example shows how the route is allowed only for users being part of the admin or research role-group, or if they have the 'upload' tag set on their user. The latter is intended for students with explicit permission to upload files.

```

1  /**
2   * @desc Upload a file (the file itself)
3   * @route /api/v1/file/upload - No params, but file(s) as object
4   * @res 400 if post is misformed, 200 if file is uploaded
5   */
6  router.post(
7    "/upload",
8    authorize({ role: ["admin", "researcher"], tags: ["upload"] }),
9    multer,
10   async function (req, res, next) {
11     // code removed for display-purposes
12   }
13 );

```

Code listing 4.9: Implementation of authorization middleware

2. File-filtering

During meetings with the employer, it was clearly a wish to restrict some users from accessing various files. For instance, a student should not be able to see files containing the tag `exam`. This was solved by creating a function which will check all the files a user is accessing, and remove the ones the user is restricted to access.

```

1  function exec() {
2    // Check all files in the array
3    files.forEach(file => {
4      // Check if student and exam tag is set, and if file is private
5      if(isStudent(file) || isPrivate(file)) {
6        // Remove that file from array if user should not have access
7        files.splice(files.map(function(e){return
8          e.sha256}).indexOf(file.sha256), 1)
9      }
10   })
11   resolve(files)

```

Code listing 4.10: Filefilter removing files unauthorized for users.

In Code listing 4.10, the array of files will be looped through on line 3, and each file will be checked. If a file is tagged with `exam` and the user is a student,

or if the file is tagged with *private* and the user is not the one who uploaded it, the file will be removed from the array. At the end at line 10, the filtered array will be returned and eventually sent to the user. This way, users will not be able to access files marked with specific tags if their account is not authorized for them.

The full filefilter code can be found in Appendix C and Code listing C.7.

4.2.3 API

Communication between the frontend (browser) and the backend, is requiring a push/pull API. The API is hosted on the backend server on port 3000, and is behind the URI path */api*.

A challenge is updating the API over time, without causing incompatibility with older versions of frontends and pure API users. This has been solved by versioning the API. The current API is now version 1, and all API routes have a prefix of */api/v1/*. By implementing a versioning like this, new API routes can be implemented on */api/v2/* in the future, while still keeping the old version running. This ensures compatibility across versions.

All API routes are protected with a guard to limit access to only those who need them. For instance a route for deleting a user is only accessible by users having the role set to admin in the database. This is achieved with the middleware covered in Section 4.2.2

File endpoints

The file endpoints consists of routes related to file modification and querying. These routes makes the user able to upload, download and get metadata of files. In Figure 4.25 there is an overview of the API routes created.

File		All routes concerning files	∨
GET	/api/v1/file/	Gets a set of files from the database	↩
GET	/api/v1/file/{hash}	Get data for a specific file	↩
GET	/api/v1/file/{hash}/file	Downloads a file	↩
GET	/api/v1/file/count	Get the number of files in the database	↩
POST	/api/v1/file/upload	Uploads files to the repository	↩
POST	/api/v1/file/reanalyze	Triggers reanalysis of files	↩
POST	/api/v1/file/download	Download several files	↩
PUT	/api/v1/file/tags	Updating filetags	↩

Figure 4.25: Overview of file-routes documented with SwaggerHub

User endpoints

The user endpoints consists of routes related to user manipulation. The user may query information about themselves, and an admin may delete or edit a users with these endpoints. In Figure 4.26 there is an overview of the API routes created.

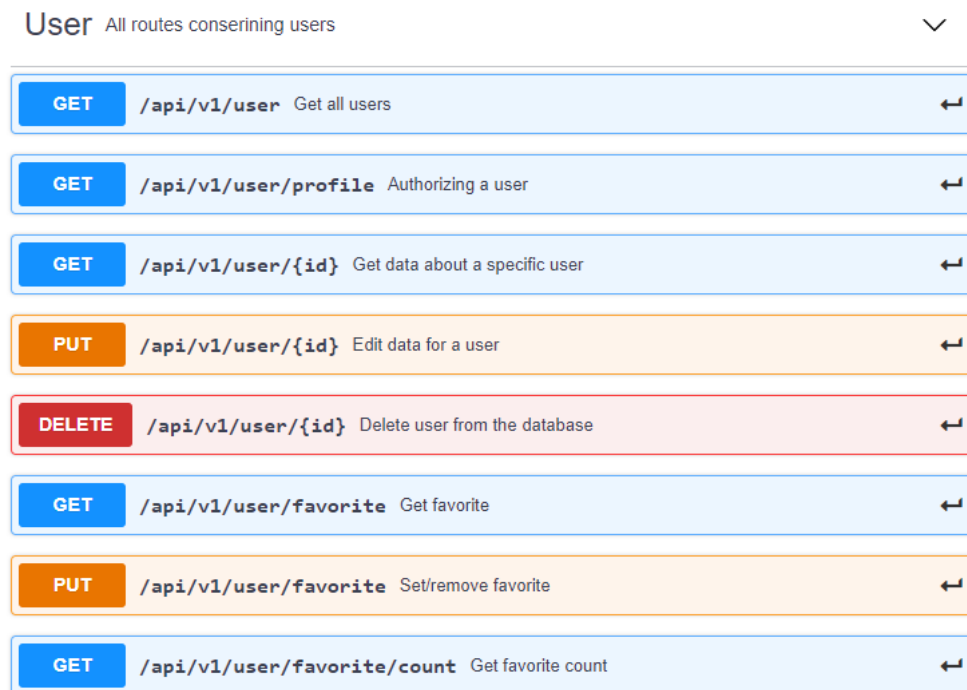


Figure 4.26: Overview of user-routes documented with SwaggerHub

Admin endpoints

The admin endpoints help the admin access the logs from the backend server. All of these endpoints are configured as websockets which will push a stream of logs once the connection is established. On connection, all logs from the current date will be pushed, along with new entries when they are made. The routes passing logs to the administration panel could be configured as a GET-route as well, but it would not be able to send logs live as the events occur which is a purpose and usecase of websockets.

Documentation

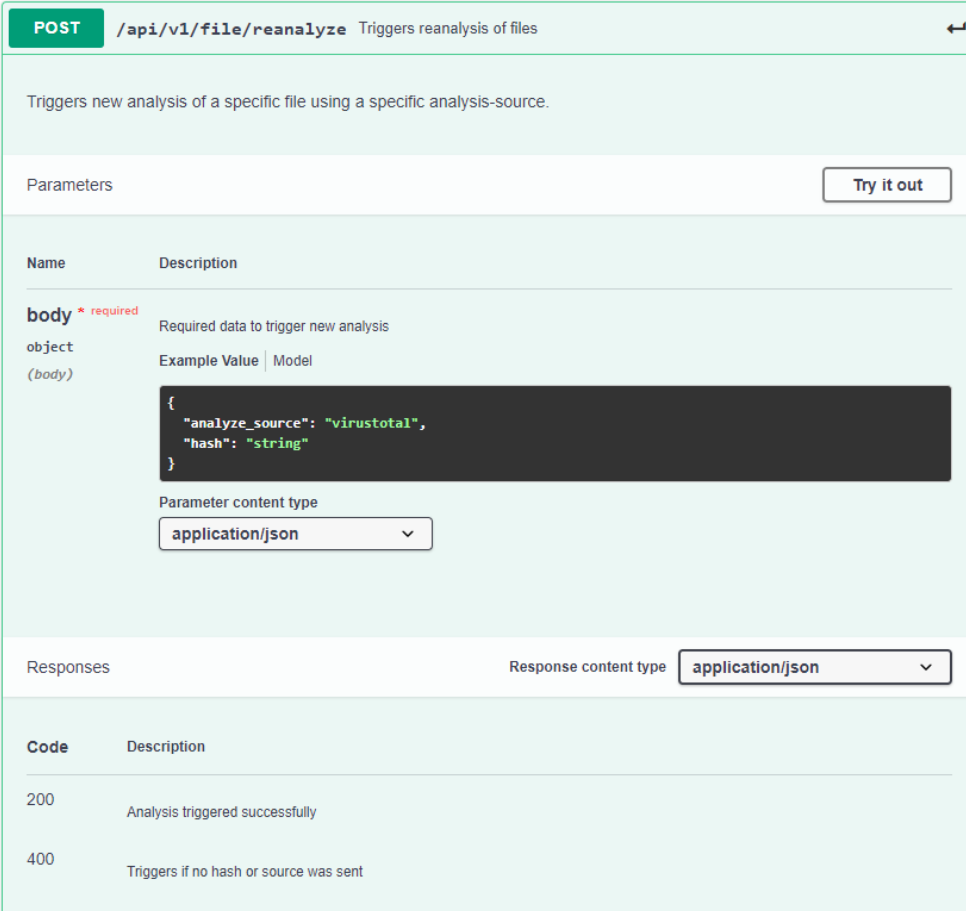
In order to effectively develop and integrate an API, it's crucial to document it well. Two API documentation tools which frequently shows up during research is a tool provided by Postman, and another tool called Swagger⁷.

In 2017, Postman released the results of a community-developer survey[17]. It showed that developers work a lot with APIs, as 70% of the community spends more than a quarter of their time working with APIs. Another key-highlight of the survey was that API documentation is critical, and could be better.

⁷<https://swagger.io/tools/swaggerhub/>

Both Postman and Swagger provides the same features and use the same OpenAPI syntax. In the end, Swagger was chosen as the API documenting tool as it is subjectively speaking, more visually pleasing. Since both services have implemented the same OpenAPI specification, it can be migrated to Postman in the future if needed.

Both Figure 4.25 and Figure 4.26 are products of the documentation written through Swagger. Each route are collapsible and once expanded they show additional information such as what parameters the route accepts, and what response the user can expect to get in return. An example is shown in Figure 4.27 where the route accepts a body-object with "analyze_source" and "hash" as arguments. In return it will either send HTTP response code 200, or 400.



The screenshot displays the Swagger UI for a POST endpoint: `/api/v1/file/reanalyze`. The description states: "Triggers new analysis of a specific file using a specific analysis-source." A "Try it out" button is visible. The "Parameters" section shows a required "body" object with the description "Required data to trigger new analysis". An example JSON value is shown:

```
{  "analyze_source": "virustotal",  "hash": "string"}
```

 The "Parameter content type" is set to "application/json". The "Responses" section shows two possible outcomes: a 200 response for "Analysis triggered successfully" and a 400 response for "Triggers if no hash or source was sent". The "Response content type" is also set to "application/json".

Figure 4.27: Example of a detailed view of a route shown in Swagger

The full documentation for the API can be found in Appendix B

4.2.4 Upload

The ability to upload samples to the repository is an important function. Having a pipeline that is fault-proof is therefore a significant part of the uploading process. When a user uploads a sample, the sample is put in a folder with a unique ID that will follow the file through the whole pipeline. Further, the file is checked for compression and one of two situations will occur:

1. If the file is compressed, a docker container unzips the file. And does so until there is no longer a compressed file present. In cases where there are more than 10 compressed files within each other, the file will be thrown out of the pipeline. This is done to prevent denial of the unzipping service by uploading a compressed file with an unreasonable amount of recursive zipped folders inside of it. The unzip service is also capable to bruteforce the compressed archive with its wordlist containing about 10 passwords. This wordlist consists of common passwords to encrypt malicious archives with.
2. The file is not compressed and is immediately transferred for analysis.

After unzipping, the analysis process begins. The file is checked whether it already exists in the repository database, and if it does it is thrown out of the pipeline. If it does not already exist, its hash, as well as general metadata of the file that don't need to come from analysis, is added to the database. Third-party analysis based on the hash is also started, as well as static analysis. After the analysis is finished the file(s) themselves are stored in the repository's file-storage.

Should a compressed file with several samples be uploaded, the pipeline is built to handle this. Such cases result in a queue where samples are analyzed one by one. Should the compressed file consist of more than 10 files, it will split the files into several folders to try to parallelize the analysis process.

Figure 4.28 shows the graphical sketch used for implementing the upload functionality.



Figure 4.28: Docker pipeline for uploading files

4.2.5 File Analysis

Files that are uploaded are sent to analysis to extract information from the file that users can search for and view. For this, the method static analysis is used. Although static analysis only looks at the content of the file without executing it, it is nearly impossible to ensure that the tools used for analyzing are without vulnerabilities and weaknesses. Therefore, all actions done on the

samples, unzipping and analysis, is done inside of a docker container. This method provides defense in depth⁸, as a sample designed to exploit a possible vulnerability in the tools used, would also need to escape the docker container to infect the system.

Per now, the tool used for static analysis is PEframe. This tool is primarily used to perform analysis on Portable Executables, but is also able to analyse generic suspicious files. Since the repository is made to be modifiable, it should be uncomplicated to add more analysis sources. FireEye FLARE team's CAPA tool⁹ was considered, but since the tool did not have any machine readable output, only graphical, it was not prioritized to format the output data of this tool.

When more analysis sources are added to the docker container, the container may grow quite big. This could affect the overall performance of the system, as less containers can be started to analyze uploaded files. A possible solution to this would be to create designated containers for specific types of files.

4.2.6 Download

When a user is downloading one or several files, the system will decrypt the queried samples and copy the files to an encrypted zip file. The password for the zip file is set by the user, but defaults to "infected" as this is standard practice to use for zip files that contains malware. A reason why a user would want to change the password is because, by experience, some antivirus engines are able to unzip and take action on files zipped with this password. Since the filenames of a zip file are not encrypted, the password is added to the zipped folder as the name of an empty file to let the user unzip the library even tho the user have forgot their password. This is done, as the purpose of the password protection is to obfuscate the file from antivirus engines and make it non-executable, and not protect it from users trying to open the zip.

4.3 Database design

For storing data about samples/files and users, it is structured in a NoSQL database run by MongoDB. The database content is divided into several collections where each collection consist of documents containing data. The following collections are present in the database:

- **Users:** Contains one document per user registered in the repository, with necessary information for administration.

⁸[https://en.wikipedia.org/w/index.php?title=Defense_in_depth_\(computing\)&oldid=999960956](https://en.wikipedia.org/w/index.php?title=Defense_in_depth_(computing)&oldid=999960956) Accessed: 2021-05-02 18:25

⁹<https://github.com/fireeye/capa>

- **Files:** One document per file/sample uploaded containing metadata about the file and data from various analysis-sources
- **Uploadstatus:** One document per upload. Contains the status of that current upload.
- **Download:** One document per Download. Contains the file(s) downloaded, along with the user downloading the file(s).

4.3.1 Users

Every user of the repository will get a unique object in the collection when registering, and an object will be stored as shown in Code listing 4.11. This object contains data about the user, with some of it coming from Feide, while other is system specific. The properties of this object is explained in further detail in Table 4.1.

```

1 {
2   "_id": {
3     "$oid": "6053167e03f5b13c040a46d3"
4   },
5   "active": true
6   "tags": [
7     "upload"
8   ],
9   "favorites": [
10    "8da1be1c6179a66381135e6c030ddd0b58b932411672b5015e884718a9c745b9",
11    "57bf4e824e079c3fbf72eec0d51007850888a016e26c99084bd8cf8aeb57b10a",
12    "4a9758db0222776ffbf5b1d3123161da1585d6a48a8d4450b069abd5591061a",
13    "72f266ddfe97003446b27ad01474b5a1737730ef4cf92b3f249eef63e8142772"
14  ],
15  "feide_id": "5a9108a2-118b-4e97-8b47-5d23f79cf39f",
16  "name": "Christian Simoes Isnes",
17  "role": "student",
18  "lastLogin": {
19    "$date": "2021-05-05T12:18:28.922Z"
20  },
21  "dateRegistered": {
22    "$date": "2021-03-18T08:59:42.083Z"
23  },
24  "__v": 0
25 }

```

Code listing 4.11: Database structure for user objects

<code>_id</code>	This is an ID given by MongoDB to that object for unique identification.
<code>active</code>	Boolean identifying if a user is active or not. New users are default set to false and need manual activation to be able to access the platform. Administrators have permissions to change this value in the admin-dashboard.
<code>tags</code>	Array containing various tags admins can assign to a user. Functionality is added to the "upload" tag, which grants the user access to the upload-page to upload samples.
<code>favorites</code>	Array containing sha256 hashes the user have favorited for later use.
<code>feide_id</code>	String of the unique feide_id of that particular user.
<code>name</code>	String with the full name of the user, received from Feide's OpenID connect endpoint.
<code>role</code>	String showing the role of the user. (Admin/Researcher/Student)
<code>lastLogin</code>	Date value holding the last time the user was active in the repository
<code>dateRegistered</code>	Date value holding exact time the user registered on the repository.

Table 4.1: DB structure for users

4.3.2 Files

Every file uploaded will get their own entry in the collection. In this step, each file entry is updated consecutively with the data from their analysis. Code listing 4.12 shows how the database entry for each file object is set up. Table 4.2 explains each property in further detail.

```

1 {
2   "_id": {
3     "$oid": "6092a3692fa7b325c3d67efe"
4   },
5   "pending_analysis": {
6     "virustotal": false,
7     "static_analysis": false
8   },
9   "filetype": "Unknown",
10  "uploaded_by": "Gjert Michael Torp Homb",
11  "tags": [],
12  "sha256": "d20d3c377f9a6cd80339dd457b5ced7c2bbdd62197d8ef99085ec104fd1f7709",
13  "md5": "7daf4fd0a1f6634bcf40721cc4e3b6b7",
14  "original_filename": "GB-7daf4fd0.png",
15  "size": {
16    "$numberLong": "1252"
17  },
18  "date_added": {
19    "$date": "2021-05-05T13:53:45.165Z"
20  },
21  "_v": 0,
22  "analyzed_info": []
23 }

```

Code listing 4.12: Database structure for file objects

_id	This is an ID given by MongoDB to that object for unique identification.
pending_analysis	Object containing enabled analysis-sources as subitems. The subitems are boolean values indicating if the analysis is still in progress, or not.
filetype	String containing the filetype.
uploaded_by	String containing the name of the user which uploaded the file.
tags	Array of strings containing the tags a user has assigned to the file. For instance "exam".
sha256	String containing the sha256 hash of the file
md5	String containing the md5 hash of the file
size	Long value containing the size of the file in bytes
date_added	Date value holding exact time the file was added to the database
analyzed_info	Array containing one object for each analysis-source holding it's respective data.

Table 4.2: DB structure for files

4.3.3 Uploads

Every upload will get their own entry in the collection. In this step, each entry is updated when a new step in the analysis process is reached. Code listing 4.13 shows how the database entry for each object is set up. Table 4.3 explains each property in further detail.

```

1 {
2   "_id": {
3     "$oid": "606eed0fad06c93d22d212b6"
4   },
5   "contains_compressed": false,
6   "unzipped": true,
7   "added_database": false,
8   "analyzed": false,
9   "analyze_queue": null,
10  "added_storage": false,
11  "splits": [],
12  "fileId": "yr4Bty",
13  "__v": 0
14 }

```

Code listing 4.13: Database structure for upload-status objects

<code>_id</code>	This is an ID given by MongoDB to that object for unique identification.
<code>contains_compressed</code>	Boolean identifying whether the file being uploaded contains other compressed files, or is compressed itself.
<code>unzipped</code>	Boolean identifying whether a compressed file has been unzipped. Note: If a single uncompressed file is uploaded, <code>unzipped</code> is still true.
<code>added_database</code>	Boolean identifying whether the data of the file(s) being uploaded has been added to the database.
<code>analyzed</code>	Boolean identifying whether the file(s) being uploaded has been analyzed.
<code>analyze_queue</code>	Int identifying which position in the analyze queue the file is.
<code>added_storage</code>	Boolean identifying whether the file(s) has been added to storage.
<code>splits</code>	If more than 10 files are uploaded, they are split into splits of 10 to parallelize the process. The splits row contains references to the other uploadstatus objects created.
<code>fileId</code>	A randomly generated ID for the uploadstatus object.

Table 4.3: DB structure for uploads

4.3.4 Downloads

Every download will get their own entry in the collection. Each entry will have a list of the files downloaded, along with the user downloading the files and the password for the archive. Code listing 4.14 shows how the database entry for each download object is set up. Table 4.4 explains each property in further detail.

```

1 {
2   {
3     "_id": {
4       "$oid": "608fe0932473a961a83b2b82"
5     },
6     "files": [
7       "e0b96c98a691938b04a4240de745c0db0d55e573570815b3e7b4ea5bb2b1a52a",
8       "57bf4e824e079c3fbf72eec0d51007850888a016e26c99084bd8cf8aeb57b10a"
9     ],
10    "downloader": "Christian Simoes Isnes",
11    "password": "infected",
12    "downloadTime": {
13      "$date": "2021-05-03T11:37:55.671Z"
14    },
15    "__v": 0
16  }

```

Code listing 4.14: Database structure for download objects

<code>_id</code>	This is an ID given by MongoDB to that object for unique identification.
<code>files</code>	Array containing the files that are downloaded
<code>downloader</code>	String containing the name of the user which initiated the download
<code>password</code>	String containing the password of the file archive that was downloaded
<code>downloadTime</code>	Date value holding exact time the file(s) was downloaded

Table 4.4: DB structure for downloads

4.3.5 Secure traffic

When deployed in a production environment, it is recommended to generate and sign a certificate to use MongoDB with TLS. This will ensure authenticity and confidentiality, as it would counteract eavesdropping of the traffic passing between the backend server and the database.

4.4 Storage

Two types of storage methods are used to store the samples and its metadata. This is done to ensure optimal storage of both the files and the characteristics/metadata of the files.

4.4.1 Metadata

Discussed in Section 4.3. The metadata of the files are stored in a MongoDB database. This makes it possible to index and search across all the metadata in the database. This metadata consists of: SHA256 sum, MD5 sum, original filename, size and who it is uploaded by.

4.4.2 File-storage

Storing of the encrypted files is done on the local filesystem the server is running on. A distributed network filesystem like glusterFS was considered, but since the repository is not disk-intensive, this was considered unnecessary and would make the system harder to maintain. The file-storage is used only when uploading, downloading and when files are sent to reanalysis.

Furthermore, to increase performance when storing files in the file-storage, a form of hash table¹⁰ is used to evenly distribute files across different folders. The files are stored in folders where the filename is changed to their SHA256 hash. As seen in Figure 4.29 and Code listing 4.15, the first two characters of the hash is the name of the root folder. The next two characters is the name of the subfolder. The file is then stored within that subfolder. This makes it

¹⁰https://en.wikipedia.org/w/index.php?title=Hash_table&oldid=1019270041 Accessed: 2021-05-03 12:03

possible to store large amount of files without affecting the performance of the filesystem [18].

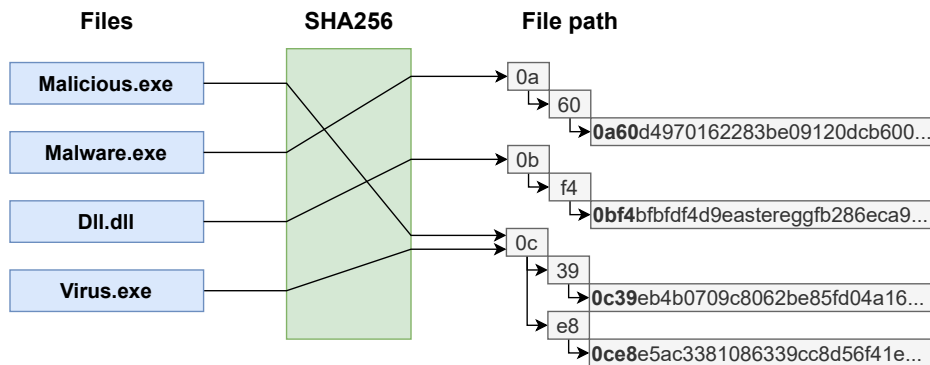


Figure 4.29: How the path for file storage is calculated

```

1
2 // get first two characters of hash
3 let folder1 = hash.slice(0, 2)
4 //get next two characters of hash
5 let folder2 = hash.slice(2, 4)
6 //concatenate to get the full path to store the file
7 let folderpath = `${process.env.dbfilePath}/${folder1}/${folder2}`
8 //create directory and return it
9 fs.mkdir(folderpath, { recursive: true }, (err => {
10   if (err) {
11     console.error(err)
12   } else {
13     resolve(`${process.env.dbfilePath}/${folder1}/${folder2}/${hash}`)
14   }
15 })

```

Code listing 4.15: Calculate filepath for file storage

4.5 Logging

Logging refers to the procedure of storing data about a system for a predetermined amount of time.¹¹ These logs can be analyzed to look for errors and unwanted activity, or to automatically warn when some parts of the system are not working as expected, to name a few. Logging must be implemented in a large infrastructure and it is included in the malware/Goodware Repository. There are simple ways to log, such as `console.log()`, however this is not an ideal solution as it can only be viewed in the console and will be deleted once the program stops. These are best used during the development process to debug, and may cause a small decrease in performance.¹² For more advanced logging capabilities, a dedicated library is needed. For Node.js, there are options like Winston, Bunyan and Log4js-node.

¹¹<https://www.techopedia.com/definition/596/data-logging>

¹²<https://blog.logrocket.com/node-js-logging-best-practices/>

4.5.1 Winston

Winston is a logger built for Node.js applications. As Node.js is heavily used for this project, a logger based on this runtime environment was needed. This logger is designed to be simple and flexible which is great, as the time spent on implementing logging will be limited[19]. The reasoning behind choosing Winston, is mainly because it supports multiple transports. This allows for simple remote storage of log files, with various types of logs, such as e.g. system-, user-, and error-logs. Additionally it is the most popular logger library for Node applications, which was taken into account when choosing it.

When implementing Winston, a dedicated logger directory is created for a clean structure. In this directory the various loggers are created. These loggers are then imported and referenced in the other files making up the repository, and when triggered they ship logs to a file.

The example below (Code listing 4.16) shows a logger designed for storing error messages that may occur.

```

1 const winston = require('winston');
2 require('winston-daily-rotate-file');
3
4 var errorTransport = new winston.transports.DailyRotateFile({
5
6   //Specify information about the logfiles including where to store them
7   name: 'errorlogger',
8   level: 'error',
9   filename: 'error-%DATE%.log',
10  dirname: 'logging/errorlogs',
11  datePattern: 'YYYY-MM-DD',
12  zippedArchive: true,
13  maxSize: '20m',
14  maxFiles: '14d'
15 });
16
17 const errorlogger = winston.createLogger({
18
19   //Formatting the log-message
20   format: winston.format.combine(winston.format.timestamp(), winston.format.json()),
21   //Ships the logs to the transport specified on line 4
22   transports: [
23
24     errorTransport
25   ]
26 })
27
28 //Export the logger
29 module.exports = errorlogger;

```

Code listing 4.16: Winston errorlogger

Figure 4.30 shows an example of how a info-log file may look.

```

backend > logging > infologs > ⓘ info-2021-04-19.log
1  {"message":"started virustotal","level":"info","timestamp":"2021-04-19T10:22:44.750Z"}
2  {"message":"Connected to MongoDB event","level":"info","timestamp":"2021-04-19T10:22:45.591Z"}
3  {"message":"Connection successfull","level":"info","timestamp":"2021-04-19T10:22:45.593Z"}

```

Figure 4.30: Example of info-log file

4.5.2 Winston-daily-rotate

When handling logs an important feature is to be able to swap to an empty log on a fitting time interval to ensure that no log file gets too big. This enables a more structured manner of storing log files, as well as making the logging flexible and customizable. The current interval for changing to a clean log-file is every 24 hours, however this can be easily changed. Each file is named with their respective date for easy backtracking.

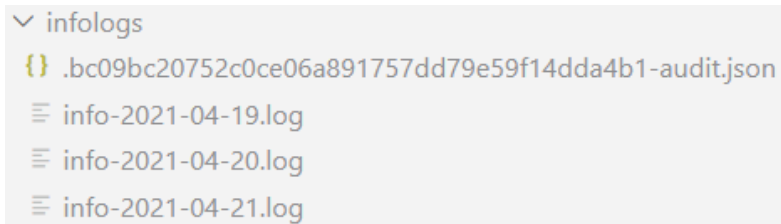


Figure 4.31: Example list of daily rotated logs

4.5.3 What to log

Deciding what to log is a task requiring some thought. After all, some info and data about the system is more useful when debugging and searching for unwanted activity.¹³ User activity is important to log, as this will enable traceability and accountability for the users. Additionally, system logs play an important part in gauging the overall state of the system.

4.6 System requirements

When deploying the repository onto servers, some requirements need to be met. It is recommended that the database, backend and frontend is deployed on three different servers. This is to ensure that the frontend and database is responsive when the backend is busy analyzing files. The requirements listed below are results from testing the system. Less resources than the minimum requirement is possible, but would affect the stability of the system.

¹³<https://coralogix.com/log-analytics-blog/important-things-log-application-software/>

4.6.1 Frontend

	Minimum	Recommended
Disk	50 GB	50 GB
CPU	2 Cores	4 Cores
RAM	4 GB	8 GB
OS	Ubuntu 18.04	Ubuntu 20.04
Software	Webserver (Apache / NGINX)	

The frontend is the least demanding part of the repository. Its main job is to serve static files to the user and request content from the backend.

4.6.2 Backend

	Minimum	Recommended	Notes
Disk	50 GB	200 GB	Disk size is mainly dependent on the size of the file storage.
CPU	4 Cores	16 Cores	More CPU means faster upload and analysis
RAM	16 GB	16 GB	
OS	Ubuntu 18.04	Ubuntu 20.04	
Software	Docker Node.js NPM	Git	

The backend is the the most CPU-demanding part of the repository, and thus need the most processing power. The processing power is mainly used to analyze files that are uploaded. This means that analysis is faster the more CPU cores it gets. It is worth noticing that when more analysis sources are added later in development, more processing power is needed.

4.6.3 Database

	Minimum	Recommended
Disk	50 GB	50 GB
CPU	2 Cores	4 Cores
RAM	4 GB	8 GB
OS	Ubuntu 18.04	Ubuntu 20.04
Software	MongoDB	

MongoDB is highly efficient. In testing(Section 4.6.4) it was discovered that a server dedicated for MongoDB running with 2 CPU cores, were able to insert

about 17 000 files in 2 minutes.

The disk usage on the database is further quite lean. 17 000 files uses around 108 MB of the disk, which means 1,7 million files would use 11 GB of disk on the database server. Note that this is only with one analysis source. As more sources are added, the documents in the database will use more disk space.

4.6.4 Testing

At the end of the project, the repository was tested to measure performance on upload and analysis, and locate eventual bugs in the software. The most interesting parts to measure performance were on the backend and the database. These two servers will experience much load when big archives are uploaded and analyzed. During performance testing, 17 000 files were uploaded to the backend, analyzed, and inserted into the database.

Backend

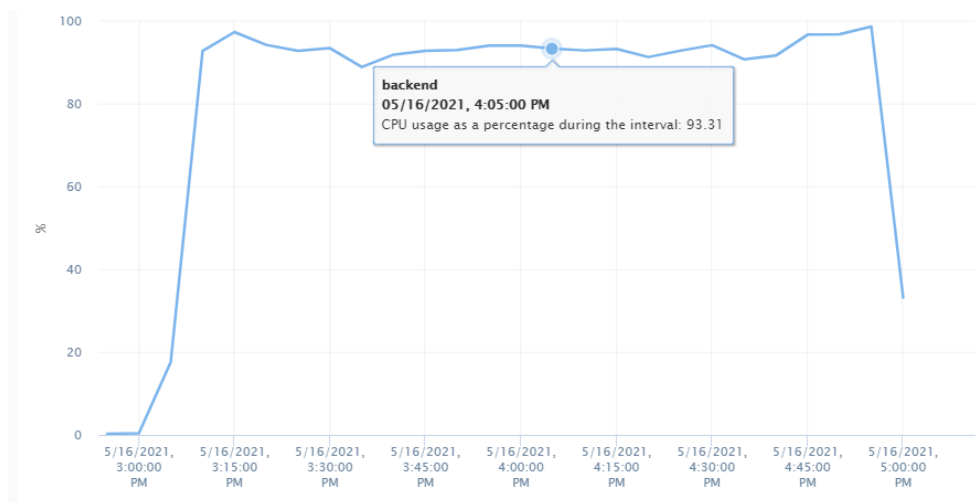


Figure 4.32: CPU usage of backend in percent

The main consumer of processing power on the backend is the analysis module. As seen in Figure 4.32, the backend used 1 hour and 55 minutes to analyze the provided 17 000 files. This translates to roughly 8500 files an hour. The backend was limited to the minimum requirement of 4 CPU cores and 16 GB of RAM. It is expected that the analysis time will scale linear relative to the CPU power.

Critique of testing

When testing the analysis module in the backend, the only analysis source that is implemented, PEframe, was used. This tool have been observed to spend a

vast difference in resources on files of the same size. Some files are observed to use up to 5 minutes in analysis, only to output several Megabytes of noise. This means that the selection of files used to test performance can give a wide spread in the results.

Database

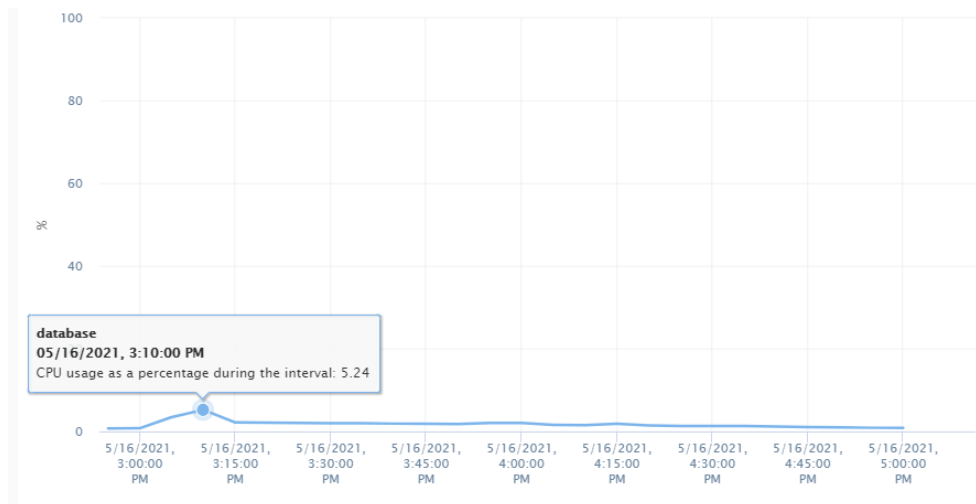


Figure 4.33: CPU usage of database in percent

As a result of how the backend is designed, the database will experience the most load when the initial entries of an upload is added. During this step, all files from the archive is added simultaneously. The database consequently needs to handle this traffic. When it comes to adding analysis data, the backend will add the data in a much larger timeframe, so there will not be as much load on the database at this point. The database was limited to the minimum requirement of 2 CPU cores and 4 GB of RAM. As seen in Figure 4.33, the database peaked at 5 percent CPU usage. This is the time when all initial entries was added to the database. Note that the actual use was higher, but the graph calculates the average use in a larger timeframe.

Chapter 5

Security and Legal aspects

Handling and storing large quantities of malware samples brings forth several challenges related to security and legal aspects. Some of the main challenges includes avoiding having malware executing on the NTNU network as well as avoiding misuse of the available samples. As the purpose of the platform is to facilitate research at the Department of Information Security and Communication Technology, making users and partners aware of these challenges is vital. In other words, ensuring the secure handling of samples is an integral part of this project. This chapter discusses the various security and legal related challenges faced, both in regards to technological, but also human aspects. Additionally, some suggested solutions to these challenges are described.

5.1 Security

When storing live malware samples that is being accessed by several people, there are many aspects to take into consideration related to security. After all, the main purpose of a malware is to cause damage, and/or disruption to users or systems. Any malicious activity is not only harmful for other users but also for NTNU from a public relations standpoint. Therefore it is important to make users aware of the consequences of breaking NTNU's guidelines.

As Feide has been implemented as the main way to access the platform, only people who are authorized and are involved with NTNU to some extent, are allowed to access the website and use it. There are security features and guards to protect the different components from unauthorized use. The only way of NTNU partners is to create a guest user with Feide and then gain access when an admin gives them authorization.

Confidentiality, Integrity, and Availability (CIA) of the platform and its data is an important aspect to consider while developing. The platform have been developed with this model in mind, and several decisions have been made with this model as a reason. Examples of that is choosing the platform stack and protecting API routes. Since a popular software stack is used, it is optimized

to prevent downtime and unexpected crashes. Protecting the API routes is one of the measures to ensure integrity and confidentiality of the data.

5.2 Secure storage of malicious executables

When storing malware, it is crucial that the malware is not executable and that any eventual antivirus on the server will not be able to scan and delete the malware. For this reason, the repository is encrypting the files with AES 256 in CBC mode, making the malware unreadable on the server. Zipping the file is not good enough, as modern antivirus engines are able to scan and delete files inside a zipped folder. While the malware only needs to be obfuscated to avert execution and antivirus, a simple obfuscation of the file data would suffice. However, that would demand making of a new function to manipulate the content of the file. This method have a lot of pitfalls and since encryption is a native function in Node, that was considered the better option. When a user uploads samples, the samples will be temporarily be stored in plaintext on the system while it is scanned and analyzed. This is not ideal, as at this moment the samples are both executable and able to trigger antivirus. The best solution here would be to never write the sample to disk in plaintext, but instead keep it as a datastream in memory and encrypt the sample before writing to disk.

5.3 Legal aspects

Most malware aims to harm or disrupt normal operation of a system. This means that there is potential for users of the repository to commit illegal activities with malicious samples. Such activities may result in legal ramifications against NTNU as they are responsible. Therefore, with the probable implementation of the repository, a plan describing the consequences of misusing the system should be composed. As a foundation for this plan, NTNU already manages an Information Security Policy[20]. This policy describes the basic rules and strategies regarding information security at NTNU. Some of these policies and laws are GDPR, ISO 27001 and The Personal Data Act (Personopplysningloven). These policies could be used as a base for the terms of service described in the next chapter.

5.3.1 Avoiding misuse

To avoid the misuse of malware samples downloaded from the system, a terms of service should be created. The terms of services should contain definitions of what a breach of the terms imply, so that users are aware of what they should not do when using the system. Additionally the terms should include what repercussions a breach will cause. As composing a terms of service is not included in the groups scope, this will be up to NTNU to handle. The terms

of service should in some way or form be acknowledged by the user either before being allowed access, or when entering the web page, as this puts the responsibility on to the user.

5.3.2 Copyright

During development of the repository, a variety of third party packages and libraries have been used. It is important to follow limitations given by the owner of the packages. All the packages in the repository uses licenses where modification and usage of the code is allowed free of charge. Some of the packages does however require credit for usage of the package.

5.4 Malware research ethics

Performing research and work related to malware comes with some ethical implications that should be addressed[21]. While malware research generally has good intentions, there are some often overlooked aspects in regards to ethics. For the project, the group received two compressed archives containing several thousand malware samples each. The purpose of receiving these were for testing whether the system could handle uploading large amounts of real samples. As the server running the platform was connected to the school network, great caution needed to be taken so that no malware would execute and cause harm on this network. Therefore, the IP's used was shared with the NTNU Security Operations Center (SOC) for surveillance, before attempting to upload these. These tests went well, however they are a testament to how cautious one has to be when researching with malware.

Chapter 6

Results and going forward

This chapter discusses the results from the development process. As features were often discussed weekly with the employer, some choices that were made during development are mentioned. Finally, some suggestions for future improvements of the repository are discussed.

6.1 Final product

The NTNU Malware Lab wanted a storage system for malware and goodware samples, in an effort to facilitate more efficient research and as well as to generate new knowledge and research methods related to information security. From the beginning, some general requirements as to what functionality the system should handle were set (See Section 2.2). However, during weekly meetings with the Malware Lab new functionality was often discussed and later implemented. Additionally the system should meet certain non-functional requirements, namely: Portability, Modularity/Scalability and Low complexity of maintenance as described in Section 2.7.

The final product of this project consists of a working proof of concept malware and goodware repository system. The system contains the main functional requirements, as well as lots of other functionality described earlier in this report. It allows for multiple users to log in and be authorized based on their role in the Feide system. The system supports uploading of files with the subsequent analysis of them. The data from the analysis is then viewable for each and every file. Additionally the users are able to download files of their choosing. Overall the group and the Malware Lab are very satisfied with the result.

Non-functional requirements

The non-functional requirements are also a vital measurement of whether the goals were reached. Below is a discussion of whether the non-functional requirements were met:

- **Portability:**
 - In regards to the portability of the system, setting it up on a new infrastructure is pretty simple. The frontend is easily deployed on a new infrastructure. Angular uses a compiler called ahead-of-time (AOT) which will compile all the code into efficient JavaScript code. This compiled code can be deployed on any webserver that can host HTML and JavaScript code such as NGINX and Apache in a matter of minutes. The backend can be deployed to a new infrastructure relatively quickly and simple. The project contains a file called *package.json* which is readable by Node.js and will install all necessary packages and modules in a few minutes. In the future, compiling the backend into a Docker container can be of interest. This way the environment is the same each time, and the container can be installed on a new server knowing it will work, every time.
- **Modularity/Scalability:**
 - The way the system is developed, adding new components and functionality is mostly pretty simple. With Angular on the frontend, adding new components is simply done with a single command to create its file structure and the adding relevant libraries and code. As this is an open source project and is developed by the group, the backend is designed with the intention to later be able to support more analysis tools and with that, more functionality. Therefore, this should be a relatively simple process.
- **Low complexity of maintenance:**
 - When it comes to maintenance, the system is designed to be of low complexity. The MEAN stack is modern and well documented. Third party modules picked during the development are ones that are still being maintained, if available. This was done to ensure that eventual bugs that might occur would be fixed if an issue was reported.

6.2 Choices made during the project

During the project some decisions were made that may have diverted from or supported the original plan:

- When creating a full stack, choosing a method that has been proven to work was a must. And therefore, the choice was the MEAN stack. Choosing the MEAN
- Innsida uses session tokens when handling authentication cookies. Therefore, to standardize what NTNU uses on their platform there was implemented a session-based authentication solution.
- Employer did not require Feide to be implemented, but mentioned it

would be best if it was. There was a solution for local authentication already implemented, but Feide was chosen for the final iteration.

- A NoSQL database structure was chosen, as the database needs to be able to handle data which may not be structured to fit in a SQL database.
- It was mentioned in the project plan that the design was not going to be a part of the project. But when working on the system there was necessary to have some design to make the system more usable. This does not mean that the design is finished as it needs a lot more polishing, but some design and CSS is in place to make the experience more passable.

6.3 Critique of final product

During the development, some improvements have been found. The frontend is developed with AngularJS, which is a fairly big and heavy framework. Some time was used to study the functionality of the framework. This time could instead have been used on developing more features in the repository if a lighter framework had been used. Further, during testing of the product, it was discovered that the repository was not able to extract certain type of encrypted zip files. This should have been discovered much sooner, and suggests that testing should have been more used in the development of the repository. Lastly, when files are uploaded, the backend spins up a container for each upload-object and deletes it afterwards. This could have been solved more neatly by having one container that is constantly running and analyzing files when they are transferred to the container.

Chapter 7

Conclusion

In this project a proof of concept repository for NTNU Malware Lab's malware and goodware samples has been developed. By researching and looking at other solutions to repositories a system has been made that fulfilled the needs that NTNU Malware Lab had. As a result, a repository which is modular, scalable, modern and easy to use was created.

The system enables an easy push-pull API, meaning uploading and storing the samples is easy as well as viewing the details and downloading the samples. In addition, the structure of how the samples are stored is more efficient and it is easier to both browse and find files when on a larger scale.

When users want to access the site and samples there is implemented restrictions to prevent unwanted tampering and usage. Enabling new users to be able to access has also been simplified and can be done from the platform itself. If let's say a sample to be used in an exam is uploaded to the platform, an admin or the uploader is able to set a flag which will make all students unable to download and view the file.

7.1 Project assessment

The proof of concept product that has been developed is satisfactory for both the group and the employer. Although if the development was to be done again the knowledge gained throughout the project would help with making the correct choices earlier and it would make the whole process of creating the full-stack faster and more efficient. Additionally, learning how to take requests from an employer has been important.

7.2 Knowledge gained

When working with this project and bachelor thesis, the group has learned both technical and academic skills. Firstly, when working on a project of this scale, planning, researching technologies, talking with employers and choosing a de-

velopment method that suits our group and project will make the development easier and prevent unnecessary revisions and ambiguities. Secondly for more technical skills, learning how to program a full-stack using Angular, Node.js, Express, MongoDB, TypeScript, Docker and other tools to realize the project that the employer has envisioned have been accomplished.

7.3 Limitations and future work

There are several steps that should be done to make this proof of concept in to a full working product:

1. A search engine should be implemented to better search through samples and their metadata. The current search functionality is dissatisfactory and was not prioritized in this project. Since the database consists of MongoDB, the use of ElasticSearch should be considered, as this search engine seems to be well supported for MongoDB.
2. It is recommended to add more analysis sources. An idea would be to create a container for each type of file to have designated tools working on different files. This is something that was not prioritized in this thesis, but would greatly improve the analysis functionality.
3. Working on the design to make it more aligned with other NTNU systems would be beneficial. As mentioned in the project plan (see Appendix A), the plan was not to focus on any design which was not necessary. Some design has been implemented to make the system more user-friendly, but more could be done and should be done.
4. Some optimizations should be done on the file list. Currently it only loads the page it currently is on, which is not bad, but when the pages get bigger (for example 250, 1000, 2500... rows per page) there should be implemented lazyloading to reduce the total API calls to the backend and make it seem faster for the users. Lazyloading means pre-loading the next 2-3 pages of the pagination on the users browser, so when the user click the next line it instantly goes to the next page which already was loaded on their browser
5. When talking about file list optimizations; the checkbox's batch action should be worked with. There has been developed a proof of concept of batch actions when reanalyzing and download, but none for favoriting and deleting the checkboxed rows.
6. When uploading a batch of samples, the uploader should be able to set options for the batch. For example if the uploader wants to upload a set of files for the current semesters exams the uploader needs to manually

change all the files after they have been added to the database and change all the files to be tagged with "exam". This is tedious and in the future a function to just say on the upload page that the current batch that is being uploaded should all be tagged with "exam".

7. The current parsing of the analysis data is not satisfactory. Trying to read raw JSON data is hard and not viable in the long term. Creating a solution where the JSON data is parsed and easy to read through and finding specific information should be a priority in future development on the frontend. When it comes to the VirusTotal data, a new widget that implements VirusTotal on third party websites have been released. This should be looked into if VirusTotal data is going to be used further down the line.

7.4 Evaluation of the groups work

From the start of the project the group was interested in the project and had a clear vision of how the repository should be. The group members started reading, researching and working on the project before the semester started to be as prepared as possible. Starting strong only meant that the rest of the project went well.

This was not the first time the group has been working together in a project. This is why organizing, communication and planning how to carry out the project went as well as it did. The group did not encounter any problems working together, and all conflicts have been solved with minimal issues. The group worked with developing the system from January and worked with the project almost every weekday Monday-Friday since project start. The group members schedules were known from before project start, in this case work and other school subjects, which meant they were accounted for early. A worklog has been created to track what and how much the members have done each working day throughout the project, the worklog can be seen on Appendix B.

The group members strengths and weaknesses were addressed early and meant choosing a role which enabled each other the most were set early. The group members had similar amounts of work distributed across the project development and writing the rapport. The supervisor and employer were both helpful with keeping the project on the right path and giving advice on what to implement.

The Kanban board showed to be tremendous help for the group as it enabled individual work and tracking what needed to be done by setting deadlines and objectives. As a development method it works well, it was easy to set up, plan and use.

Bibliography

- [1] *Virustotal*. [Online]. Available: <https://www.virustotal.com/>, Accessed: 2021-05-10 20:50.
- [2] M. E. Whitman and H. J. Mattord, *Principles of information security*. Cengage Learning, 2011.
- [3] *Ncsc*. [Online]. Available: <https://nsm.no/fagomrader/digital-sikkerhet/nasjonalt-cybersikkerhetssenter/>, Accessed: 2021-5-11 11:02.
- [4] A. Apvrille and M. Pourzandi, “Secure software development by example,” *IEEE Security & Privacy*, vol. 3, no. 4, 2005.
- [5] M. R. Stytz, “Considering defense in depth for software applications,” *IEEE Security & Privacy*, vol. 2, no. 1, pp. 72–75, 2004.
- [6] A. Adhikari, “Full stack javascript: Web application development with mean,” 2016.
- [7] S. Alimadadi, A. Mesbah, and K. Pattabiraman, “Understanding asynchronous interactions in full-stack javascript,” in *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*, IEEE, 2016, pp. 1169–1180.
- [8] *Web development stacks – what stacks (should) we use in 2021?* The Software House, Jul. 2020. [Online]. Available: <https://tsh.io/blog/web-development-stacks/>, Accessed: 2021-3-1 12:00.
- [9] M. A. Jadhav, B. R. Sawant, and A. Deshmukh, “Single page application using angularjs,” *International Journal of Computer Science and Information Technologies*, vol. 6, no. 3, 2015.
- [10] *Typed javascript at any scale*, TypeScript. [Online]. Available: <https://www.typescriptlang.org/>, Accessed: 2021-2-10 14:10.
- [11] A. Oualim, *How to serve an angular app with node js api on a nginx server*. [Online]. Available: <https://medium.com/@anasecn/how-to-serve-an-angular-app-with-node-js-api-on-a-nginx-server-ca59de51850>, Accessed: 2021-5-19 10:30.

- [12] Ryan Chenkie, “Angular authentication: Using route guards,” Jul. 2017. [Online]. Available: https://medium.com/@ryanchenkie_40935/angular-authentication-using-route-guards-bf7a4ca13ae3, Accessed: 2021-4-29 13:34.
- [13] Sherry Hsu, “Session vs token based authentication,” Jun. 2018. [Online]. Available: <https://medium.com/@sherryhsu/session-vs-token-based-authentication-11a6c5ac45e4>, Accessed: 2021-4-26 14:55.
- [14] Feide, *Openid connect/oauth technical details*. [Online]. Available: https://docs.feide.no/service_providers/openid_connect/index.html, Accessed: 2021-5-15 14:05.
- [15] T. Ukachukwu, *Authentication using passport.js in a node.js backend api*. [Online]. Available: <https://javascript.plainenglish.io/authentication-using-passport-js-in-a-node-js-backend-api-51e9946549cb>, Accessed: 2021-5-19 11:40.
- [16] Passport, *Passport overview*. [Online]. Available: <http://www.passportjs.org/docs/>, Accessed: 2021-5-15 14:39.
- [17] *Postman 2017 state of api survey*. [Online]. Available: <https://www.prweb.com/releases/2017/10/prweb14767541.htm>, Accessed: 2021-5-14 14:30.
- [18] Nick Coons, *Storing large files*, Jul. 2013. [Online]. Available: <https://stackoverflow.com/questions/17935978/storing-large-files-binary-data-in-a-mysql-database-when-is-it-ok/17941042#17941042>, Accessed: 2021-3-13 11:34.
- [19] *Winston*, Techopedia, Aug. 2020. [Online]. Available: <https://github.com/winstonjs/winston>, Accessed: 2021-2-9 10:15.
- [20] *Policy for information security*. [Online]. Available: <https://innsida.ntnu.no/wiki/-/wiki/English/Policy+for+information+security>, Accessed: 2021-5-12 14:00.
- [21] J. P. Sullins, “A case study in malware research ethics education: When teaching bad is good,” in *2014 IEEE Security and Privacy Workshops*, IEEE, 2014, pp. 1–4.
- [22] *The mongodb 4.4 manual*, MongoDB. [Online]. Available: <https://mongoosejs.com/docs/index.html>, Accessed: 2021-2-10 14:54.
- [23] *Mongoose getting started*, Mongoose. [Online]. Available: <https://mongoosejs.com/docs/index.html>, Accessed: 2021-2-10 14:58.
- [24] The Net Ninja, *Mongodb for beginners*, Jan. 2017. [Online]. Available: <https://www.youtube.com/playlist?list=PL4cUxeGkcc9jpvoYriLI0bY8D0gWZfi6u>, Accessed: 2021-2-10 14:59.

- [25] Erkan Güzeler, “Angular role-based routing access with angular guard,” Apr. 2020. [Online]. Available: <https://medium.com/echohub/angular-role-based-routing-access-with-angular-guard-dbecaf6cd685>, Accessed: 2021-2-10 15:13.
- [26] guelfoweb, *Peiframe*. [Online]. Available: <https://github.com/guelfoweb/peiframe>, Accessed: 2021-05-10 20:50.
- [27] *Rest api tutorial*. [Online]. Available: <https://restfulapi.net/>, Accessed: 2021-5-14 14:30.

Appendix A

Additional Material

A.1 Project agreement



PROJECT AGREEMENT

between NTNU Faculty of Information Technology and Electrical Engineering (IE) at Gjøvik (education institution), and

NTNU

_____ (employer), and

Erlend Husbyn, Gjert Michael Homb, Michael Cortes Birkeland og

Christian Simoes Isnes

_____ (student(s))

The agreement specifies obligations of the contracting parties concerning the completion of the project and the rights to use the results that the project produces:

1. The student(s) shall complete the project in the period from 11.01.2021 to 20.05.2021.

The students shall in this period follow a set schedule where NTNU gives academic supervision. The employer contributes with project assistance as agreed upon at set times. The employer puts knowledge and materials at disposal necessary to complete the project. It is assumed that given problems in the project are adapted to a suitable level for the students' academic knowledge. It is the employer's duty to evaluate the project for free on enquiry from NTNU.

2. The costs of completion of the project are covered as follows:
 - Employer covers completion of the project such as materials, phone/fax, travelling and necessary accommodation on places far from NTNU. Students cover the expenses for printing and completion of the written assignment of the project.
 - The right of ownership to potential prototypes falls to those who have paid the components and materials and so on used to make the prototype. If it is necessary with larger or specific investments to complete the project, it has to be made an own agreement between parties about potential cost allocation and right of ownership.

3. NTNU is no guarantor that what employer has ordered works after intentions, nor that the project will be completed. The project must be considered as an exam related assignment that will be evaluated by lecturer/supervisor and examiner. Nevertheless it is an obligation for the performer of the project to complete it according to specifications, function level and times as agreed upon.
4. All passed assignments will be registered and published in NTNU Open, which is NTNU's open archive.

This depends on that the students sign a separate agreement where they give the library rights to make their main project available both on print and on Internet (cf. The Copyright Act). Employer and supervisor accept this kind of disclosure when they sign this project agreement, and they must possibly give a written message to students and head of Department if they during the project period change view on this kind of disclosure.

The total assignment with drawings, models and apparatus as well as program listing, source codes and so on included as a part of or as an appendix to the assignment, is handed over as a copy to NTNU who free of charge can use it in lessons and in research purpose. The assignment or appendix cannot be used by NTNU for other purposes, and will not be handed over to an outsider without an agreement with the rest of the parties in this agreement. This applies as well to companies where employees at NTNU and/or students have interests.

5. The assignment's specifications and results can be used by the employer's own work. If the student(s) in its assignment or while working with it, makes a patentable invention, relations between employer and student(s) applies as described in Act respecting the right to employees' inventions of 17th of April 1970, §§ 4-10.
6. Beyond the publishing mentioned in item 4, the student(s) have no right to publish his/hers/theirs assignment, fully or partly or as a part of another work, without consensus from the employer. Equivalent consent must be made between student(s) and lecturer/supervisor regarding the material placed at disposal by the lecturer/supervisor.
7. The students shall hand in the assignment with attachments electronic (PDF) in NTNU's digital exam system. In addition the students shall hand in a copy to the employer.
8. This agreement is drawn up with one copy to each party. On behalf of NTNU it is the head of the Department/Group that approves the agreement.
9. In each case it is possible to enter separate agreement between employer, student(s) and NTNU who closer regulate conditions regarding issues such as ownership, further use, confidentiality, cost coverage, and economic utilization of the results.

If employer and student(s) wish an additional or new agreement, this will occur without NTNU as a partner.

10. When NTNU also act as employer, NTNU accede to the agreement both as education institution and as employer.

11. Possible disagreements concerning understanding of this agreement are solved by negotiations between the parties. If consensus is not achieved, the parties agree that the disagreement is solved by arbitration, according to provision in Civil Procedure Act of 13th of August 1915, no 6, chapter 32.

12. Participants by project implementation:

NTNUs supervisor (name): Mohamed Abomhara

Employers contact person (name): Andrii Skelaginov

Student(s) (signature): Erlend Hordbryn date 12.01.2021

Gjerd T. Horn date 12.01.2021

Michael C. Muehlen date 12.01.2021

Christian S. Jones date 12.01.2021

Employer (signature): [Signature] / Andrii Skelaginov / date 14.01.2021

The Project Agreement is to be handed in in digital version in Blackboard. Digital approval by head of the Department/Group.

If a paper version of the Agreement is needed, it must be handed in at the Department in addition.

Head of Department/Group (signature): _____ date _____

A.2 Project description,

Title of the Bachelor Thesis: Building Multi-user Malware and Goodware Repository

The main goal of this project is to enhance malicious software-focused research and operations support at NTNU (Malware Lab, System Security Group, SOC, Norwegian Cyber Range, etc.) by building a repository (software platform) to give access to high-quality dataset of recent "malicious" samples and "good" software samples. The value of the categorized malware samples enriched with cyber threat intelligence is the core when it comes to research initiatives, project proposals and master project topics at NTNU. Current capabilities include multiple novel datasets acquired and built through years, e.g. possession of 400+ archives with 12 TBytes of unstructured malware samples from VirusShare. The platform needs to be developed allowing access to most recent and relevant malware samples serving as a core component in education and research activities at NTNU. Besides source of files and characteristics from VirusTotal, students should consider FireEye FLARE team's CAPA - open-source tool to identify capabilities in executable files that can be used to define characteristics in addition to VirusTotal. Once completed, the platform will allow collaboration from multiple directions: uploading and updating datasets, fetching malware samples under defined specifications and cyber threats intelligence, and sharing through machine learning and unsupervised learning.

Anticipated tasks:

1. Design top-down framework covering: optimal files and characteristics storage, database design, "push" / "pull" API, user access and roles
2. Define functional and non-functional requirements to handle ELF / PE32 / APK / Mach-O files
3. Focus on availability and building up new sources of characteristics
4. Proof-of-concept with the focus on: portability, modularity, and complexity of maintenance

Contact:

Andrii Shalaginov, andrii.shalaginov@ntnu.no
Christoffer Vargtass Hallstensen, christoffer.hallstensen@ntnu.no
Geir Olav Dyrkolbotn, geir.dyrkolbotn@ntnu.no

References:

1. VirusTotal, <https://www.virustotal.com/gui/>
2. >CMATLiS< - Complex Mobile Application Testing Laboratory for Information Security assessment, <https://github.com/ashalaginov/CMATLiS>
3. Storing large files / binary data in a mysql database: when is it ok?, <https://stackoverflow.com/a/17941042>
4. Functional Requirements vs Non Functional Requirements: Key Differences, <https://www.guru99.com/functional-vs-non-functional-requirements.html>
5. VirusTotal API v3 Reference, <https://developers.virustotal.com/v3.0/reference>
6. CAPA: The FLARE team's open-source tool to identify capabilities in executable files. <https://github.com/fireeye/capa>

A.3 Group rules

Group rules

Michael Cortes Birkeland, Erlend Husbyn, Christian Isnes,
Gjert Michael Torp Homb

The point of this section is for the group to have a collective understanding of what is both expected and required of them. This includes a description of how a regular week should look like in terms of amount of work and other routines like meetings. Additionally, there is a set of rules which should be followed, and the consequences of breaking them.

Routines

The routines are a general set of weekly activities that should be followed to achieve the main goals of the week. As a general, there should be an attempt to individually perform 25-30 hours of work – weekly. The group has agreed with the supervisor to conduct bi-weekly meetings, and weekly meetings with NTNU Malware lab. Within the team, there should be a weekly meeting. Additionally, the group will communicate daily over the Internet and we intend to continue with this over the course of this thesis. Should an emergency situation occur, group members need to be prepared to take over another group members task.

Rules

Group members are expected to meet at the agreed meetings at the correct time, unless a valid reason is presented.

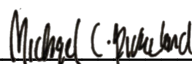
Group members are expected to do their given tasks within the specified time frames.

Consequences of breaking rules

Repeated violations of these rules will lead to the group forwarding these issues to the supervisor and if this does not help we would have to escalate to having to contact the bachelor coordinator.

Signatures:

Michael Cortes Birkeland:




Erlend Husbyn:



Christian Isnes:



Gjert Michael Torp Homb:



A.4 Projectplan

Project plan

Building Multi-user Malware and Goodware Repository

Michael Cortes Birkeland, Erlend Husbyn, Christian Isnes,
Gjert Michael Torp Homb

1. Goals and constraints

1.1. Background

At NTNU, many students and researchers are working with research and education related to Malware and Goodware. This type of research and study requires some place to store malware and goodware samples in an efficient and structured manner to be used for the purpose of research and education.

A malware is a software designed with the intent to cause damage, disrupt or gain access to a network or computer system.¹ Goodware, on the other hand, is software that does what it is supposed to do with no unexpected behaviour.² A multi-user Malware and Goodware repository refers to a storage system allowing multiple users with different roles such as researcher or student to interact with files and metadata of the files in the database. These roles allow for different functionality depending on who the user is.

The current capabilities for storing malware and goodware samples for research and education at NTNU lack structure and functionality. These malware and goodware samples are an important part of everyday work of students, researchers and staff. Therefore, the NTNU Malware lab wants to have a better way of storing these samples, which essentially is the task we have been given.

1.2. Project goals

The overall goal for this entire project is to enhance the research capabilities as well as the operations support for the various groups and departments at NTNU. These groups include the Malware Lab, System Security Group, SOC, Norwegian Cyber Range, etc.

¹ <https://www.forcepoint.com/cyber-edu/malware>

² <https://androidforums.com/threads/what-is-goodware.832075>

The intended goals of this thesis are as follows:

- Develop a working multi-user malware and goodware repository with functionality such as upload/download, searching by metadata and adding relevant information to a file such as tags.
- Improve the current unstructured and functionless system making research and education within and among various research groups more efficient.
- Make the multi-user malware and goodware repository more portable and easy to maintain.

1.3. Constraints

There are some constraints to this project. As this is a project within NTNU, the organizational rules should be followed. Additionally at the start of the bachelor a project agreement is signed, giving certain guidelines for what can and can't be done in regards to the project. For the project dynamic analysis and sandboxing of the samples uploaded will not be considered.

2. Scope

2.1. Problem area

The necessity of this project arised due to the fact that the current capabilities for storing malware and goodware in use within NTNU is outdated. These capabilities consist of multiple datasets containing samples that have been built through the years, including 400+ archives with about 12TB of samples. The most important factor in why a new storage capability is wanted, is the unstructured fashion these samples are stored in. To make fetching samples for use in research and education more efficient, there is a need for a more structured and sophisticated storage system.

2.2. Limitations

- The repository is only going to perform static analysis³ of the samples with tools like virustotal and third party libraries. The tools in question are going to be determined later in the development phase. However, it is built to be extendable with more tools in the future.

³ Analysis of the malware performed without executing the program

- The frontend is made to be functional, with little to no graphical design in mind.

2.3. Task description

The project should be a proof of concept full-stack solution of a malware and goodware repository. The final report, and repository, should cover optimal file and characteristics storage, database design, REST-API, user access and roles. Along with definitions of functional and nonfunctional requirements to handle ELF/PE32/APK and Mach-O files. These file types are executables made for the most common operating systems. ELF is made for Linux⁴, PE32 for Windows⁵, APK for Android⁶ and Mach-O for Apple based operating systems⁷. These file types behave differently, meaning that certain precautions need to be taken when implementing support for these types of files.

Furthermore, the platform should be able to perform static analysis of the samples uploaded using linked third party services, and local static analysis libraries.

3. Project organization

3.1. Responsibilities and roles

Gjert - Group leader, responsible for communication

Michael - LaTeX responsible

Christian - Code Quality Assurance

Erlend - Responsible for conflict resolution

Responsible for communication: There will be two meetings a week for follow up with team members and bi-weekly for follow up with the supervisor. Gjert is responsible for arranging the meeting calls and planning them.

LaTeX responsible: We will be writing the main project document in LaTeX on the Overleaf platform. Michael is responsible for ensuring that the document runs correctly and

⁴ https://en.wikipedia.org/wiki/Executable_and_Linkable_Format

⁵ https://en.wikipedia.org/wiki/Portable_Executable

⁶ <https://fileinfo.com/extension/apk>

⁷ <https://en.wikipedia.org/wiki/Mach-O>

Code Quality Assurance: Christian is responsible for the quality of the code pushed to Git, and reviewing all pull-requests to the master branch. It is important that the syntax of all code pushed is consistent and well commented to facilitate future development.

Responsible for conflict resolution: Should any conflict or disagreement arise within the group, Erlend will try to be a mediator to hopefully be able to solve the conflict internally. Should this prove difficult, Erlend will include the group's supervisor to resolve the conflict.

3.2. Routines and rules in the group

The point of this section is for the group to have a collective understanding of what is both expected and required of them. This includes a description of how a regular week should look like in terms of amount of work and other routines like meetings. Additionally, there is a set of rules, and the consequences of breaking them.

Routines

The routines are a general set of weekly activities that should be followed to achieve the main goals of the week. As a general, there should be an attempt to individually perform 25-30 hours of work – weekly. The group has agreed with the supervisor to conduct bi-weekly meetings, and weekly meetings with NTNU Malware lab. Within the team, there should be a weekly meeting. Additionally, the group will communicate daily over the Internet and we intend to continue with this over the course of this thesis. Should an emergency situation occur, group members need to be prepared to take over another group members task.

Rules

Group members are expected to meet at the agreed meetings at the correct time, unless a valid reason is presented.

Group members are expected to do their given tasks within the specified time frames.

Consequences of breaking rules

Repeated violations of these rules will lead to the group forwarding these issues to the supervisor and if this does not help we would have to escalate to having to contact the bachelor coordinator.

4. Planning, follow-up and reporting

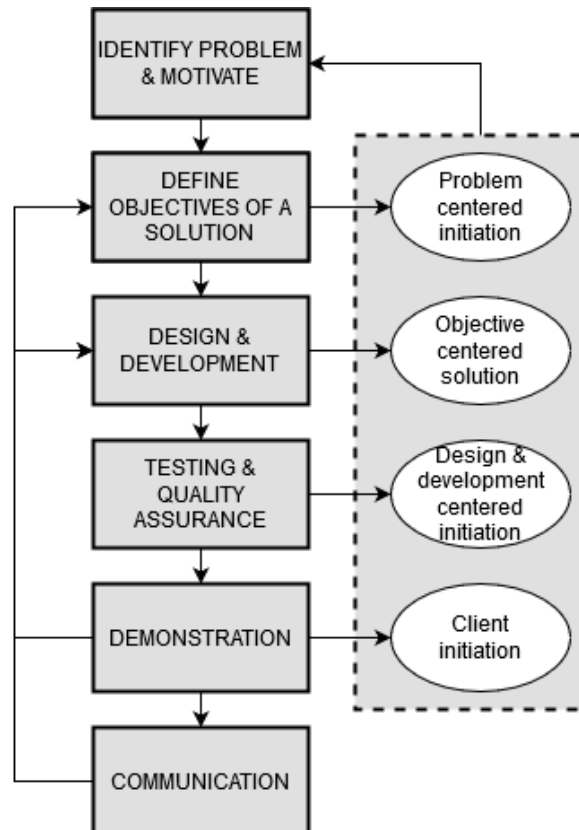
4.1. Main division of the project

Choice of System development model / process framework with argumentation

We have chosen to use the Kanban board system development framework for our development focused project. The Kanban board was the most versatile and optimal for individual working. The Kanban board gives the team members the opportunities to work whenever they want and track their own progress on the kanban board.

■ **Choice of research methodology and approach**

The research methodology is based on researchgates⁸ example of a research methodology.



The methodology has been edited and been adjusted to be more aligned to a development based project by adding the testing and quality assurance.

The steps of the model are the following:

- Identify problem & motivate means identifying the problem(s), and why the module/function needs to be implemented. This should be based on chapter 2.1 Problem area.

- Define objectives of a solution figuring out what the final state of the module should contain and be able to do. This should be based on chapter 1.2 Project goals.
- Design and development means figuring out the design of the module and developing it.
- Testing & quality assurance defines a set of procedures regarding testing that the code, and module work, as well as documenting the work. These procedures are defined in chapter 5 Organizing quality assurance.
- Demonstration refers to demonstrating the module to the supervisors, throughout the process, so that they have an idea of our progress and also if they agree with our solution.
- Communication refers to discussing whether the goals has been reached, and if necessary going back up the ladder and redoing what needs change.

- **Describe the group's way of following the model**

As Kanban board is chosen as our System development framework, an online resource trello.com will be used to host the Kanban board. The board will be divided into 7 parts: Backlog, to do, Development, Testing, Deployment, Documentation and Done. Backlog being pre-development of a task and done being the end when the task has been fully developed, tested and documented. The choice of methodology has been based on the Kanban Board, identifying problems is the initial step of any Kanban based development projects.

4.2. Plan for status meetings and decision points in the period

There will be continuous communication between the group-members almost daily, where small decisions will be taken. Once per week the group will have a status meeting where each member covers the status on what they're working with. This meeting will generally be held on Mondays at 12:00, however this may vary depending on the availability of the group members. During this meeting, the group will also handle bigger decision-points. When meeting with project owners or our supervisor, meeting reports will be written. Also, the group has agreed on a bi-weekly meeting with the supervisor for follow-ups and guidelines. In addition to, on call meetings with the supervisor when needed.

5. Organizing quality assurance

5.1. Documentation and source code

The thesis will be documented in a LaTeX document while working on the project. All references will also be documented and placed in the bibliography. Code being worked on will be pushed onto a git repository, and for this project Gitlab will be used. For testing the developed software, a server owned by a member of the group, located in the NTNU server-room will be used.

5.2. Risk analysis

When working on a project of this size, there are several things that might not work out as planned. Therefore, it is necessary to perform a risk analysis to mitigate the probability of an unexpected incident halting our progress. Although some risks are not easily mitigated, there is still a point in being aware of them.

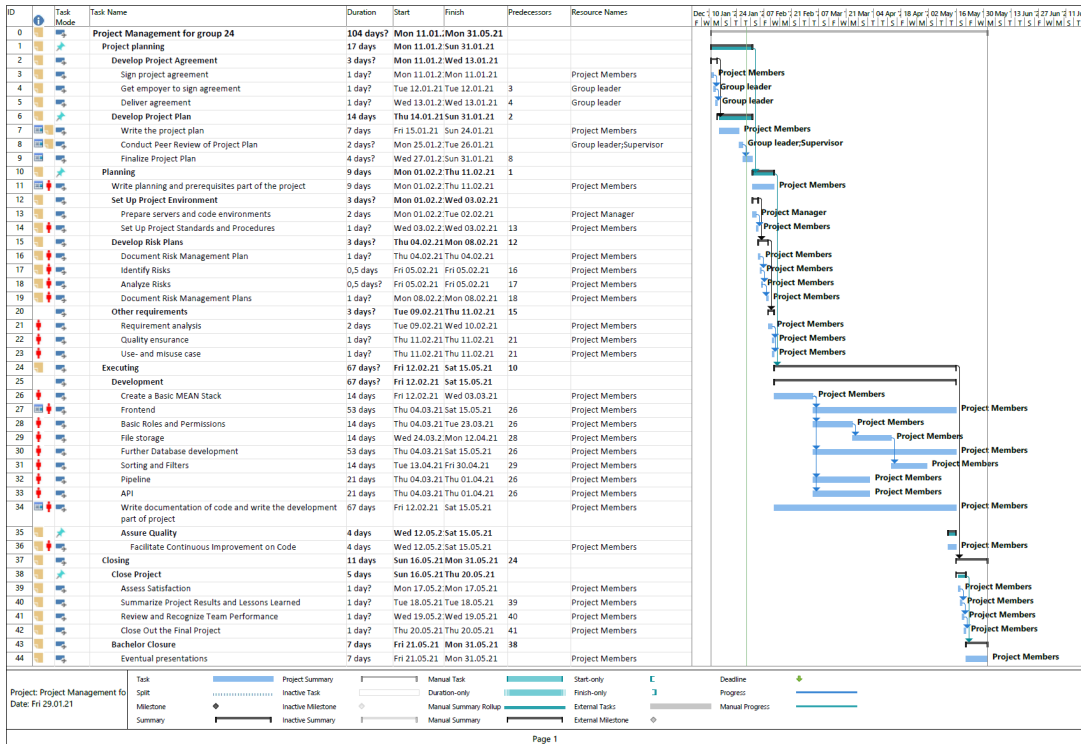
Possible risks:

Risk	Description	Mitigation(s)
Extended downtime of text editing software	Downtime in the text editing software may cause loss of work and inability to work for the downtime period.	By keeping regular local backups, chances of losing lots of work is smaller. Additionally we should have other text editing software available. The overleaf document is being backed up every day at 12:00 and 00:00.
Group member(s) or supervisor(s) become ill over extended period of time	Due to the current pandemic there is an inherent risk that people involved in the project may become ill.	The government regularly updates the public on new guidelines, and these should be monitored and followed.
Loss of code	Crashes or unforeseen situations may cause loss of code.	This situation can be mitigated by regularly pushing code to the groups repository. Code should be pushed after ensuring that it is working.
Lack of time/Scope is too large	If the defined scope turns out to be too massive, there might not be enough time to complete the set goals.	To prevent taking on too much work, sticking to the specified tasks and goals will be important. Further functionality can be added if time allows it.

6. Plan for implementation

6.1. Gantt chart

The Gantt chart is a plan for the whole process from start to finish, with simple descriptions of the work and their respective time-limits. The final project document will be worked on throughout the project.



A.5 Meeting Logs

A.5.1 With supervisor

18.01.2021

Referat møte 18.01

Først litt introduksjoner

Han gleder seg til å jobbe med oss :)

Han vil helst ha møte på slutten av dagen 14-15
Vi skal diskutere om vi vil ha fixed ukentlige møter, eller biweekly.

Project plan meeting førstkommende fredag22. Kl 14-15
Dele draft 21. så han kan lese igjennom først
Vil ha tilgang til litt div filer gjennom teams, gir også tilgang til gitlab.

Han kan gjerne motta i Word.

TLF: 0047 91 526 213
Campus: A126 eller A127

Send epost med alle våre adresser så setter han opp møter for oss.

16.03.2021

Referat 16.03.2021 med veileder

Ask if NTNU can provide a license to virustotal to be able to analyze more than 500 samples a day.

Mohamed is now added to the overleaf document,

Use git product number to refer to previous work.

How feide is integrated. Not important to explain in depth of how feide works.
Mention how we customized previous code that we have used.

No problem to use previous work just mention how and why we used it and if we customized it.
Changes we have made should be added to the appendix.

18.01.2021

Referat 22.04.2021

Agenda:

- Discuss comments received from supervisor

Why does the NTNU not use a commercial product?

Remind Mohamed within 27. April to give a second opinion.

Employer: Malware lab

A.5.2 With employer

20.01.2021

**Referat 20.01.2021
Møte med Andrii**

Why we chose MEAN stack.

Comments about the current plan:

- Plan looks fine
- Motivation for the task
- - Many people working with malware analysis. ELA file or exe's. Data more or less the same. Need file storage and metadata storage.
- Basic front-end functionality. File size. Search by metadata. Access control. E.g admin, student, research members
- Local users are fine for us. Study how to implement it with feide.
- Admin: delete/upload files.
- Research members: Access and upload files.
- Master/bachelor students: Access certain files.
- Pipeline: Specify how to add other modules
- PEframe for metadata to put in the database, in addition to virustotal.

1.

- Specify users.
- How files are uploaded

2.

- Think of some limitations

3.

- Add metadata for example

4.

- Modularity for the research team at NTNU. E.g. add new modules and such

- Risk management is important for the project plan.
- Other systems than the MEAN stack, might be a question.

- Include similar projects in the report.

27.01.2021

Referat 27.01.21
Meeting with Andrii and Geir Olav

Goodware: Something we know is good. Things we definitely know are not malware.
E.g. binaries from windows. Should be defined in the repository.

General malware description (what kind of malware) in the repository

Tendency that code has good intentions can be used for bad actions.
Should we somehow label goodware that we know have been used for evil purposes?
User added comments maybe.

Functionality to add more info on a file. Use for REM courses. Label if it has been used
for an exam. Suitable for dynamic analysis for example?

How to handle passwords?

Admin creates users. Applies username and role. Receives one-time password that
admin passes on to the user.

Making a group for a class may be a relevant task.

Trouble regarding passworded zip files.

Upload zip files to the repository. Should be able to decrypt zip files with known
password

samples should be encrypted when being stored.

Full stack-framework. Front to back-end.

Current capabilities:

Stored in virtual machines. Peer-to-peer.

There should be focus on Logging. Accountability.

Admin labels. May be extended in the future or if we have time.

Let IT know of our project.

Ethical and legal part of thesis. Ethical implications of storing malware. Legal
implications. What can go wrong?

03.02.2021

Referat 03.02.21 meeting with Andrii

Show that we have ability to add data.
Avclass is a tool that we may use to classify data.

Option to classify should be implemented.
Get the report started soon.

Møte neste torsdag kl 10

<https://github.com/malicialab/avclass>

11.02.2021

Referat 11.02.2021
Møte med Andrii

Authorization and roles based on tags. Who should access what?

Admin -> Bulk deletion etc. Log files

Researcher -> Can edit etc, read/write.

Student -> Access to files unless tagged with hidden or exam ex.,. read only

Admin, researcher, student. List of tags not accessible to students. Research level should be able to edit/delete metadata/tags.

What should be in the report:

Explore some basic status of malware.

CAPA output.

Document 2-factor authentication in report, and implement if possible. Feide also.

19.02.2021

**Referat 19.02.2021
Møte med Andrii**

Able to upload files. Working on opening zipped files.
Logging up and running. Decide what to log.
Feide login running.

Mach-O fetch metadata. Challenging to analyze.

Currently private server. Do we have to talk to someone before uploading some malware to this. It's ok as long as Christoffer knows the IP.

Static analysis on the zip file itself. Can't recall any examples of malicious zip files, the payload is most interesting. Most focus on the executables.

25.02.2021

Referat 25.02.2021

Thoughts on logging?

Ta opptak av en demo i forkant av møte.

Sha256 er ryddig å forholde seg til for sortering. Metadata fil i en mappe med filen, så trenger man ikke tilgang til databasen for å

Logging on admin panel. Who, when and what.

Access periodically. Might not need logs for every day

Download multiple files at the same time?

Password protected zip files. Maybe find another than infected.

08.03.2021

Referat 08.03.2021

Demo of feide and basic UI.

Take general items to show in expanded file view as many of the fields change regularly.

Load balancing in regards to VirusTotal analysis. 4 requests per minute or something. Look into rate limits.

Look at e.g. peframe to get metadata for particular platforms. Windows: peframe. Linux: readelf. Would be nice to have platform specific information for the files. This will be helpful for the search function.

Andrii will send us malicious files today.

How to organize filetype.

```
/usr/share/applications/defaults.list  
image/bmp=org.gnome.eog.desktop  
image/jpg=org.gnome.eog.desktop  
image/pjpeg=org.gnome.eog.desktop  
image/svg+xml=org.gnome.eog.desktop
```

Some other repos have hex view. If possible to implement as a web component that would be helpful. Then you can have hexadecimal representation etc.

Demonstration of unzipping files.

Maybe add a shortcut func where we place files in a folder, and the script will process.

How to classify researchers?

Admin may be able to edit the database?

Misuse. With feide problems with user accounts being compromised isn't too realistic.

17.03.2021

Referat 17.03.2021

Short presentation for NTNU might happen in the near future. We will be noticed.
Access control more or less in place, says Andrii.

System currently running on a private server in NTNU.

System logs implemented soon.

Maybe a limit on how many samples a student/single user can download. (E.g. 10 000 a day)
Rather have admins being able to hide certain samples, for example ones that will be used for exams or assignments.

: tag file with exam, hidden, private

25.03.2021

Referat 25.03.2021

Presentation: Maybe after easter.

Domain name: Up after easter.

Adding a logo would be nice.

Proof-of-concept. Portability? Specify versions of software components. All info that is needed to move the product. Snapshot of software versions. Package files are in the system.

Modularity

Classify malware: AVclass

Parallel optimization of for loops to classify more than one file.

08.04.2021

Referat 08.04.2021

Mainly look at the report.

When uploading files the system returns files is uploaded. What happens next.

Pefram question. Sometimes peframes. Does strange things. Sometimes it produces files that are bigger than itself. 55 mb is too big.

2 causes:

- Symbols sequence.

Security of the whole solution. Describe what measures we have regarding executing malware

Be aware of challenges regarding accidental execution of malware. How trustworthy is third-party tools? Make sure to write some lines about this in the thesis.

In terms of open source programs, check their reviews. You never know what code is in open source projects.

Concerns about storing large amounts of malware.

15.04.2021

Referat 15.04.2021

Core functionality, Download, upload, see info.

Hundreds of thousands of files. What will happen? What are the bottlenecks. What are the limitations?

Searching bottlenecks. Monolithic search engine like sphinx. Basic search functionality will be ok for now.

22.04.2021

Referat 22.04.2021

Agenda:

- Vise demo av download og litt søk.

Searching in MongoDB is difficult.

Sphinx search engine.

Store passwords of zipped files somehow, to avoid having to download large batches more than one time. Consider resilience in the long run.

Question from Andrii:

- Regarding what happens after the delivery on May 20th.

29.04.2021

Referat 29.04.2021

Detail more with screenshots in the appendix.

Remember to credit source code.

Appendix B

Additional documentation

B.1 API Documentation

Malware and Goodware Repository

Table of Contents

[File](#)

- [GET /api/v1/file/count](#)
- [POST /api/v1/file/download](#)
- [GET /api/v1/file/](#)
- [GET /api/v1/file/{hash}/file](#)
- [GET /api/v1/file/{hash}](#)
- [POST /api/v1/file/reanalyze](#)
- [PUT /api/v1/file/tags](#)
- [POST /api/v1/file/upload](#)

[User](#)

- [GET /api/v1/user/favorite/count](#)
- [GET /api/v1/user/favorite](#)
- [PUT /api/v1/user/favorite](#)
- [GET /api/v1/user](#)
- [DELETE /api/v1/user/{id}](#)
- [GET /api/v1/user/{id}](#)
- [PUT /api/v1/user/{id}](#)
- [GET /api/v1/user/profile](#)

File

GET /api/v1/file/count

Get the number of files in the database (apiV1FileCountGet)

Return type

Object

Example data

Content-Type: application/json

```
"{}"
```

Produces

This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- application/json

Responses

200

ID of the upload [Object](#)

500

Triggers if the database throws an error

POST /api/v1/file/download

Download several files (apiV1FileDownloadPost)

Triggers a download of a multiple files

Request body

body (required)

Body Parameter – Required data to download files

Produces

This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- application/zip

Responses

200

A ZIP file protected with the password 'infected'.

400

Triggers if no hashes was sent

GET /api/v1/file/

Gets a set of files from the database (**apiV1FileGet**)

Used for getting a specific number of files and searching in the database.

Query parameters

size (optional)

Query Parameter – The paginate size to get

index (optional)

Query Parameter – The paginate index to start from

text (optional)

Query Parameter – The string to search for in the database

Return type

array[[file](#)]

Example data

Content-Type: application/json

```
[ {
  "uploaded_by" : "uploaded_by",
  "date_added" : "date_added",
  "original_filename" : "original_filename",
  "pending_analysis" : "{}",
  "sha256" : "sha256",
  "size" : 0,
  "analyzed_info" : "{}",
  "tags" : [ "tags", "tags" ],
  "md5" : "md5"
}, {
  "uploaded_by" : "uploaded_by",
  "date_added" : "date_added",
  "original_filename" : "original_filename",
  "pending_analysis" : "{}",
  "sha256" : "sha256",
  "size" : 0,
  "analyzed_info" : "{}",
  "tags" : [ "tags", "tags" ],
  "md5" : "md5"
} ]
```

Produces

This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- application/json

Responses

200

Successful

500

If the database throws any error

GET /api/v1/file/{hash}/fileDownloads a file (**apiV1FileHashFileGet**)

Triggers a download of a specific file.

Path parameters**hash (required)***Path Parameter* – The hash of the requested file**Produces**

This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- application/zip

Responses

200

A ZIP file protected with the password 'infected'.

400

Triggers if no hashes was sent

GET /api/v1/file/{hash}Get data for a specific file (**apiV1FileHashGet**)

Used for grabbing metadata and analysis-date of a sample stored in the database

Path parameters**hash (required)***Path Parameter* – The hash of the requested file**Return type**[file](#)**Example data**

Content-Type: application/json

```
{
  "uploaded_by" : "uploaded_by",
  "date_added" : "date_added",
  "original_filename" : "original_filename",
  "pending_analysis" : "{}",
  "sha256" : "sha256",
  "size" : 0,
  "analyzed_info" : "{}",
  "tags" : [ "tags", "tags" ],
  "md5" : "md5"
}
```

Produces

This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- application/json

Responses

200

Successful [File](#)

404

If the file does not exist in the database

500

If the database throws any error

POST /api/v1/file/reanalyzeTriggers reanalysis of files ([apiV1FileReanalyzePost](#))

Triggers new analysis of a specific file using a specific analysis-source.

Request body**body (required)***Body Parameter* – Required data to trigger new analysis**Produces**

This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- application/json

Responses

200

Analysis triggered successfully

400

Triggers if no hash or source was sent

PUT /api/v1/file/tagsUpdating filetags ([apiV1FileTagsPut](#))

Lets admins and researcher update the tags of a file

Request body**body (required)***Body Parameter* – Required data to download files**Return type**

Object

Example data

Content-Type: application/json

```
"{}"
```

Produces

This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- application/json

Responses

200

Tags are updated. [Object](#)

400

Triggers if no hash or tags was sent

500

Triggers if MongoDB is having trouble updating the tags

POST /api/v1/file/upload

Uploads files to the repository ([apiV1FileUploadPost](#))

Accepts a single file, or multiple files

Consumes

This API call consumes the following media types via the Content-Type request header:

- multipart/form-data

Form parameters

file (required)

Form Parameter – The uploaded file data

Return type

Object

Example data

Content-Type: application/json

```
"{}"
```

Produces

This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- application/json

Responses

200

ID of the upload [Object](#)

400

Triggers if no files are sent to the route

User

GET /api/v1/user/favorite/count

Get favorite count ([apiV1UserFavoriteCountGet](#))

Used for getting the number of favorites a user have

Return type

Object

Example data

Content-Type: application/json

```
"{}"
```

Produces

This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- application/json

Responses

200

Successfull [Object](#)

500

Internal server error

GET /api/v1/user/favorite

Get favorite (**apiV1UserFavoriteGet**)

Used for getting tags for a user.

Query parameters

size (optional)

Query Parameter – The paginate size to get

index (optional)

Query Parameter – The paginate index to start from

Produces

This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- application/json

Responses

200

Successfull

500

Internal server error

PUT /api/v1/user/favorite

Set/remove favorite (**apiV1UserFavoritePut**)

Used for adding/removing a favorite for a user. It works as a toggle. If the hash already exist on the user it will remove it, otherwise it will add it.

Request body

body (required)

Body Parameter – Required data to set/remove favorite

Produces

This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- application/json

Responses

200

Successfull added/removeda favorite

400

Missing parameters

404

User does not exist

500

Internal server error

GET /api/v1/user

Get all users (**apiV1UserGet**)

Used for getting all the users. Require admin-access

Query parameters

size (optional)

Query Parameter – The paginate size to get

index (optional)*Query Parameter* – The paginate index to start from**Return type**array[[user](#)]**Example data**

Content-Type: application/json

```
[ {
  "authored" : "true",
  "favorites" : [ "favorites", "favorites" ],
  "role" : "student",
  "feide_id" : "5a91dghd2-118b-4e97-8b47-5d2gfhfjg39f",
  "name" : "Christian Simoes Isnes",
  "photo" : "https://api.dataporten.no/userinfo/v1/user/media/p:619f1215-5b3c-44cb-8754-e511bc382ec5",
  "groups" : [ "{}", "{}" ],
  "active" : true,
  "token" : "46h8e4jd-d232-4e56-91e9-3b3ee8365ca4",
  "tags" : [ "upload", "upload" ]
}, {
  "authored" : "true",
  "favorites" : [ "favorites", "favorites" ],
  "role" : "student",
  "feide_id" : "5a91dghd2-118b-4e97-8b47-5d2gfhfjg39f",
  "name" : "Christian Simoes Isnes",
  "photo" : "https://api.dataporten.no/userinfo/v1/user/media/p:619f1215-5b3c-44cb-8754-e511bc382ec5",
  "groups" : [ "{}", "{}" ],
  "active" : true,
  "token" : "46h8e4jd-d232-4e56-91e9-3b3ee8365ca4",
  "tags" : [ "upload", "upload" ]
} ]
```

Produces

This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- application/json

Responses

200

Successfull

500

Internal server error

DELETE /api/v1/user/{id}Delete user from the database (**apiV1UserIdDelete**)

Used for deleting all data about a user from the database

Path parameters**id (required)***Path Parameter* – Feide_id for the user**Produces**

This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- application/json

Responses

204
Successful deleted user
400
Missing parameters
404
User does not exist
500
Internal server error

GET /api/v1/user/{id}

Get data about a specific user (**apiV1UserIdGet**)

Used for grabbing data stored about a user from the database

Path parameters

id (required)

Path Parameter – Feide_id for the user

Return type

[user](#)

Example data

Content-Type: application/json

```
{
  "authored" : "true",
  "favorites" : [ "favorites", "favorites" ],
  "role" : "student",
  "feide_id" : "5a91dghd2-118b-4e97-8b47-5d2gfhfjg39f",
  "name" : "Christian Simoes Isnes",
  "photo" : "https://api.dataporten.no/userinfo/v1/user/media/p:619f1215-5b3c-44cb-8754-e511bc382ec5",
  "groups" : [ "{}", "{}" ],
  "active" : true,
  "token" : "46h8e4jd-d232-4e56-91e9-3b3ee8365ca4",
  "tags" : [ "upload", "upload" ]
}
```

Produces

This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- application/json

Responses

200

Successful [user](#)

400

If the request is missing feide_id as parameter

404

If the user does not exist in the database

500

If the database throws any error

PUT /api/v1/user/{id}

Edit data for a user (**apiV1UserIdPut**)

Used for editing data about a user in the database

Path parameters

id (required)

Path Parameter – Feide_id for the user

Request body

body (required)

Body Parameter – Required data to edit user

Produces

This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- application/json

Responses

204

Successful editing of a user

400

Missing parameters

404

User does not exist

500

Internal server error

GET /api/v1/user/profile

Authorizing a user (`apiV1UserProfileGet`)

Used for authorizing a user if the user is logged in. Will return user-object if user is serialized.

Return type

[user](#)

Example data

Content-Type: application/json

```
{
  "authored" : "true",
  "favorites" : [ "favorites", "favorites" ],
  "role" : "student",
  "feide_id" : "5a91dghd2-118b-4e97-8b47-5d2gfhfjg39f",
  "name" : "Christian Simoes Isnes",
  "photo" : "https://api.dataporten.no/userinfo/v1/user/media/p:619f1215-5b3c-44cb-8754-e511bc382ec5",
  "groups" : [ "{}", "{}" ],
  "active" : true,
  "token" : "46h8e4jd-d232-4e56-91e9-3b3ee8365ca4",
  "tags" : [ "upload", "upload" ]
}
```

Produces

This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- application/json

Responses

200

Response containing information of the user logged in. [user](#)

401

Triggers if the user is not authenticated

500

Triggers if an internal server error occurs.

Models

[[Jump to Methods](#)]

Table of Contents

1. [file -](#)
2. [user -](#)

file -

pending_analysis (optional)

[Object](#)

uploaded_by (optional)

[String](#)

tags (optional)

[array\[String\]](#)

sha256 (optional)

[String](#)

md5 (optional)

[String](#)

original_filename (optional)

[String](#)

size (optional)

[Integer](#)

date_added (optional)

[String](#)

analyzed_info (optional)

[Object](#)

user -

feide_id (optional)

[String](#)

example: 5a91dghd2-118b-4e97-8b47-5d2gfhfjg39f

name (optional)

[String](#)

example: Christian Simoes Isnes

photo (optional)

[String](#)

example: <https://api.dataporten.no/userinfo/v1/user/media/p:619f1215-5b3c-44cb-8754-e511bc382ec5>

token (optional)

[String](#)

example: 46h8e4jd-d232-4e56-91e9-3b3ee8365ca4

groups (optional)

[array\[Object\]](#)

active (optional)

[Boolean](#)

example: true

authored (optional)

[String](#)

example: true

role (optional)

[String](#)

example: student

tags (optional)

[array\[String\]](#)

favorites (optional)
[*array\[String\]*](#)

B.2 Worklog

Arbeidslogg i Timer	Michael	Gjert	Christian	Erlend	HVA DU HAR G.JORT	Michael	Gjert	Christian	Erlend
13.01.2021	5t	4t	6t		Leser opp på MongoDB og Mongooose		Oppsett av repository		
14.01.2021	5t				Leser opp på MongoDB og Mongooose				
15.01.2021	8t	5t	6t	6t	Leser opp på MongoDB og Mongooose	Virustotal - pipeline			Watched tutorials on node and mongooB
16.01.2021			5t						
17.01.2021		3t							
18.01.2021	6t		5t	5t	Jobbet med forprosjekt				Jobbet med forprosjekt
19.01.2021	5t		5t	6t	Jobbet med forprosjekt				Jobbet med forprosjekt
20.01.2021	5t	2t	6t	7t	Jobbet med forprosjekt	leser opp på node			Jobbet med forprosjekt
21.01.2021	7t		6t	6t	Jobbet med forprosjekt				Jobbet med forprosjekt
22.01.2021	6t	7t	7t	6t	LaTeX backup, testing	Test, db og models			Viledningsmøte, forprosjektarbeid
23.01.2021	7t	10t	7t	5t	Testing	models, pipeline, db			Forprosjektarbeid
24.01.2021	7t	5t	7t	5t	Ganit-schema	proggng			Forprosjektarbeid
25.01.2021	6t	2t	7t	6t	Fullført gantt, test testing	prosjektplan			Forprosjektarbeid, Study/MEAN stack
26.01.2021	7t	3t	7t	6t	Forprosjekt	prosjektplan			Forprosjektarbeid
27.01.2021	7t	4t	5t	8t	Forprosjekt	prosjektplan			Forprosjektarbeid
28.01.2021	7t	3t	6t	8t	Forprosjekt	prosjektplan			Forprosjektarbeid
29.01.2021	8t	6t	7t	6t	Forprosjekt	prosjektplan, test, services			Forprosjektarbeid, Front-end
30.01.2021									
31.01.2021			3t						
01.02.2021	4t	4t	8t	7t	passport,js and JWT's				Forprosjektarbeid, innloggingssystem
02.02.2021	10t	7t	7t	7t	JWT implementering	malwarebazaar			Error handling
03.02.2021	8t	7t	6t	7t	Kont. JWT implementering	pipeline, docker			Error handling, research
04.02.2021	8t	5t	5t	5t	Mer dynamisk auth.guard	pipeline, docker			Error handling
05.02.2021	8t	4t	5t	8t	Jobber med rapport, litt koding	docker			Arbeid med rapport
06.02.2021			6t						
07.02.2021	7t		5t		JWT's				
08.02.2021	8t	10t	5t	6t	Authorization	unzip			LAOS kurs, litt front-end
09.02.2021	9t	8t	8t	5t	Authorization	pipeline			Front-end, Winston logging research
10.02.2021	9t	6t	5t	5t	Documenting	pipeline			Prosjektskriving
11.02.2021	8t	7t	5t	5t	MongoDB reconnect	pipeline			Winston logging
12.02.2021		7t	5t	4t	Assignment in IMT4116	pipeline			Winston logging
13.02.2021					Assignment in IMT4116				
14.02.2021					Assignment in IMT4116				
15.02.2021			6t	2t	Assignment in IMT4116				
16.02.2021	8t	10t	8t	6t	File overview	pipeline, upload			LAOS kurs, winston-daily-rotate
17.02.2021	7t	6t	5t	7t	Implemented file overview	upload			Winston-daily-rotate logging
18.02.2021	9t	6t	6t	5t	Assignment in IMT4116, looking into pagination	fileclass			Winston-daily-rotate logging
19.02.2021	8t	6t	5t	6t	Expansion	fileclass			Pagination
20.02.2021			8t		Assignment in IMT4116				Pagination
21.02.2021			7t		Assignment in IMT4116				Refactored two UNINETT modules to support OAuth2 to OIDC (for MFA +++)
22.02.2021		5t	6t	6t	Assignment in IMT4116	pipeline			Overleaf documentation about everything related to FEIDE
23.02.2021	7t	6t	6t	6t	Misuse cases, Documenting	report writing, docker			Developed middleware basert på groups, id or all authenticated
24.02.2021	8t	4t			Sorting in file overview, begun locking into select	filepage			Jobb
25.02.2021	9t	6t	5t		Implemented detailed fileview	meeting			Jobb
									Combining all branches and auth_guard

26.02.2021	7 t	4 t	Information cards				User (de)serialization		
27.02.2021									
28.02.2021			FRI, Skitur			FRI	FRI		FRI, Skitur
01.03.2021			FRI, Skitur			FRI	FRI		FRI, Skitur
02.03.2021						FRI	FRI		FRI, Skitur
03.03.2021	5 t	6 t	Information cards cont.				User (de)serialization		websocket
04.03.2021	7 t	4 t	Information cards cont.				User (de)serialization		Rapportskriving
05.03.2021	8 t	4 t	Information cards cont.			unzip			Rapportskriving
06.03.2021		8 t				unzip			
07.03.2021							FRI		
08.03.2021	6 t	9 t	Meeting, started looking into 404-pages			virustotal rate limit	Redesigned serialization of users		Meeting, troubleshooting
09.03.2021	7 t	5 t	implemented 404-page				Redesigned serialization of users		Rapportskriving
10.03.2021	6 t	5 t	Documenting				Fileview organization		Websocket, logview
11.03.2021	9 t	6 t	Refresh virustotal front-end				Fileview organization		Websocket, logview
12.03.2021	7 t	5 t	Refresh added to frontend fileview						Websocket, logview
13.03.2021		6 t							
14.03.2021		4 t							
15.03.2021	7 t	10 t	Documenting			multiple file upload	Ironed out countless frontend bugs and structured file-view		
16.03.2021	7 t	5 t	Meeting, Documenting, logview			logview	Implementing favorites, added tags to usr+file, guarded routes		
17.03.2021	8 t	12 t	Div. fixes, rapportskriving			logview	Reworked user routes		Rapportskriving
18.03.2021	8 t	8 t	File upload frontend				Meeting and DeleteUser on admin panel		Rapportskriving
19.03.2021	7 t	6 t	Userlist pagination and sorting				Meeting and DeleteUser on admin panel		Rapportskriving
20.03.2021							EditUser on admin panel		Rapportskriving
21.03.2021							DeleteUser on admin panel		
22.03.2021		7 t							
23.03.2021	7 t	6 t	Selection file-list						Rapportskriving
24.03.2021	7 t	5 t	Documenting				Favorites		Rapportskriving
25.03.2021	8 t	6 t	Fixed selection and added mass-action to list				Favorites		Rapportskriving
26.03.2021	7 t	12 t	Started working on actions for selection, div. fixes			analyze	Passport		Rapportskriving
27.03.2021		12 t				analyze is now working	Parsing analysis data		
28.03.2021		5 t					Parsing analysis data		
29.03.2021	8 t	8 t	Checkbox functionality and added to user-list				Parsing analysis data		
30.03.2021		6 t	Assignment in IMT4116				Optimizing frontend		
31.03.2021		4 t	Assignment in IMT4116				Frontend og API		
01.04.2021		6 t	Assignment in IMT4116				Frontend og API		Rapportskriving
02.04.2021		9 t	Assignment in IMT4116				Frontend og API		Rapportskriving
03.04.2021		12 t	Assignment in IMT4116			upload			Rapportskriving
04.04.2021		8 t	Assignment in IMT4116			download			
05.04.2021		4 t	Assignment in IMT4116			download	Frontend og API		Rapportskriving
06.04.2021		8 t	Assignment in IMT4116			upload status	Frontend og API		Rapportskriving
07.04.2021		6 t	Assignment in IMT4116			fileclass	Frontend og API		Rapportskriving
08.04.2021	7 t	6 t	Charts and meeting			upload status	Frontend og API		Rapportskriving, Upload status
09.04.2021	8 t	8 t	Charts and Report writing				Frontend og API		Upload status
10.04.2021									
11.04.2021									
12.04.2021	6 t	8 t	Report writing			upload status	Report writing		Rapportskriving
13.04.2021	6 t	8 t	Report writing			upload status	Report writing		Rapportskriving

Appendix C

Code examples

Authorize

```
1  /**
2  * @desc Authorization-check for backend
3  * @params req, res, args (request, response and arguments)
4  * @res Will return true if authenticated/authorized, and send an immediate 401/403 if
   unauth.
5  */
6  module.exports.authorize = function (args) {
7    return (req, res, next) => {
8      let allowedRole = [];
9      let allowedTags = [];
10     let allowAll = false;
11
12     // Check if req.user is on request, and user is active
13     if (req.user && req.user.db.active) {
14       // Set 'all' from middleware
15       if (args.all !== undefined) {
16         allowAll = args.all;
17       }
18
19       // Set role from middleware
20       if (args.role !== undefined) {
21         allowedRole = args.role;
22       }
23
24       // Set tags from middleware
25       if (args.tags !== undefined) {
26         allowedTags = args.tags;
27       }
28
29       // User is authenticated, data set, start checking for authorization
30       exec();
31     } else {
32       // User did not have a session, or inactive, unauthenticated
33       return res.status(401).json({
34         message: "Unauthorized",
35       });
36     }
37
38     /**
39     * @desc Authorization-check for backend
40     * @params req, res, args (request, response and arguments)
41     * @res Will return true if authenticated/authorized, and send an immediate 401/403
       if unauth.
42     */
43     function exec() {
44       // Check if "any: true " is passed
45       if (allowAllAuthenticated()) {
46         return next();
47       }

```

```

48
49     // Check if users role is in the routes groups
50     else if (isUserRoleAllowed(req.user.db.role)) {
51         return next();
52     }
53
54     // Check if users tags is in the routes tags
55     else if (isAnyUserTagsAllowed(req.user.db.tags)) {
56         return next();
57     }
58
59     // User not authorized
60     else {
61         return res.status(401).json({
62             message: "Unauthorized",
63         });
64     }
65 }
66
67 /**
68  * @desc Check if specifictag is allowed
69  * @params userTag - Users tag to check
70  * @res Will return true if the users tag is in the list of allowed tags passed from
71  * middleware
72  */
73 function isTagAllowed(userTag) {
74     for (var i = 0; i < allowedTags.length; i++) {
75         if (allowedTags[i] == userTag) {
76             return true;
77         }
78     }
79     return false;
80 }
81
82 /**
83  * @desc Check if any tag is allowed
84  * @params userTags[]
85  * @res Will return true if any tags is in the list of allowed tags passed from
86  * middleware
87  */
88 function isAnyUserTagsAllowed(userTags) {
89     for (var i = 0; i < userTags.length; i++) {
90         if (isTagAllowed(userTags[i])) {
91             return true;
92         }
93     }
94     return false;
95 }
96
97 /**
98  * @desc Check if any role is allowed
99  * @params role[]
100  * @res Will return true if any role is in the whitelisted IDs passed to authorize.js
101  */
102 function isUserRoleAllowed(userRole) {
103     for (var i = 0; i < allowedRole.length; i++) {
104         if (allowedRole[i] == userRole) {
105             return true;
106         }
107     }
108     return false;
109 }
110
111 /**
112  * @desc Check if anyone is allowed
113  * @res Will return true if the "all" flag is passed
114  */
115 function allowAllAuthenticated() {
116     if (allowAll) {
117         return true;
118     }
119 };

```

Code listing C.1: The authorization middleware protecting the REST API
Passport strategy

```

1 const userService = require('../../../../db/user-service')
2 var openid = require('openid-client')
3 var passport = require('passport')
4 var util = require('util')
5 const request = require('request')
6 var User = require('./User').User
7
8 var OICStrategy = function(config) {
9   this.name = 'passport-openid-connect'
10  this.config = config || {}
11  this.groupsUrl = 'https://groups-api.dataporten.no/groups/me/groups'
12  this.client = null
13  this.tokenSet = null
14  this.init()
15    .then(() => {
16      console.log("Initialization of OpenID Connect discovery process completed.")
17    })
18 }
19 util.inherits(OICStrategy, passport.Strategy)
20
21 OICStrategy.prototype.init = function() {
22   if (!this.config.issuerHost) {
23     throw new Error("Could not find required config options issuerHost in
24       openid-passport strategy initialization")
25   }
26   return Promise.resolve().then(() => {
27     return openid.Issuer.discover(this.config.issuerHost)
28   })
29   .then((issuer) => {
30     this.client = new issuer.Client(this.config)
31   })
32   .catch((err) => {
33     console.error("ERROR", err);
34   })
35 }
36
37 OICStrategy.prototype.authenticate = function(req, opts) {
38   if (opts.callback) {
39     return this.callback(req, opts)
40   }
41   try {
42     var authurl = this.client.authorizationUrl(this.config)
43     this.redirect(authurl)
44   } catch (error) {
45     console.error("Error getting authUrl", error);
46     this.fail(error)
47   }
48 }
49
50
51 OICStrategy.prototype.getUserInfo = function() {
52   return this.client.userInfo(this.tokenSet.access_token)
53     .then((userinfo) => {
54       this.userinfo = userinfo
55     })
56 }
57
58 OICStrategy.prototype.loadGroups = function() {
59   var that = this; // Workaround for accessing
60   return new Promise(function(resolve, reject) {
61     'this' inside a subfunction, promise.
62     var options = {
63       url: that.groupsUrl,

```

```

64     headers: {
65         'User-Agent': 'passport-dataporten',
66         'Authorization': 'Bearer ' + that.tokenSet.access_token
67     }
68 };
69 // console.log("Performing OAuth 2.0 Request", options);
70 request(options, function (error, response, body) { // Send request to groups API
71     if (!error && response.statusCode == 200) {
72         var data = JSON.parse(body);
73         resolve(data); // Resolve with data
74     }
75     reject(error); // Reject if error
76 });
77
78 }).then(function(groups) {
79     that.groups = groups;
80     return groups;
81 });
82 };
83
84
85 OICStrategy.prototype.callback = function(req, opts) {
86     return this.client.callback(this.config.redirect_uri, req.query)
87         .then((tokenSet) => { // Get users basic information
88             this.tokenSet = tokenSet
89             return this.getUserInfo()
90         })
91         .then(() => { // Get users groups
92             return this.loadGroups()
93         })
94         .then(() => { // Set user object with user-data
95             var user = new User(this.userinfo)
96             user.token = this.tokenSet
97             user.idtoken = this.tokenSet.claims
98             user.groups = this.groups
99             this.success(user)
100         })
101         .catch((err) => {
102             console.error("Error processing callback", err);
103             this.fail(err)
104         })
105     }
106
107 OICStrategy.serializeUser = function(user, cb) {
108     userService.findUser(user.data.sub)
109     .then(usr => {
110         if (usr) {
111             cb(null, user.serialize());
112         } else {
113             userService.createUser(user)
114             .then(data => {
115                 cb(null, user.serialize());
116             })
117         }
118     })
119     .catch(err => {
120         this.fail(err)
121     })
122 }
123 OICStrategy.deserializeUser = function(packed, cb) {
124
125     userService.findUser(packed.data.sub)
126     .then(usr => {
127         packed.db = usr
128         cb(null, User.unserialize(packed))
129     })
130
131 }
132 }
133
134 exports.Strategy = OICStrategy
135

```

Code listing C.2: Passport strategy handling user-logins

Auth_service

```

1 import { Injectable, EventEmitter, Output } from "@angular/core";
2 import {
3   HttpClient,
4   HttpHeaders,
5   HttpClientModule,
6 } from "@angular/common/http";
7 import { AuthData } from "../auth-data.model";
8 import { Router } from "@angular/router";
9 import { User } from "../models/user";
10 import { catchError, map } from "rxjs/operators";
11 import { BehaviorSubject, of } from "rxjs";
12 import { Subject } from "rxjs";
13 import { TmplAstRecursiveVisitor } from "@angular/compiler";
14
15 const httpOptions = {
16   headers: new HttpHeaders({
17     mode: "no-cors",
18   }),
19 };
20
21 @Injectable({ providedIn: "root" })
22 export class AuthService {
23   authStatusListner = new Subject<boolean>();
24   isAuthenticated = false;
25   public user;
26   public subscription$;
27
28   constructor(private http: HttpClient, private router: Router) {}
29
30   getAuthStatusListner() {
31     return this.authStatusListner.asObservable();
32   }
33
34   getUserProfile() {
35     this.http
36       .get("/api/v1/user/profile") // is not, pull profile
37       .subscribe(
38         (user: any) => {
39           if (user && user.active && user.authored) {
40             // check if user exist on session and is active
41             this.user = user;
42             this.authStatusListner.next(true); // permit user
43           } else {
44             if (!user.active || !user.authored) {
45               // check if user is not active
46               this.authStatusListner.next(false);
47             } else {
48               this.authStatusListner.next(false);
49             }
50           }
51         },
52         (error) => {
53           // all unknown errors redirect to login, typical for user does not exist
54           // in db etc
55           console.log("User is not authenticated, redirecting.");
56           this.router.navigate(["/login"]);
57         }
58       );
59   }
60
61   updateAuthenticated(status: boolean) {
62     console.log("Updating isAuthenticated to, ", status);
63     this.authStatusListner.next(status);
64   }
65
66   public signout() {
67     this.updateAuthenticated(false);
68   }
69 }

```

Code listing C.3: Auth Service Angular

AuthGuard

```

1 import { Injectable } from "@angular/core";
2 import {
3   ActivatedRouteSnapshot,
4   CanActivate,
5   Router,
6   RouterStateSnapshot,
7 } from "@angular/router";
8 import { AuthService } from "../auth.service";
9 import { Subscription } from "rxjs";
10 import {
11   HttpClient,
12   HttpHeaders,
13   HttpClientModule,
14 } from "@angular/common/http";
15 import { Console } from "console";
16 import { getMatInputUnsupportedTypeError } from "@angular/material/input";
17
18 @Injectable({
19   providedIn: "root",
20 })
21 export class AuthGuard implements CanActivate {
22   userIsAuthenticated = false;
23   private authListenerSubs: Subscription;
24   constructor(
25     private authService: AuthService,
26     private router: Router,
27     private http: HttpClient
28   ) {}
29
30   canActivate(
31     route: ActivatedRouteSnapshot,
32     state: RouterStateSnapshot
33   ): Promise<boolean> {
34     return new Promise((resolve) => {
35       this.authListenerSubs = this.authService
36         .getAuthStatusListener()
37         .subscribe((isAuthenticated) => {
38           this.userIsAuthenticated = isAuthenticated;
39         });
40
41       if (this.userIsAuthenticated) {
42         // Checking if global var isAuth is set
43         resolve(true);
44       }
45       // is not, pull profile
46       else {
47         this.http
48           .get("/api/v1/user/profile")
49           .subscribe(
50             (user: any) => {
51               // check if user exist on session and is active
52               if (user && user.active && user.authored) {
53                 this.authService.user = user;
54                 console.log("Auth guard: User authenticated");
55                 this.authService.authStatusListener.next(true);
56                 resolve(true); // permit user
57               } else {
58                 // check if user is not active
59                 if (!user.active || !user.authored) {
60                   console.log("User must talk to an admin");
61                   // redirect to notactive to tell user to talk to an admin
62                   this.router.navigate(["/accessdenied"]);
63                   this.authService.authStatusListener.next(false);
64                   resolve(false);
65                 }
66                 // user profile does not exist on session, redirect to login
67                 else {
68                   console.log("User is not authenticated, redirecting...");
69                   this.authService.authStatusListener.next(false);
70                   this.router.navigate(["/login"]);
71                   resolve(false);
72                 }
73             }
74           );
75     }
76   }
77 }

```

```

73     },
74     },
75     (error) => {
76         // all unknown errors redirect to login, typical for user does
77         // not exist in db etc
78         console.log("User is not authenticated, redirecting.");
79         this.router.navigate(["/login"]);
80     }
81 });
82 });
83 }
84 }

```

Code listing C.4: AuthGuard Angular

AdminGuard

```

1
2 import {
3     CanActivate,
4     ActivatedRouteSnapshot,
5     RouterStateSnapshot,
6     Router,
7 } from "@angular/router";
8 import { Injectable } from "@angular/core";
9 import { Subscription } from "rxjs";
10 import { HttpClient, HttpHeaders, HttpClientModule } from '@angular/common/http';
11 import { AuthService } from "./auth.service";
12
13 @Injectable()
14 export class AdminGuard implements CanActivate {
15     user
16     userIsAuthenticated = false
17     private authListenerSubs: Subscription
18     constructor(private authService: AuthService, private router: Router, private http:
19         HttpClient) {}
20
21     canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot): Promise<boolean> {
22         return new Promise((resolve) => {
23             this.authListenerSubs = this.authService.getAuthStatusListener()
24             .subscribe(isAuthenticated => {
25                 this.userIsAuthenticated = isAuthenticated
26             })
27
28             if(this.userIsAuthenticated) { //
29                 // Checking if global var isAuth is set
30                 resolve(true)
31             } else {
32                 this.http.get('/api/v1/user/profile') //
33                 // is not, pull profile
34                 .subscribe((user : any) => {
35                     if (user && user.active && user.authorized ) { //
36                         // check if user exist on session and is active
37                         this.authService.user = user
38                         this.authService.authStatusListener.next(true);
39                         if(user.role == 'admin') {
40                             resolve(true)
41                         } else {
42                             this.router.navigate([''])
43                             resolve(false)
44                         } // permit user
45                     } else {
46                         if(!user.active || !user.authorized) { //
47                             // check if user is not active
48                             console.log('User must talk to an admin')
49                             this.router.navigate(['/accessdenied'])
50                             // redirect to notactive to tell user to talk to an admin
51                             this.authService.authStatusListener.next(false);
52                             resolve(false);
53                         } else {

```

```

48 |             console.log("User is not authenticated, redirecting...") //
49 |                 user profile does not exist on session, redirect to login
50 |             this.authService.authServiceListener.next(false);
51 |             this.router.navigate(['/login'])
52 |             resolve(false);
53 |         }
54 |     }, error => {
55 |         console.log("User is not authenticated, redirecting.") //
56 |             all unknown errors redirect to login, typical for user does not
57 |             exist in db etc
58 |         this.router.navigate(['/login'])
59 |     });
60 | }
61 |

```

Code listing C.5: AdminGuard Angular

LoginGuard

```

1 |
2 | import {
3 |     CanActivate,
4 |     ActivatedRouteSnapshot,
5 |     RouterStateSnapshot,
6 |     Router,
7 | } from "@angular/router";
8 | import { Injectable } from "@angular/core";
9 | import { AuthService } from "../auth.service";
10 | import { HttpClient, HttpHeaders, HttpClientModule } from '@angular/common/http';
11 |
12 |
13 | @Injectable()
14 | export class LoggedInGuard implements CanActivate {
15 |     user
16 |     constructor(private router: Router, private AuthService: AuthService, private http:
17 |         HttpClient) {}
18 |
19 |     canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot): Promise<boolean>
20 |     {
21 |         return new Promise((resolve) => {
22 |             this.http.get('/api/v1/user/profile') // is
23 |                 not, pull profile
24 |             .subscribe((user : any) => {
25 |                 console.log('User is authenticated, redirect to index')
26 |                 this.router.navigate([''])
27 |                 resolve(false)
28 |             }, error => {
29 |                 console.log("User is not authenticated, stay.") // all
30 |                     unknown errors redirect to login, typical for user does not exist
31 |                     in db etc
32 |                 resolve(true)
33 |             });
34 |         });
35 |     }
36 | }

```

Code listing C.6: LoginGuard Angular

FileFilter

```

1 | /**
2 | * @desc Authorization-check for backend

```

```

3  * @params req, res, args (request, response and arguments)
4  * @res Will return true if authenticated/authorized, and send an immediate 401/403 if
   unauth.
5  */
6  module.exports.filefilter = function (files, user) {
7    return new Promise((resolve, reject) => {
8
9      exec();
10
11     /**
12     * @desc Remove files if
13     * @params req, res, args (request, response and arguments)
14     * @res Will return true if authenticated/authorized, and send an immediate 401/403
       if unauth.
15     */
16     function exec() {
17       // Check all files in the array
18       files.forEach(file => {
19         // Check if student and exam tag is set, and if file is private
20         if(isStudent(file) || isPrivate(file)) {
21           // Remove that file from array if user should not have access
22           files.splice(files.map(function(e){return
               e.sha256}).indexOf(file.sha256), 1)
23         }
24       })
25       resolve(files)
26     }
27
28     /**
29     * @desc Check if user is student, and if file contains student-limiting tag such as
       "exam"
30     * @params file - File to check user against
31     * @res Will return true if user is student and file has exam tag. Else false
32     */
33     function isStudent(file) {
34       if(user.db.role == 'student' && file.tags.includes('exam')) {
35         return true
36       } else {
37         return false
38       }
39     }
40
41     /**
42     * @desc Check if any tag is allowed
43     * @params userTags[]
44     * @res Will return true if any tags is in the list of allowed tags passed from
       middleware
45     */
46     function isPrivate(file) {
47       if(file.tags.includes('private') && !user.db.name == file.uploaded_by) {
48         return true
49       } else {
50         return false
51       }
52     }
53   })
54 };

```

Code listing C.7: Filefilter to remove files user is unauthorized for

