

Johannes Madsen Barstad
Odin Korsfur Henriksen
Jonas Kjærandsen
Peder Andreas Stuen

NTNU Threat Total

A Self-Service Threat Intelligence Solution

Bachelor's thesis in Bachelor in Digital Infrastructure and Cyber Security

Supervisor: Espen Torseth

May 2022

Johannes Madsen Barstad
Odin Korsfur Henriksen
Jonas Kjærandsen
Peder Andreas Stuen

NTNU Threat Total

A Self-Service Threat Intelligence Solution

Bachelor's thesis in Bachelor in Digital Infrastructure and Cyber
Security
Supervisor: Espen Torseth
May 2022

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

SAMMENDRAG

Tittel:	NTNU Threat Total	Dato:	19.05.2022
<hr/> <hr/> <hr/>			
Deltakere:	Johannes Madsen Barstad		
	Odin Korsfur Henriksen		
	Jonas Kjærandsen		
	Peder Andreas Stuen		
Veileder:	Espen Torseth		
Oppdragsgiver:	NTNU SOC		
Stikkord eller nøkkelord:	Full stack, webteknologier, cybersikkerhet, trussel-etterretning, selvhjelpsløsning		
Antall sider og ord: 72 sider, 19764 ord.	Antall vedlegg: 8	Publiseringsavtale inngått: Ja	
Kort beskrivelse av bacheloroppgaven:			
<p>Threat Total er en full stack web applikasjon med et brukervennlig grensesnitt, som bruker NTNUs farger og logoer. Applikasjonens backend er en robust Golang-server med caching i Redis, som reduserer lasten på våre tredjeparts datakilder i tillegg til forespørselstiden. Tråder er også brukt for å redusere prosesseringstiden på forespørsler hvor det er hensiktsmessig. På grunn av integrering med Feide autentisering er det å logge inn og ut like enkelt som med andre NTNU-tjenester. Dette gjør Threat Total til en sømløs brukeropplevelse for NTNU ansatte og studenter.</p>			

ABSTRACT

Title:	NTNU Threat Total	Date:	19.05.2022
<hr/> <hr/> <hr/>			
Participants:	Johannes Madsen Barstad		
	Odin Korsfur Henriksen		
	Jonas Kjærandsen		
	Peder Andreas Stuen		
Supervisor:	Espen Torseth		
Employer:	NTNU SOC		
Keywords:	Full stack, web application, cyber-security, threat-intelligence, self-help		
Number of pages and words: 72 pages, 19764 words.	Number of appendixes: 8	Availability: Open	
Short description of the bachelor thesis:			
<p>Threat Total is a full stack web application with a user-friendly frontend using NTNU branding colors and logos. The backend is a robust Golang application with caching in Redis which reduces the load on our third-party data providers as well as the request time for our users. Threading is also used to reduce the processing time for requests where appropriate. Due to integration with Feide authentication logging in and out is as simple as with any of the NTNU services, which makes using Threat Total a seamless user experience for members of NTNU staff and students.</p>			

Preface

The authors of this bachelor thesis, Johannes Madsen Barstad, Odin Korsfur Henriksen, Jonas Kjærandsen and Peder Andreas Stuen, would like to thank our supervisor, Espen Torseth, for the collaboration during this project period. He helped us in the form of guidance, discussion, and advice throughout the project. The weekly scheduled meetings with Espen helped us to keep the spirit up in tough times, as well as guided us in the right direction when questions arose. This was really appreciated by all the group members.

We would also like to thank our employer, NTNU SOC with our contact persons Christoffer Vargtass Hallstensen and Frank Wikstrøm. We genuinely think that this project has been a great learning experience for us, as well as a fun way of learning new tools and how full stack web development is conducted in a real-life setting. We would also like to thank our advisor Christoffer and Frank for guidance throughout the project, as well as interesting and great conversations. A thank you should also be given to NTNU SOC for entrusting us with this task, with the hopes that this report will justify our implementation.

Table of Contents

Preface.....	i
Table of Figures	vi
Table of Tables	vii
Glossary	viii
1 Introduction.....	1
1.1 Task Information.....	1
1.1.1 Problem area.....	1
1.1.2 Scope	1
1.2 Target Group	2
1.3 Internal Background and Competence	3
1.4 Time Frames.....	4
1.5 Roles and Responsibilities	5
1.6 Report Structure	6
2 Theory	8
2.1 Introduction	8
2.2 Background and Purpose of the Project	8
2.3 Research Methodology.....	9
2.4 Full Stack Web Development	9
2.4.1 Frontend responsive web design	9
2.4.2 Interaction design	10
2.4.3 Networking operations	11
2.4.4 Testing	11
2.4.5 Authentication and authorization.....	12
2.4.6 Security in software development	13
2.5 The Scrum Model.....	14
3 Requirements Specification	16
3.1 Introduction	16
3.2 Functional Requirements.....	16
3.3 Non-Functional Requirements	17
3.4 Use Cases	18
3.4.1 Use case 1 – Log in.....	18

Prelude

3.4.2 Use case 2 – Investigate URL.....	20
3.4.3 Use case 3 – Investigate file hash.....	21
3.4.4 Use case 4 – Investigate file	22
3.4.5 Use case 5 – Log out.....	23
4 Design and Technologies.....	24
4.1 Introduction	24
4.2 Prerequisites	24
4.3 Design Decisions.....	25
4.3.1 Frontend.....	25
4.3.2 Backend	26
4.3.3 Caching	26
4.4 From Idea to Sketch	27
4.5 Wireframes and Prototyping	28
4.6 Wireframe Validation.....	28
4.7 User Interface, the End Product	29
5 Development Process.....	30
5.1 Introduction	30
5.2 Scrum	30
5.3 An Alternative Development Framework to Scrum	31
5.4 Our Development Framework.....	31
5.5 Development Environment and Tools Used	32
5.6 Summary of the Scrum Sprints	33
6 Implementation and Production Process.....	35
6.1 Introduction and an Overview of the Application Structure.....	35
6.2 Intelligence Sources	36
6.2.1 Google safe browsing.....	36
6.2.2 Hybrid analysis.....	37
6.2.3 AlienVault	38
6.2.4 VirusTotal.....	39
6.3 Login Functionality.....	40
6.4 URL and Domain Search	41
6.5 File Hash Search.....	44

Prelude

6.6 File Upload Process.....	47
6.7 Escalate to Manual Analysis	48
6.8 Frontend Structure.....	48
6.9 Translation.....	49
7 Code Review and Code Quality.....	50
7.1 Introduction	50
7.2 Component Structure.....	50
7.3 Increasing Performance with Threading	52
7.4 Efficient Code with Caching.....	53
7.5 Code Modularity	54
8 Testing and Quality Assurance	55
8.1 Introduction	55
8.2 Static Code Analysis	55
8.3 SonarQube.....	56
8.4 Usability Testing	58
8.4.1 Results From Usability Testing	59
8.4.2 Weaknesses.....	59
8.5 Regression Testing	60
8.6 Destructive Testing	61
8.7 Unit and API Testing.....	62
8.8 Logging in Threat Total	64
8.9 Known Vulnerabilities	66
8.9.1 Potential RCE vulnerability in “file.filename”.....	66
8.9.2 Potential malicious downloads from screenshot functionality	66
9 Installation and Realisation.....	67
9.1 Introduction	67
9.2 Installation.....	67
10 Ending Chapters.....	68
10.1 Introduction.....	68
10.2 Results.....	68
10.3 Task Critiques	68
10.3.1 Internal API access.....	68

Prelude

10.3.2 Worklog time tracking.....	69
10.3.3 Sprint durations.....	69
10.3.4 Automated testing.....	70
10.3.5 User testing and usability testing.....	70
10.4 Future Work and Development.....	70
10.5 Evaluation.....	71
10.5.1 Organization.....	71
10.5.2 Work distribution.....	71
10.5.3 Project as a form of work.....	71
10.6 Conclusion.....	72
References.....	xi
Appendices:.....	xiv

Table of Figures

Figure 1: CSS media query example	10
Figure 2: An example of a scrum board, screenshot from Jira Atlassian tool	14
Figure 3: General overview of the Full stack application, made in draw.io	24
Figure 4: Example snippet of navbar component	25
Figure 5: Example snippet of how the Navbar component is included in homepage.js	26
Figure 6: Example of how connection to the Redis pool was implemented.....	27
Figure 7: First wireframe iteration of the index / homepage in balsamiq.....	28
Figure 8: The end product.....	29
Figure 9: Detailed overview of the application.....	35
Figure 10: Feide authentication implementation	40
Figure 11: Example of investigating a URL	41
Figure 12: Golang data structure for hybrid analysis.....	42
Figure 13: How the user response is made based on verdict from google safe browsing	42
Figure 14: Function which sets response based on source status	43
Figure 15: Function for checking if the process of gathering threat intelligence is finished.....	43
Figure 16: Example of how the result page could look like	44
Figure 17: Data parsing for Hybrid Analysis file hash search.....	45
Figure 18: “Setresulthash” will set the user response based on verdict from antivirus agent	45
Figure 19: File hash search results.....	46
Figure 20: File upload pipeline	47
Figure 21: Frontend component sharing	48
Figure 22: Example of a button being translated with json data.....	49
Figure 23: Component structure, made in draw.io	50
Figure 24: Image of structure, made in draw.io.....	51
Figure 25: Visualization of threading, made in draw.io	52
Figure 26: cache miss vs cache hit. Visualization made in draw.io	53
Figure 27: Example of code before revision.....	54
Figure 28: Example of code after revising code modularity.....	54
Figure 29: Example of SonarQube report.	56
Figure 30: Example of a detected security flaw.....	57
Figure 31: Regression testing flowchart - Made in draw.io.....	60
Figure 32: Logerror function	65
Figure 33: Loginfo function.....	65

Table of Tables

Table 1: Common http request methods	x
Table 2: Common http request methods	11
Table 3: Use case 1 - Log in	18
Table 4: Use case 1 - Basic flow.....	18
Table 5: Use case 1 - Alternative flow.....	19
Table 6: Use case 2 - Investigate url.....	20
Table 7: Use case 2 - Basic flow.....	20
Table 8: Use case 3 - Investigate file hash.....	21
Table 9: Use case 3 - Basic flow.....	21
Table 10: Use case 4 - Investigate file	22
Table 11: Use case 4 - Basic flow.....	22
Table 12: Use case 5 - Log out	23
Table 13: Use case 5 - Basic flow.....	23
Table 14: Programs and tools used.....	32
Table 15: API endpoints	36
Table 16: Usability testing	59
Table 17: Test functions.....	64

Glossary

Agile: An iterative approach to project management and software development. Consists of smaller increments, allowing to respond quickly to changes in a project.

API: Acronym of Application Programming Interface, is a connection between applications, typically used to aid explaining how requests between applications are to be handled.

CI: Continuous Integration, refers to the automation process for developers, in which is designed to automatically integrate code changes and updates from several team members during software development. [38]

Client: An application that is available to the end user, such as a web-browser, typically runs locally on a user's device.

Client side: Refers to the operations performed by a client. Typical operations are to request pages from the server, to display on a client. [24]

CPU: Central Processing Unit is the “brain” of the computer and takes instructions from a program or application and performs calculations based on these.

CSS: Cascading Style Sheet is a styling language which is used in accommodation with HTML and provides different shapes, animations, colours, fonts, and layout. CSS is a standardized language across the entire World Wide Web for developing design. [8]

Domain: Is a name that identifies a part of the internet and is usually categorized with a domain name, or a URL.

Feide: “Felles elektronisk identitetshåndtering” (common electronic identity management) is a centralized identity management solution developed for the Norwegian educational sector. Feide provides the user with SSO (Single Sign On) and SLO (Single Log Out) functionalities. [9]

File-hash: A long string of characters that works like the fingerprint of a file.

Full stack web development: Refers to web development where both client-side and server-side code is being developed. It is up to the developers to connect these two sides and make them communicate. [26] [27]

Gin-gonic: Is a Golang HTTP web framework with better performance than the ordinary HTTP library which comes prebuilt in with Golang. [10]

GitLab: Is a platform which is used by organizations, NTNU among others, for software development in a collaborative manner. GitLab provides the functionality with planning, building, securing, and deploying software. [2]

Golang: Is a programming language that was created at and for Google in 2007. Golang is a good language for server-side coding as it has many built in functionalities and libraries for making web servers and communication over HTTP.

HTML: Hyper Text Markup Language is the standard markup language to be displayed in a web browser and defines the structure and meaning of the website's content. [14]

IOC: Indicator of Compromise is an artifact that enable information security professionals and system administrators to better detect malicious activities inside the network. [16]

IP: Internet Protocol works like an identifier or name for different services, host, and clients on the internet. The IP address consist of four octets which represents the device or service. An example is Google's DNS server and has the IP address, 8.8.8.8.

JavaScript: is a widely used programming language used for web development, especially when it comes to frontend development.

Man-in-the-middle attack: An attack where the attacker sits in the middle of traffic, intercepts, and changes the traffic. This can be an issue if dealing with unencrypted traffic as the attacker can then plainly see the traffic and change it. Can be mitigated using encrypted data traffic through for example TLS and by using other confirmation methods such as checksums.

Networking snooping attacks: An attack where an attacker observes local network traffic and reads the data sent over the network. Can be mitigated using encrypted data traffic through for example TLS.

OS: Operating system which allocates resources to each individual component and provides the user with a graphical user interface to interact with the computer. Windows is an example of an operating system.

RAM: Random Access Memory is a type of computer memory which stores information that different programs and processes need during execution of them. This memory can be accessed in any desired order and is why it is called random access. [15]

RCE vulnerability: Remote Code Execution vulnerability allows an external attacker to execute arbitrary code on a remote system or device. The impact of this can range between executing malware remotely, to an attacker gaining full access over a system. [40]

ReactJS: Is a web development library, made for JavaScript and provides the functionality with making "React Components". These work as templates when making different pages within the web application.

Redis: Is an open source, in-memory data storage service which is used as a cache service among other things. [20]

REST API: Representational State Transfer Application Programming Interface uses REST's architectural style and allows for interaction with web services that is RESTful. REST APIs allow the following most widely used methods for transferring data [25]:

Method	Description
GET	Retrieves information from the API.
POST	Sends data and updates the API.
PUT	Adds data to the API.
DELETE	Deletes data from the API.

TABLE 1: COMMON HTTP REQUEST METHODS

Reverse proxy: A reverse proxy is a proxy on the host side. It sits in front of the backend services and routes the requests to them. To the client it appears as the request data comes from the reverse proxy itself. A reverse proxy can be set up to provide additional reliability and security by for example implementing https on the traffic it routes, and by distributing the requests to several backends therefore balancing the load.

Server: A computer or system with its role being providing resources to other clients.

Server side: Refers to the operations performed by a server. Typical operations are to serve pages that are requested by the client. [24]

TLS: TLS stands for transport layer security and is a set of protocols which uses cryptographic functions to secure data transfer.

URL: Uniform Resource Locator is a reference to a web resource somewhere on the Internet and is usually written in the search bar of a browser.

1 Introduction

The introduction chapter will discuss matters such as the definition and information about the task in hand, the target group for our final product, as well as the report itself. It will also provide some information about the group member's professional background and competence. Time frames when it comes to project organization and how to use the time that we were given in a reasonable fashion. Roles and responsibilities regarding our supervisor and contact persons in NTNU SOC, and general information about the report, such as the layout with focus on the main chapters, font and styling is included in this chapter.

1.1 Task Information

The following subchapters will provide task related information such as the problem area, scope, target group, and internal background and competence. The code is publicly available at the following GitLab repository: <https://git.gvk.idi.ntnu.no/Johannesb/dcsg2900-threattotal>

1.1.1 Problem area

In a world that is constantly evolving, especially when it comes to digitalization and the threats that comes with it, it is important to have established a defence and a way to warn if an incident or attack occurs. This brings us to the purpose of this task, which is to create a web application for NTNU's Security Operations Centre (NTNU SOC) which in essence will lessen their workload in an otherwise hectic day to day job. NTNU SOC is an emergency response function when it comes to cyber incidents and is a part of the Digital Security Section at NTNU IT. The web application will allow all authorized "NTNU'ers" to submit URLs, domains, file hashes and files that seems suspicious for either automatic analysis or if the user wants to, the choice of escalating the case into a manual analysis by the SOC which is built into the application.

1.1.2 Scope

The scope of this bachelor thesis is to develop the application "Threat Total" for NTNU's SOC department, which will be used to check URLs, Domains, and file-hashes against several databases of IOCs, both public and private.

Our scope is to develop a user-friendly application which will be used by NTNU students and employees to lookup if a URL, domain, or file hash is malicious or not. We will develop the website to support both English and Norwegian. The application will use publicly available reputation sources and hash databases, as well as NTNU's private reputation database to check if the domain, URL, or file hash has a reputation of being malicious. The application will be utilizing a REST API to communicate and access data from NTNU SOC. Communication with

Introduction

public available anti-virus agents will also be done through REST APIs. The website application will also be utilizing the Feide portal login system and will be able to retrieve contact information about the current logged in user. This will be used whenever a hash, domain or URL is unknown for NTNU, it will be possible to create a case for NTNU SOC. Security analysts can then further check it out, analyse the case, and proceed with further communication with the user.

We will develop the backend in Golang, and the front-end will be developed using JavaScript, HTML and CSS for styling. To increase the efficiency of our application it will also be important to look at implement support for caching to store the most recently searched domains, URLs, and file hashes.

Information about functional and non-functional requirements, use cases and user stories can be found under the chapter, “Requirements specification”. These specifications will provide information about what needs to be in place for the application to achieve the original goal of the task, as well as overall good usability.

1.2 Target Group

Web application:

- The web application, which is our final product, can be used by all authorized members of NTNU as it requires authorization with Feide. This includes all students, teachers and scientists that are members of NTNU and have a valid NTNU email, as well as a valid NTNU account. It is stated in the requirements for the task that the user interface must be as simple as possible. All kind of users with different technological competence will hence be able to use our application.

Report:

- The report is intended for our supervisor, contact persons at NTNU SOC, as well as curious students as this report will be publicly available at NTNU Open. This report will thus also be a source for inspiration for future bachelor students which study similar topics as us. Internal and external exam sensors who will grade our bachelor thesis base the final grade on this report. These people will also be amongst the target group of the report.

1.3 Internal Background and Competence

The team consist of four students who attend the bachelor's program, Digital Infrastructure and Cyber Security, DIGSEC for short. All the members have competence in the following relevant fields of study and is thereby also well qualified for this type of task:

A big part of the bachelor's program itself is *programming* in general. We have experienced both programming languages associated with web development, such as JavaScript, HTML and CSS and lower-level languages such as C and C++, as well as languages for backend programming such as Golang which was also included as a part of the bachelor's program.

The team have good experience with establishment of *digital infrastructure*. By using the IaaS solution, OpenStack, Virtual Machines, as well as the containerization service, Docker, the team feels that this field of study is under control.

Networking is fundamental part of any IT related studies and should therefore be common knowledge. The team have the experience of two courses which are directly related to the topics of networking and network security. This knowledge will come in handy as it will be used during the application development.

REST API's will be used throughout the entire application and knowledge about this topic is essential. Courses on this topic as well as learning-by-doing methodology will make sure that the competence here is sufficient.

Information security is a big part of the bachelor's program. With each course through the years, the principal of security has been implemented in one way or another. Our mindset around information security and securing of the application is therefore good. User authorization, securing of endpoints and a secure handling of malicious files will be implemented accordingly.

One of the first courses at NTNU revolved around *software development* together with different types of *development methodologies* and kinds of *testing*. The team holds the basic knowledge on this field. Finding the right development methodology and testing methods will therefore be easier.

Collaborative project work is a big part of the IT world itself. It is therefore also natural that this topic has been a huge part of the bachelor's program. Our knowledge in experience on this topic will contribute positively during this project.

Backend and frontend technologies, and tools are essentials that the team needs to learn more about and get familiar with. Despite us having the fundamental knowledge, we will be using different frameworks and libraries regarding the chosen programming languages. These offer different functionalities from the basic tools in the language. Together with these, there will be other dependencies which will work in collaboration with the main application which the team will have to get familiar with. Knowledge in these fields will be learned along the way, both in the form of self-study and in the principle of learning-by-doing.

1.4 Time Frames

We were originally given a total time of a little over four months for the main project and a time frame and management plan had to be set up to better divide the give time into different activities. To manage the time, we chose to use a Gantt chart, as well as sprints according to our software development methodology. Our Gantt chart is divided into 17 different categories, each with its own timeframe, and milestones for more significant and important tasks. These milestones are respectively:

- Project plan delivery
- Wireframe GUI TT
- Threat Total Application Prototype
- Threat Total 1.0 release
- Bachelor thesis delivery

We have taken the advantage of Jira from Atlassian which is a software for agile project management. By using this tool, we were able to make weekly sprints with tasks that should be finished within the week of the sprint. Tasks are firstly written on a Kanban board, then assigned to one of the team members. The tasks start in the “TO DO” section of the board and moves on to the “IN PROGRESS” section as soon as they are assigned to a member of the team. Once the task is done, it gets moved into the “DONE” section of the Kanban board. After the week is done, the sprint gets terminated and all the group members have an internal discussion about the following week’s tasks and goals. These tasks and goals will then be transferred into the new sprint in addition to new tasks.

1.5 Roles and Responsibilities

Information about employer and supervisor:

- Employer: NTNU SOC
- Contact person: Christoffer Vargtass Hallstensen, Group leader SOC
christoffer.hallstensen@ntnu.no
- Contact person: Frank Wikstrøm, Security analyst
frank.wikstrom@ntnu.no
- Supervisor: Espen Torseth, Senior advisor
espen.torseth@ntnu.no

The team members of the project:

- Johannes Madsen Barstad, Project Leader
johanmba@stud.ntnu.no
- Odin Korsfur Henriksen
odinkh@stud.ntnu.no
- Jonas Kjærandsen
jonakj@stud.ntnu.no
- Peder Andreas Stuen
pederas@stud.ntnu.no

To have a final say in tough decisions we decided to choose Johannes Madsen Barstad as our group leader. His responsibility lies in making the final decision when problems arise. In addition to this, he is responsible for maintain communication with the supervisor. The group leader is also our main connection to our contact persons at the NTNU SOC, which is our employer.

We have also assigned Peder Andreas Stuen with the main responsibility for the report. With responsibilities such as making sure that the quality of it is sufficient in accordance with our goals, and generally to make sure that the progress on our report is following the goals in the Gantt chart. His responsibility is also to make sure that the report is delivered on time, and that both the team, and our supervisor is satisfied before delivery.

Jonas Kjærandsen and Odin Korsfur Henriksen's main jobs are to develop the application we are assigned to make. This includes backend and frontend coding, integrating multiple REST API's and researching and implementing different tools, libraries and techniques which could be of use in the project. Barstad and Stuen will also contribute in this part as it is the basis of the bachelor's project.

1.6 Report Structure

The report is divided into ten main chapters with several sub-chapters. Chapters in this report are as follows in chronological order:

1. Introduction
2. Theory
3. Requirements specification
4. Design and Technologies
5. Development process
6. Implementation and production process
7. Code review and code quality
8. Testing and quality assurance
9. Installation and realization
10. Ending chapters

The *introduction* chapter contains information such as the definition of our given task and a description of how we should use our timeframe. Information about competence within the team, as well as roles inside and outside the team can be found here.

The *theory* chapter focuses more on the field of study, background, and in-depth information about the task. Theory about this task's subject, as well as our reasoning for choosing it can also be found here.

Requirements specifications describes how the software is expected to perform, as well as what it will do with respect to functional and non-functional requirements. It also describes the functionality the product needs to fulfil all the stakeholders needs. Elements such as use cases will be used to visualize different real scenarios.

The *design and technologies* chapter mainly consist of design choices from start to finish, throughout the entire project period. From the first wireframes and sketches, to finished application. All our decisions along the way will be explained and justified here.

The development process describes our choice of development methodology, which in our case is a combination of scrum and Kanban. These development frameworks are quite agile and suite the team's way of working very well.

The *implementation and production* chapter describes our progress throughout the project period. From obstacles and problems to smart solutions and general progress is described here. Justification for coding languages, frameworks, methods, external APIs, and tools can also be found here.

The *code review and code quality* chapter hold information about how the code is structured, as well as measure we took to increase performance and shorten the code by modularising it.

Introduction

The chapter about *testing and quality assurance* will describe the measures we took to test different functions and functionalities within the application, as well as assuring the quality and security of the code through static code analysis.

Installation and realization work like an instruction for the user to get going with the application, and states how to deploy the application. How to install dependencies and which commands to run. See the README.md file in the project repository for more information regarding this.

The *ending chapters* contain information about discussions regarding results and possible alternatives and different solutions. Critiques based on the project start and the project end, as well as evaluation of the entire team's work and distribution of tasks is stated. The report will be finished with a conclusion where we will discuss the report overall.

Appendices for this report includes:

- Spreadsheet of usability testing.
- Original Gantt scheme.
- Updated Gantt scheme.
- Project plan which gives the reader an idea of the team's thoughts and plans prior to the project period.
- Signed collaboration agreement with our supervisor and employer.
- Time log.
- Meeting log.
- Wireframes.

This report uses the font-style Times New Roman for text and Arial for headlines. A font size of twelve and a line spacing of 1.15 is used for the document text. The team saw this fit for a report like this, since the report is formal and consists of academic level writing. The line spacing of 1.15 provides better readability as the lines are more separated, but not too separated.

2 Theory

2.1 Introduction

In this section of the report, we are to further elaborate the purpose of this project. This chapter discusses the different fields of expertise we have been diving into during this project, as well as introducing necessary explanations required to fully understanding concepts and definitions used throughout the report.

2.2 Background and Purpose of the Project

Why Threat Total as a project?

The NTNU Security Operation Centre (SOC) has a need for a self-help portal for handling security related incidents in the form of this task. The security related incidents involve a proactive action, where a user can upload a suspected malicious item or link to the portal. The portal will then check whether the item is malicious or not and return an accessible answer in simple terms. This is where the application comes in, Threat Total.

The purpose of this project is to help NTNU students and employees to retrieve threat intelligence on items through a self-help portal, as this can help protect them from potential digital threats, and reduce the workload for the NTNU SOC. The biggest value in the Threat Total application is the interactions of the end user. The user can through our application, receive guidance of what to do with a potential malicious item, without needing to have a broad understanding of cyber security. This is done by the high-level information design of Threat Total, where the interaction design is easy to understand and displayed information is assessable for a wide target group. In the end, the intention here is to reduce the potential overhead towards the NTNU SOC, as the application may help reduce the amount of incidents that the SOC must handle.

The major parts of this project were to:

- Create a user-friendly frontend.
- Create a stable and secure backend.
 - Fetch data from different open sources.
 - Prepare functionality for fetching data form internal NTNU sources.
- Between the making of the frontend and backend, we had to figure out a way to represent the data in a high-level language.
- Integrate Feide authentication.
- Cache data to limit unnecessary third-party requests and improve request times.

2.3 Research Methodology

As the team did not have one fixed, defined way of conducting the research by itself, the general flow of research consisted of three methods. These consisted respectively of *trial-and-error*, *self-study* and *learning by doing*.

The method of trial-and-error consists of trying to solve a problem based on several attempts with different solutions. By doing this, we can observe the outcome of every solution and exclude those that fail and move on with the next one until one proposed solution is successful.

The principle of self-study is a qualitative research method and focuses on the learning outcome of each individual. Self-study is done among other things by reading texts, relevant documents and watching videos, all while taking notes. The aim of this methodology is to improve the knowledge in the specific field that is being researched.

Learning by doing is a practical research method that resolves around learning from experiences directly from the performed actions. As with the trial-and-error method, we are able to observe the outcome of a tried solution on move on to the next if a failure occur.

2.4 Full Stack Web Development

This project resolves largely around the process of full stack web development, which refers to the development of both the frontend (client side) and the backend (server side) of an application. [24] Developing a full stack application requires broad knowledge of information systems, as transferring and handling information is a recurring theme in full stack development. The following subchapters will focus on individual things which all relates to the principle of full stack development.

2.4.1 Frontend responsive web design

Responsive web design is all about making the design of the application look good on devices of all sizes, from phones to TVs. The application should look good independent from the size of the screen, which platform the application it is being run on and the orientation of the screen. The styling language, CSS, provides such scaling functionalities with *CSS media queries*. [7] These queries allow basing things such as the placement of objects, and the general height and width of the web application on fixed display-size breakpoints. The example below shows how the media queries in CSS can be implemented:

```
@media screen and (max-width: 1000px) {  
  body {  
    background-color: red;  
  }  
}
```

FIGURE 1: CSS MEDIA QUERY EXAMPLE

The code snippet above will set the colour of the screen to red for every pixel width less than 1000 pixels.

Responsive web design come with several different approaches of implementation. Amongst these approaches, there are mobile first design and desktop first design. [21] Mobile first design focuses on the assumption that the application will be used mainly on a phone. The design is hence firstly adapted the screen size and ratio of a mobile phone, and then gradually adapted to bigger screens with different ratios. The other approach that is quite popular to follow starts in the other end of mobile first design. In this approach the designers and developers will firstly design the application for usage on a pc display, and then gradually move on to smaller screens.

2.4.2 Interaction design

Interaction design, also referred as “IXD”, is the principal of designing an application that always has the end-user in mind [17]. It focuses on understanding the user’s needs and limitations, and thereby produce an output which precisely suites the user’s demands. Interaction design has five different dimensions to look at during the designing process to help with understanding more of the user’s needs and wishes. The first dimension focuses on adapting the amount of text shown at different objects, such as buttons and interactive objects in a web application. The second dimension focuses on the representation of the product to the user. Elements such as icons, images, colours, images, text sizes and text fonts should be considered in this dimension. The designers should focus on implementing these things to guide and aid the user in the right direction, and not cause any confusion or uncertainty. The third dimension refers to what medium the user interacts with to interact with the product. For example, the trackpad for a laptop, or the finger for a phone. The fourth dimension focuses on elements such as sounds and animations that changes over time. The fifth and last dimension describes how the product reacts on user input and responds. For example, how a website responds after a customer has purchased an item. The information provided to the customer should be detailed and not leave the customer confused as this would lead to weaker usability.

The principle of interaction design is an important part of the non-functional requirements for the application that the team have developed. More information about the non-functional requirements can be found in chapter [3.2](#).

2.4.3 Networking operations

Developing the backend for an application involves handling the serving of the pages and API calls from the frontend. This involves handling networking operations, such as HTTP requests and status codes.

General HTTP status code categories:

- 1xx – Informational response, indicates that the request has been received and understood, and is further handling the response by communicating protocol-level information.
- 2xx – Success, indicates that the request has been received, understood, and accepted.
- 3xx – Redirection, indicates that further actions must be taken in order to complete the request.
- 4xx – Client error, indicates an error in the request caused by the client.
- 5xx – Server error, indicates an error in the request caused by the server.

HTTP methods:

Method	Description
GET	Retrieves information from the API.
POST	Sends data and updates the API.
PUT	Adds data to the API.
DELETE	Deletes data from the API.

TABLE 2: COMMON HTTP REQUEST METHODS

These are the necessary HTTP methods which create the back end of our application, allowing communication between the front- and backend of the program in addition to between the backend of our program and our third-party information sources.

2.4.4 Testing

Testing is one of the most important parts in software development. For the sake of theory, we will briefly introduce two main categories of testing, as we will further elaborate this topic in later chapters. The main testing methods for creating a software are user- and technical testing methods.

User testing plays a huge role in the expenses of creating a product, both in the currency of capital and time. The earlier the product is tested and validated, the earlier the development team can catch up and figure what functionality needs to be changed, what can be kept and what should be tossed. Wireframes are a great example of how to perform user testing without having to implement code. Wireframes act as a visual representation of an application and can be as simple as a drawing of the overall look and feel of the product. Further with the use of more advanced wireframe software, it is possible to achieve a more accurate representation of the product, without implementing a single line of code.

“It is estimated to be 100 times cheaper to change a product before any code has been written, than it is to wait until the implementation is complete.” (Jakob Nielsen) [19]

Technical testing is the act of performing various input to code which has implemented features and functions. The main goal of technical testing is to uncover potential breaking changes to the application, which is usually an automated process by self-written automation and / or the IDE compiler itself. There are other types of technical testing, which cannot be automated, such as destructive and exploratory testing. These types of tests are processes that requires creativity and manual input. This can be expensive to perform, however, returns highly valuable information about the system.

2.4.5 Authentication and authorization

For the context of this report, it is necessary to define the term authentication and authorization, as these terms are vital to understand the login functionality of the application.

Authentication involves the operations necessary of proving that the user that wants to log in, is in fact the user who wants to log in. [5] In the context of this application, this process is done through Feide. We have implemented this process using the OpenID Connect protocol, a product of Microsoft. OpenID Connect is an identity layer on top of OAuth 2.0, which works in a REST type of manner and verifies the identity of the user, based on retrieved information from the authorization server [29].

Authorization involves the operations necessary for granting permissions to an authenticated user. This process specifies what data the user is allowed to view and what other actions are allowed. In the context of our application, OAuth 2.0 is responsible for handling authorization of the user who is logged in through Feide. Further elaboration of the technology itself will be presented at the technology section of the report, though for the sake of theory we will briefly present the four parties involved in the OAuth 2.0 authorization process, which is paraphrased from the Microsoft documentation [18]:

- **Authorization server**

Also known as an Identity Provider (IdP), it handles user information, user access and trust relationships between the authorization process. In this instance, the IdP is Feide, and has the security tokens that the application uses to grant, deny, or revoke a user's access to different resources.

- **Client**

In an OAuth exchange, a client is defined as an application or web service, that requires access to a protected source. In this instance, client is the Threat Total application.

- **Resource owner**

In this instance, the resource owner is defined as the end user of the application. This is because the end-user "owns" a protected resource, their user data, in which the application assesses. The resource owner has the capability of denying or accepting the access to these resources. In this instance, Feide must retrieve user information to log in, in order to use the Threat Total application.

- **Resource server**

The resource server is the host that either hosts or provides access to a resource owner's data.

2.4.6 Security in software development

When developing software, it is important to have security in mind. When security is kept in mind throughout the development process, the result will be a more secure and robust application. As the application deals with third-party APIs, with API keys used for authentication, keeping these keys secure and hidden is vital. There are a lot of ways to secure API keys, firstly by not hard coding them into variables, as they can be found in the git repository if they are included in the code. Another method to prevent the API keys from showing up in the git repository is to put the different keys into files in a folder which is "gitignored" (not included in the git version control system). By including a file called, ".gitignore" inside the git repository, with the names of the files to be ignored this is possible. A third option we used at a later stage in development is to use environment variables, which are local variables on the computer that can be retrieved by the application. A tool which can help prevent leaking of API keys and other passwords is SonarQube which is a static analysis tool described later in the document. Another thing which was implemented to secure the data of the application, is authorization of the user before allowing the web application to retrieve information from the backend. The backend then required authentication to retrieve information, this protects the data sources and protects the application from unwanted traffic.

2.5 The Scrum Model

Scrum is an agile process framework used for managing complex projects. The framework itself does not directly aid in developing code, but instead helps the team to collaborate efficiently and make progress [30]. The main components of the scrum framework are:

- **A team**, which consists of two or more team members who build the project. These are stakeholders of engineering the product and ensuring quality of the product.
- **Stand-up meetings**, which are short meetings that are held at the start of each work session, typically around 10 -15 minutes. To make the meeting as brief as possible, the following three questions are asked and answered in the meeting:
 - What did the team do last session?
 - What is the team going to do now?
 - Are there any obstacles that stand in the way of completing the upcoming tasks?
- **Sprints** are the main activity of the scrum model. A sprint consists of a work cycle with a specific period, in which at the end of the period, a goal is to be reached. A typical sprint lasts around two to four weeks.
- **A Scrum board**. Before every sprint, the sprint is planned with a scrum board, so that all members know what their tasks and roles are, both before and during the sprint. The goal of the sprint planning is to identify and define tasks, and further divide them into smaller tasks to place on the scrum board. The scrum board is used to visualize the distributed tasks and gives a brief overview of the current status of the sprint.

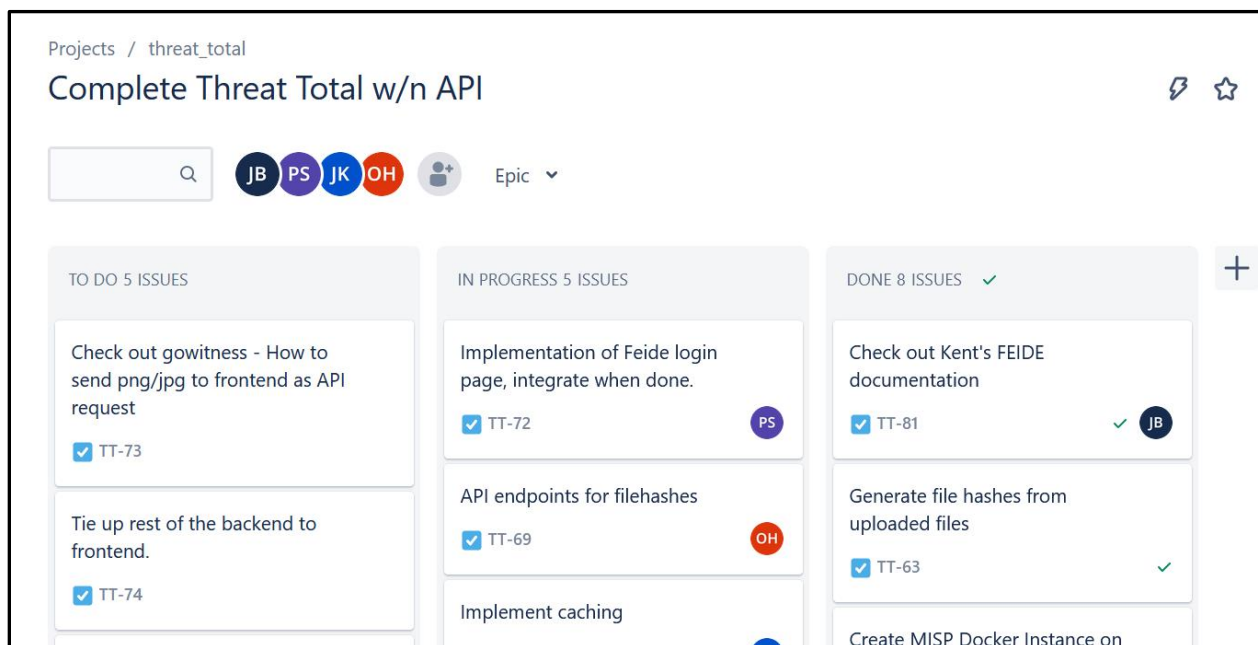


FIGURE 2: AN EXAMPLE OF A SCRUM BOARD, SCREENSHOT FROM JIRA ATlassian TOOL

Theory

- **Sprint review.** After completing a sprint, the team are to sum up the results of the sprint and have a retrospective. This way, the team can reflect on elements that went well, and areas that need improvement. This can improve not only the quality of a product, but also how the team works when producing value. The outcomes of a retrospective are actions for the next sprint, resulting in a continuous improvement throughout the upcoming sprints for a project.

These are the main components of the scrum framework that aided in structuring and developing the project.

3 Requirements Specification

3.1 Introduction

This chapter consists of different requirements based on conversations with our employer, as well as requirements which were stated in the task description. These requirements will be demonstrated using use cases and will be described by dividing them into functional and non-functional requirements.

3.2 Functional Requirements

Functional requirements are included in any software development project. These requirements define what the finished project must do. [23] If it does not meet these requirements, the application will fail. One way to test these requirements is with inputs and outputs. The requirements should state a desired output based on a specific input. An example for this application is that it should be able to authorize a user with the help of Feide. The expected output would then be a fully authorized user which was able to log in to the web application.

You can find a bullet list below of the several functional requirements that were given to the team, both via conversations and meetings with our employer, and from the project description:

- Functionality to search for a domain, URL, or file hash.
- Login functionality using Feide.
- Retrieval of contact information from Feide.
- The possibility to create an event for analysis to NTNU SOC.
- Utilize a REST API to retrieve information from NTNU SOC.
- The application should be able to show disposition for indicators (Domain, IP, et cetera)
- Gather reputation data from both public and private (NTNU's) reputation sources.
- Submit file, URL, domain.
- Block / allow list functionality for indicators.
- Filter for what can and cannot be displayed to the user.
- Submitted files should first be scanned automatically, and if no result is given the file may be submitted for manual analysis. If the file previously has been analyzed, the previous analysis should be submitted to the frontend user.
- [OPTIONAL] The application should be able to collect and display a screenshot of the domain or URL that is sent in for analysis.

3.3 Non-Functional Requirements

Non-functional requirements describe how the application goes about delivering a specific function. [23] The difference between functional requirements and non-functional requirements is that functional requirements focus on *what the system must do*, while non-functional requirements focus on *how the system works*. Non-functional requirements do not by definition have an impact on how the application will function, but more on how the application will perform. [13] In short, system usability. What kind of nonfunctional requirements must be met for the user to not be frustrated or get stuck whilst using our web application?

Non-functional requirements:

- Users should be easily able to navigate the web application.
- All essential information should be one click away.
- The web application should have a pleasing UI.
- The web application should use NTNU branding, including theme and color.
- The UI should be as simple as possible which will lead to a reduction of possible human errors.
- The process for ending with a report should not be too advanced, and the steps should be clearly defined and memorable.
- UI should follow Don Norman's six principles for designing and experiencing good UX [1]:
 - Visibility – users should be able to know all their options straight away.
 - Feedback – there should be always a reaction to a user interaction (button changing color, loading animation, et cetera).
 - Affordance – the relationship between how it is used and what it looks like.
 - Mapping – the relationship between control and effect. For example, the scroll bar on a web page which resembles your location on the web page.
 - Constraints – the limits to an interaction or an interface.
 - Consistency – the same user action must produce the same reaction from the application, every time. If this rule is broken, the user can get frustrated, and the usability of the application will go drastically down.

3.4 Use Cases

Use cases visualize different scenarios that can take place when using the web application. They represent how the application should work from the user’s perspective. Each use case is beginning with a goal, and ending when the goal is fulfilled:

3.4.1 Use case 1 – Log in

Use case 1	NTNU student or employee wants to log in
Actor	NTNU student or employee (will be referred to as a user)
Use case overview	NTNU user wants to log in to “NTNU Threat Total” via FEIDE.
Trigger	NTNU user enters their NTNU username and password and submit them for verification in FEIDE.
Precondition 1	User has a valid NTNU email and password.

TABLE 3: USE CASE 1 - LOG IN

Basic flow: Use case 1

Description	This scenario describes the situation where the NTNU student / employee is successfully logged in to the web application.
1	User navigates to “ https://www.threat-total.ntnu.no ” (example URL).
3	User clicks “Log in” button.
2	User is met with log in page for FEIDE credentials.
3	User enters the right credentials.
4	Credentials get authorized with FEIDE, and an access token gets generated, hashed, and stored as a cookie in the browser.
5	User is redirected to Threat Total’s home page.
Termination outcome	User gets redirected to the home page of Threat Total.

TABLE 4: USE CASE 1 - BASIC FLOW

Alternative flow 1A: Different route for logging in

Description	This scenario describes a different path for the user to log in and be able to use the search functions.
1A1	User enters a URL or provides a file and hits “Investigate”.
1A2	Since the user is not authorized with FEIDE, the API call is not done, and the user gets redirected to a different page which tells the user to log in and has a visible log in button.
1A3	The user clicks the log in button, enters valid credentials and gets authorized.
Termination outcome	User is authorized and logged in.

TABLE 5: USE CASE 1 - ALTERNATIVE FLOW

3.4.2 Use case 2 – Investigate URL

Use case 2	NTNU user tests a malicious URL
Actor	NTNU user
Use case overview	A valid NTNU user have received a suspicious URL via email and wants to check whether it is malicious or not.
Trigger	User puts the URL in to the search field on the home page of Threat Total and clicks the “Investigate” button.
Precondition 1	NTNU user is logged in to Threat Total.

TABLE 6: USE CASE 2 - INVESTIGATE URL

Basic flow: Use case 2

Description	This scenario describes the situation where a logged in NTNU user enters the URL of a potentially malicious website, investigates it, and receives a result.
1	NTNU user copies the URL from the email and pastes it in the search field of Threat Total’s home page.
2	NTNU user clicks the button called, “Investigate”.
3	URL will be sent to several anti-virus agents and returns a response based on if it is malicious or not.
4	NTNU user will be redirected to the result page with a form of different anti-virus agents as well as what the reported about the URL.
Termination outcome	NTNU user has successfully obtained information about the maliciousness of the URL.

TABLE 7: USE CASE 2 - BASIC FLOW

3.4.3 Use case 3 – Investigate file hash

Use case 3	NTNU user tests a file hash
Actor	NTNU user
Use case overview	A valid NTNU user wants to analyse a file hash and wants to check whether it is known to be malicious or not.
Trigger	User puts the file hash in the search field on the home page of Threat Total and clicks the “Investigate” button.
Precondition 1	NTNU user is logged in to Threat Total.

TABLE 8: USE CASE 3 - INVESTIGATE FILE HASH

Basic flow: Use case 3

Description	This scenario describes the situation where a logged in NTNU user enters the file hash of a potentially malicious file, investigates it, and receives a result.
1	NTNU user copies the file hash and pastes it in the search field of Threat Total’s home page.
2	NTNU user clicks the button called, “Investigate”.
3	The file hash will be sent to several anti-virus agents and returns a response based on if it is malicious or not.
4	NTNU user will be redirected to the result page with a form of different anti-virus agents as well as what the reported about the file hash such as known filename.
Termination outcome	NTNU user has successfully obtained information about the maliciousness of the file hash.

TABLE 9: USE CASE 3 - BASIC FLOW

3.4.4 Use case 4 – Investigate file

Use case 4	NTNU user tests a malicious file
Actor	NTNU user
Use case overview	NTNU user has received a potentially malicious file and wants to check whether it is malicious or not.
Trigger	NTNU user uploads a potentially malicious files and clicks the “Investigate” button.
Precondition 1	The NTNU user is logged in to Threat Total.

TABLE 10: USE CASE 4 - INVESTIGATE FILE

Basic flow: Use case 4

Description	This scenario describes the situation where a logged in NTNU user uploads a potentially malicious file to the Threat Total web site and receives feedback from it.
1	NTNU user navigates to “Upload file” section.
2	NTNU user clicks the button, “Upload file”.
3	“File Explorer” is opened.
4	NTNU user navigates to the file, clicks it and adds it.
5	NTNU user clicks the “Investigate” button on the web page.
6	File is sent to Golang backend, which uses the file content to prepare and send a request to the Virus Total API.
7	Virus Total API returns a file report ID
8	User is redirected to the result page, which then GET's the file report contents via the returned ID.
9	The response from Virus Total displays the information provided in the file scan report.
Termination outcome	NTNU user receives information if the file is malicious or not.

TABLE 11: USE CASE 4 - BASIC FLOW

3.4.5 Use case 5 – Log out

Use case 5	NTNU user want to log out.
Actor	NTNU user
Use case overview	NTNU user wants to log out from the current session.
Trigger	Logged in user clicks “Log out” button.
Precondition 1	User is logged in.

TABLE 12: USE CASE 5 - LOG OUT

Basic flow: Use case 5

Description	A logged in NTNU user is finished with scanning the supposed malicious URL / file and wants to log out from the web application.
1	Logged in NTNU user is finished with investigating URL or file and is redirected to result page.
2	User clicks log out button.
3	Cookie about user gets deleted, as well as the Feide session.
4	User gets redirected to logout page which provides the user with information about the log out.
Termination outcome	NTNU user is successfully logged out and the session is successfully deleted.

TABLE 13: USE CASE 5 - BASIC FLOW

4 Design and Technologies

4.1 Introduction

This chapter contains information about prerequisites for the development of the application, decisions during the project period, wireframes that was created for the application and how the user interface of the finished product looks like.

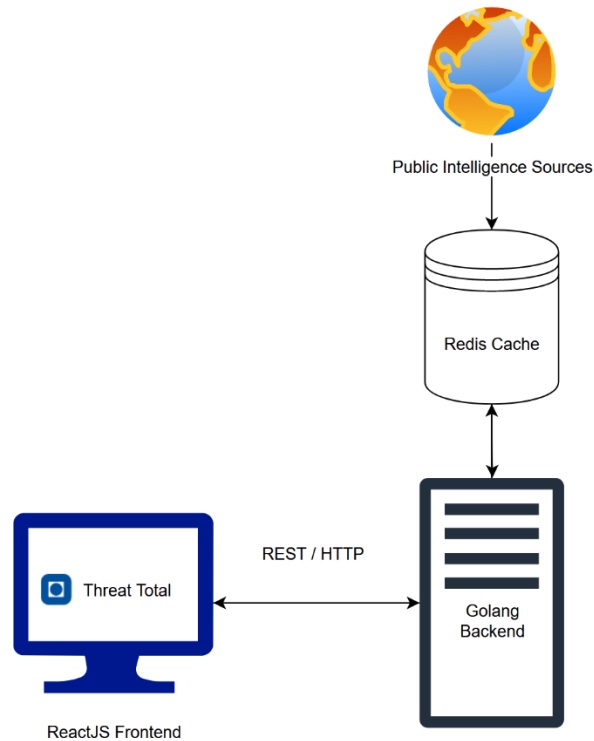


FIGURE 3: GENERAL OVERVIEW OF THE FULL STACK APPLICATION, MADE IN DRAW.IO

4.2 Prerequisites

Required fundamental understanding of the following scopes regarding design and used technologies [11]:

- Backend development in Golang.
- Frontend development in JavaScript, HTML and CSS.
- Collaboration using hosted Git instances such as GitLab or GitHub.
- Deploying the web application with a given framework.
- Fundamental understanding of the principle of caching.
- Be familiar with virtualization and containerization
- Be familiar with networking principles

4.3 Design Decisions

During the project, the team faced some challenges when it came to frontend libraries, and where and how to host the backend. A solution for storing data in the form of caching also had to be implemented both according to the task specification and the effectiveness of the application itself. The following subchapters provides explanation and reasoning as justification for our design decisions.

4.3.1 Frontend

Once the team had decided what programming language to use for the frontend, which ended up being JavaScript. We had to decide which library to use, we first decided to try implementing LitElement. LitElement is in essence a JavaScript library for web development and works by building different components and referring to them several places in the code. We ended up spending too much time trying to implement LitElement without succeeding, so in the end we decided to change tracks and find an alternative solution to implementing components. The choice ultimately landed on ReactJS, which works quite similarly by building different web-components and reusing them several places in the code by including these components on different pages or in different components. The examples below show a snippet of the application's navbar and then how it is included in the "homepage.js" file:

```

return (
  <>
  <nav className="container h-auto pl-2 pr-2 sm:pl-18 sm:pr-18 md:pl-36 md:pr-36">
    <div className="flex h-12 sm:h-14 p-1">
      <div className="h-full ml-3 w-full flex items-center">
        <a href=".">
          <img src={logo} className="sm:h-6 h-4 w-auto alt="NTNU Logo" />
        </a>
      </div>
      <div className="h-full ml-3 w-full flex items-center">
        <a href="/about" className="p-2 text-gray-500 font-semibold m-auto hover:underline">{t("about")}</a>
      </div>
      <div className="float float-right w-full h-12 sm:h-full sm:pr-3">
        <div className="float-right place-items-center h-12 mt-2 sm:mt-3">
          &#127760; <button onClick={() => changeLanguage()} className="hover:underline" title={t("languageDescr")}>
        </div>
      </div>
    </div>
  </nav>
  <div className="border-b-2 border-gray-200 h-1 w-full"> </div>
  </>
);

export default Navbar;

```

FIGURE 4: EXAMPLE SNIPPET OF NAVBAR COMPONENT

```
    })  
  
    return (  
      <div className="grid place-items-center">  
        <Navbar />  
        <SubNavbar page="home" />  
  
        <div className='flex justify-center mt-6 sm:mt-8'>  
          <img src={ntnuLogo} className="h-20 sm:h-35 md:h-40 w-auto" alt="NTNU Logo" />  
          <h1 className="text-4xl sm:text-6xl md:text-8xl font-bold sm:ml-4 ml-2 pt-2 sm:pt-4 w-auto"> Threat Total </h1>  
        </div>  
      </div>  
    )  
  )  
}
```

FIGURE 5: EXAMPLE SNIPPET OF HOW THE NAVBAR COMPONENT IS INCLUDED IN HOMEPAGE.JS

For styling the webpages the team ended up using Tailwind CSS. This allowed for rapid development of simple designs, as a lot of the CSS basics are prewritten and can be used. For example, scaling of elements to different screen sizes can be done using predefined breakpoints in Tailwind CSS using the “container” class.

4.3.2 Backend

For the backend Golang was decided upon from the beginning, as the language is well suited for developing REST API’s, is simple to write in and has thorough error handling built in. Routing was implemented using the gin-gonic library which is a web framework with functions such as middleware support, panic catching and routing. The first plan for the project was to handle both the front- and the backend using Golang, but the team early chose to split them up into two different programs with different languages. The decision was made because we found out that Golang did not provide anything extra as a simple webserver, and we therefore had no good reason to limit what webserver to use.

4.3.3 Caching

Caching is a principle within IT that focuses on storing a subset of data. [6] A request for the subset of data is being sent, and the response will both be cached / stored and sent / displayed to the user. Then the next time the user requests the same subset of data, the response time will reduce drastically as the data is already available in the cache and the data inside the cache will be shown. When it comes to caching inside a physical computer, caching of data happens in two central places. On the CPU (Central Processing Unit) and the RAM (Random Access Memory). A modern CPU operates with caching at three different levels, L1, L2 and L3 on focuses on operations specific to the operating system (OS) of the computer. The main difference between these three levels of cache is the speed of data transfer and the size of the caches where L1 is the fastest, but smallest and L3 the slowest, but largest cache. Caching in the RAM focuses on delivering data to running processes and programs during their execution and run time. For our task, we decided to use caching for requests for URLs, file upload id’s, file hashes and user

authentication within our application. The storage solution we chose to incorporate is called, Redis. This solution is an in-memory data store that is widely used in the developer community as a streaming-engine, cache, database, and message broker. [20]

Redis was implemented into the application by using the “redigo/redis” library. For the sake of testing our application and caching data, we hosted the Redis server locally, but the hosting can be done other places as well. Later followed the configuration of a password for access to the server. A pool connection was made to store and access the Redis data. The code snippet below shows how the pool connection was implemented with Golang using the “redigo/redis” library:

```
// InitPool initializes the storage pool used in the application
func InitPool() *redis.Pool {
    return &redis.Pool{
        MaxIdle: 80,
        MaxActive: 12000,
        Dial: func() (redis.Conn, error) {
            c, err := redis.Dial("tcp",
                os.Getenv("redisUrl"),
                redis.DialPassword(os.Getenv("redisPassword")))
            if err != nil {
                panic(err.Error())
            }
            return c, err
        },
    }
}
```

FIGURE 6: EXAMPLE OF HOW CONNECTION TO THE REDIS POOL WAS IMPLEMENTED

4.4 From Idea to Sketch

It all began with the team reading the project description and deciding that this task looked quite interesting and relevant in accordance with our field of study, as well as our interests. We then were assigned with the task. After being assigned with the task the team proceeded to have a meeting with the employer. In order to get more information about both the functional requirements, as well as the nonfunctional requirements of the project. This information led to the making of the first wireframes and worked as a starting point for the development of the application, and the final product. The wireframes contained elements such as how the navigation between different pages should be, and the general layout for each web page. Including elements such as branding, forms, and buttons.

Some of the key points for the nonfunctional requirements were to keep the user interface as simple as possible, and thus more accessible to people without advanced technical knowledge.

With that, as well as the task description in mind, the team collaborated on a set of wireframe sheets which worked as a fundamental basis for the design and development of the application.

4.5 Wireframes and Prototyping

A requirement we had was to make an accessible user interface with NTNU branding. So, to quickly prototype the frontend, and to get early feedback on our thoughts we made wireframes in balsamiQ. The wireframe was interactive with different pages linked together, this allowed us to test the basic use-flow of the pages before creating them. Below you can see the first iteration of the index / homepage:

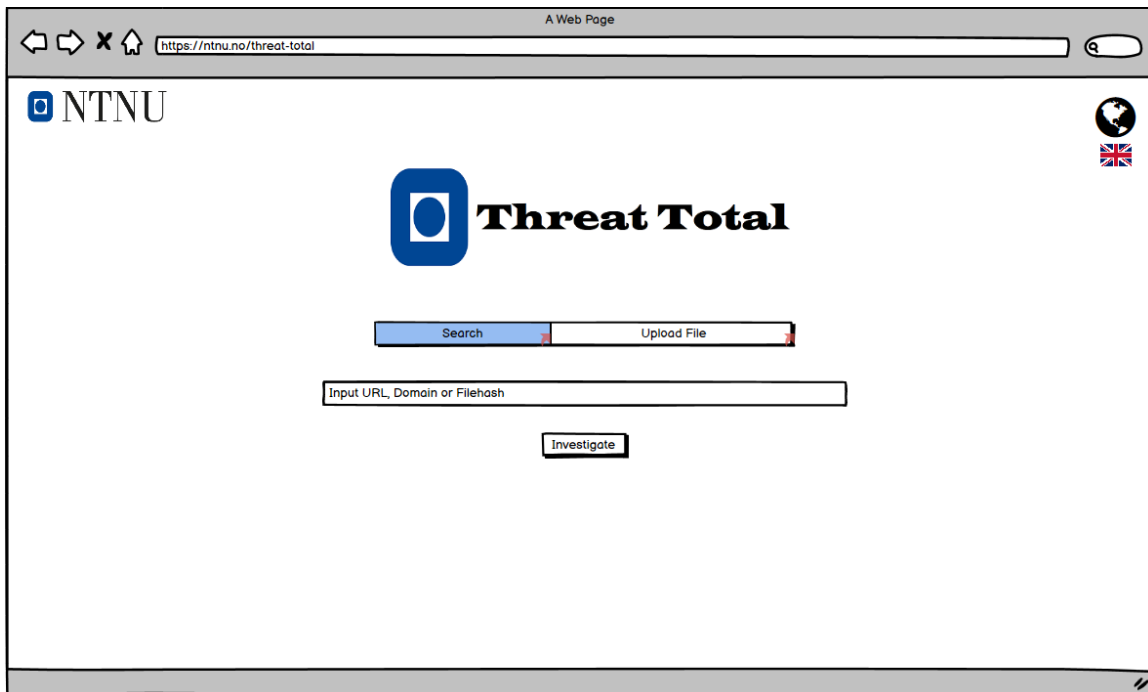


FIGURE 7: FIRST WIREFRAME ITERATION OF THE INDEX / HOMEPAGE IN BALSAMIQ

4.6 Wireframe Validation

After we had created a functional wireframe, we had a meeting with the NTNU SOC where we asked for feedback and approval of the plans. We got the feedback and approval from Frank Wikstrøm, who was mainly responsible for the non-functional design requirements of the application. He wanted the design to be as simple and accessible as possible. After showing him the wireframes, they were approved, and we could move on with implementing the design with HTML, JavaScript, and CSS.

4.7 User Interface, the End Product

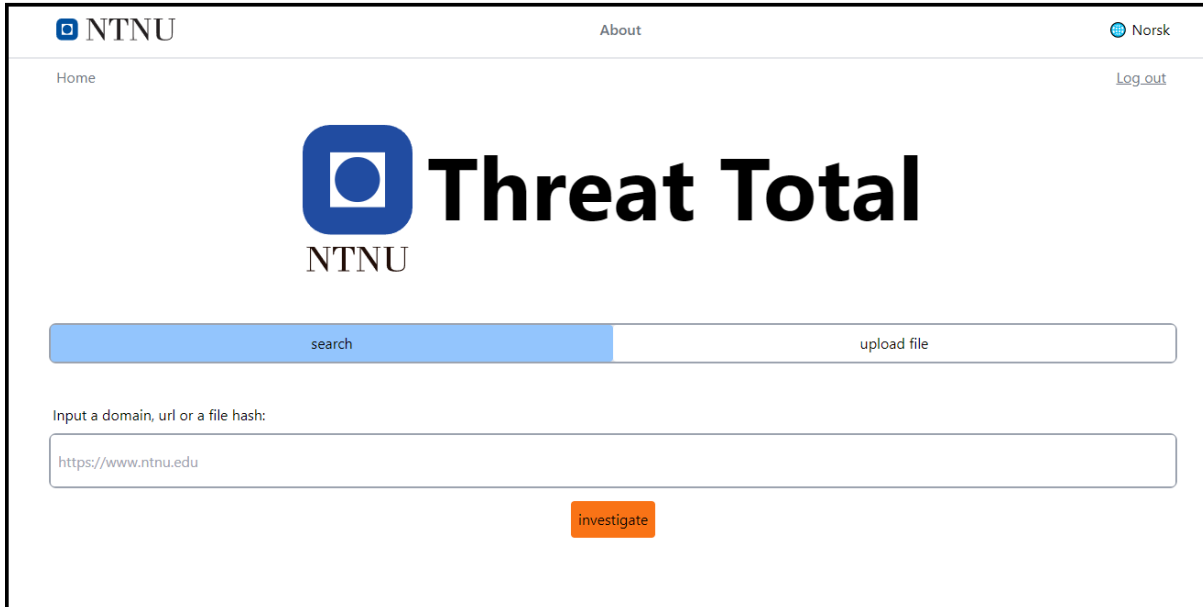


FIGURE 8: THE END PRODUCT

The final user interface of the application's homepage is similar to the original wireframe as shown above. As the design is simple and does not hold to many functionalities, we decided not to change the design too much before writing the code. The only things that were changed for the actual application is the option with clicking the button, "About" for information about the application, a "Log in/ log out" button was added to the top right, and a dynamic page view was added to the top left of the pages. This view changes based on the page the user is visiting. For example, the user is visiting the upload page, and the page view shows "Home > Upload".

5 Development Process

5.1 Introduction

This chapter contains information about the development process of the web application. What kind of software development framework the team chose to use, alternatives to it, tools used, and a summary of each scrum sprint we went through.

5.2 Scrum

Scrum is an agile framework for helping a team collaborate. [30] As described earlier in the theory part of the report, Scrum consists of several Sprints. In these sprints, there are defined tasks which are to be completed within the duration of the sprint. The duration is ideally around two to four weeks long. The actual duration of the sprints can be decided upon within the team.

In the scrum development framework, several sprint backlogs are included which keep track of tasks for a specific sprint, as well as for the entire project. A sprint backlog is in essence a “wish list” for prioritized tasks to be completed and creates the product backlog for the entire project when combined. The product backlog contains all the tasks for the entire project and provides the user a good overlook over the entire project process. This backlog can also work as a troubleshooting guide for looking at which person that oversaw a specific task at a specific time. The sprint backlog however contains all tasks for a specific sprint and lasts as mentioned ideally about two to four weeks.

As with all things, this framework comes with its advantages and disadvantages. When it comes to the advantages, scrum provides a clean and readable overview with the provided scrum board through Atlassian’s Jira software. By using sprints to define periods to work on a set of tasks, the chance of them being completed on time is greater than when not dividing and assigning people to the tasks. This goes hand in hand when it comes to larger projects. For this type of project, it is essential to have a good overview of the tasks and the management of them. Scrum being an agile framework allows for continuous feedback, both internally and from the stakeholder or employer. The received feedback can then help to direct the project to the correct path if it ever goes off course, as well as help to verify or disprove the completion of a milestone or a goal. After each sprint, the team should conduct a meeting to discuss the sprint in retrospective. This meeting should be quite short in duration and meeting focus on what went well during the sprint, what problems arose during the sprint and the solution to these problems. If the problems were solved in the end.

The scrum development framework also comes with its fair share of disadvantages. Perhaps the biggest flaw with scrum is the high possibility of scope creep. This means that the scope of the project gradually becomes bigger than originally planned without anyone on the team noticing as it can be a gradual process. As this framework is agile and dependent on the cooperation of all

team members, the project may fail if team members are not especially cooperative by nature. This is a consideration that should be made when putting the team together. Deciding how long a sprint should last can also be troublesome, especially if using the scrum framework is quite new to the team or the tasks are new to the team members. This can lead to sprint activities not being finished on time, and in the worst-case scenario this can lead to failing to complete the project.

5.3 An Alternative Development Framework to Scrum

Kanban is a popular framework like agile and scrum. Kanban is based heavily around DevOps principles and principles of continuous development. [28] In contrast to scrum, Kanban does not have any time frames, whereas scrum has sprints. These two are also quite different when you look at the approach to each framework's concepts. While scrum focuses heavily on fixed timelines for sprints, the concept of time is perceived different with Kanban. The delivery and flow should happen continuously. The division of roles are non-existing, in contrast to scrum, where roles play a big part. Changes can also be made at any time when using Kanban, unlike with scrum where changes and tasks are directly connected to sprints.

Some of the benefits of Kanban includes flexibility of planning, visual metrics, and termination of bottlenecks. Instead of assigning tasks to a specific person, members of the team are free to pick the task of their choice from the product backlog. This allows for a flexible way of working. A flow diagram can be implemented into the project to keep track of issues and spent in these. By visualizing this data in a diagram, the team might be able to spot the bottlenecks even faster. The aim with this diagram is to minimize the time spent on issues. Another key principle when working with Kanban is the amount of Work in Progress (WIP). Which can be a downside, as more work items are worked on in parallel, leading to more switching of context, and can then lead to tasks not being finished.

5.4 Our Development Framework

When deciding which development framework to use, we had to consider several different frameworks, look at pros and cons of the different frameworks, and eventually pick one. When deciding a framework there are some primary things to consider:

- The type of system which is being developed.
- How requirements and needs are defined, and whether they can be changed later.
- What the existing preferences are in the development team.
- The possibility of using a combination of different models.

For our team's case, we ended up with a mixture of scrum and Kanban, which are two agile frameworks for development. Combined, they can be called, "scrumban". During our development process we used the Kanban board from the Kanban framework and combined it

Development Process

with the principle of sprints in scrum for better time management. While allowing more time to finish a task than was originally stated in the sprint. Scrumban is hence a combined framework that suited us well, as we attended other courses next to the bachelor's thesis. With the good visualization by using sprints from scrum, the Kanban board from Kanban, and the openness when it comes to finishing tasks in Kanban. We managed to implement the framework into our way of collaborative work.

The fact that we already had knowledge about scrum from previous projects and courses also contributed to our decision to use scrum as part of the chosen framework. Sprint retrospect meetings were done weekly with the aid of our supervisor, Espen Torseth. The topics of these meeting included what we had done the previous week, what we planned to do the following week and if there were any obstacles in the way of accomplishing these tasks.

5.5 Development Environment and Tools Used

Task Description	Tool
Backend and frontend development	Visual Studio Code with the following addons: <ul style="list-style-type: none">• HTML and CSS Support• Simple React Snippets:• Tailwind CSS IntelliSense• Go
Creating wireframes	BalsamiQ
Report Writing	Microsoft Word
Diagram creation	Draw.io
Gantt scheme creation	Teamgantt.com
Hosting Redis	Windows Subsystem for Linux instance (WSL2)
Hosting SonarQube (static code analyser)	Docker
Sharing code amongst team members	<ul style="list-style-type: none">• Git• Gitlab

TABLE 14: PROGRAMS AND TOOLS USED

5.6 Summary of the Scrum Sprints

Sprint 1 Project Plan Delivery: January 25, 2022 - February 1, 2022

After defining what working framework to use and getting further information from our client, we started the first sprint. During this time period, the main goal was to create the project plan. The project plans main purpose were to define scope of the project, delegating responsibilities and accepting agreements. Having a good project plan was one of the requirements of starting the bachelor project and had to be approved by our supervisor before we could continue. This requirement laid a beneficial foundation for our main project work and report structure.

Sprint 2 Wireframes and main report: January 31, 2022 - February 7, 2022

The main goal of this sprint was to create Wireframes of our application and get feedback from our client. After getting the project plan approved, the team had to do some research to find a frontend- and backend framework that would fit our use case the most. This is the phase where the team started getting to know with the development frameworks we ended up with, in addition to establishing a structure of the main report of the project. The team also started dividing into separate areas of focus, since we were still in the research phase. Each member assigned themselves to tasks and took on different responsibilities related to the project.

Sprint 3 Threat Total Development: February 4, 2022 - February 28, 2022

The expected result of Sprint 3 was to have a user-testable prototype of our application. The user testable prototype involved making a basic web-application using the newly found frameworks of ReactJS + TailwindCSS. The main delegation of work here, were to create the different pages of the frontend, while also making some small progress on the backend. The main part here was to understand the usage of JSX with React and to create a viewable frontend.

Sprint 4 Threat Total Prototype Development: February 28, 2022 - March 13, 2022

The main goal of Sprint 4 was to create a complete frontend prototype of the website that included calls to the backend. The team spent most of their time researching the backend structure and developing it in Golang. The frontend was also redesigned, and the team got a greater understanding of how to make and use functions in React through research and testing.

As presented in the updated [Gantt scheme](#) this is the point where the actual time scheme started to stray away from what our [original plan](#) looked like.

Sprint 5 Complete Threat Total with no API: March 21, 2022 - April 8, 2022

This sprint ended up being an extension of Sprint 4, due to the complexity of the sprint and other time-consuming school tasks. This resulted in the team needing more time to develop this version of the application. The goal of this sprint remains the same, aiming towards making a more complete version of the application. Towards the end of March, we ended up with a version of the application that we were able to perform a live demo on. This led us to schedule a meeting with our client, to receive feedback on our design and technical decisions. On April 1st we had a meeting with the NTNU SOC, where we performed a live demo on the implemented functions of Threat Total. The functions implemented at the time were the search functions for hashes, domains, and URLs, and displaying results from various scanning engine sources in the frontend. After getting feedback and more information regarding API access, we went ahead and started planning our next sprint based on the information exchange of the meeting.

Sprint 6 Complete Threat Total with no API: April 4, 2022 – May 13, 2022

The end goal of this sprint was to finalize our full stack application as far as possible without access to the NTNU's internal API sources. Due to the war in Ukraine and a changed threat picture for the NTNU SOC, we were informed that we will not have access to the necessary sources for the internal functions' implementations. The reasoning behind this was a framework used for the SOC's API having a vulnerability which doesn't have a good fix yet. Therefore, we went ahead and fully implemented functions for Threat Total using public resources. In this period, we fully implemented caching, Feide login, improved search functionality and fully implemented file upload functionality. As well as implementing the final version of the "result" page, including a total verdict of the search, based on the results. Around the first week of May, the team gradually focused resources to finalizing the bachelor report, as we had to prioritize the report to finish in time.

6 Implementation and Production Process

6.1 Introduction and an Overview of the Application Structure

This chapter will focus on the structure of our full stack application and attempt to illustrate the main processes of the Threat Total application, namely URL and domain search, file hash search and file analysis. The figure below shows a detailed overview of our application's dataflow from login to a completed search or analysis. This figure will be elaborated on and explained in further detail in the chapters 6.2, 6.3 and 6.4.

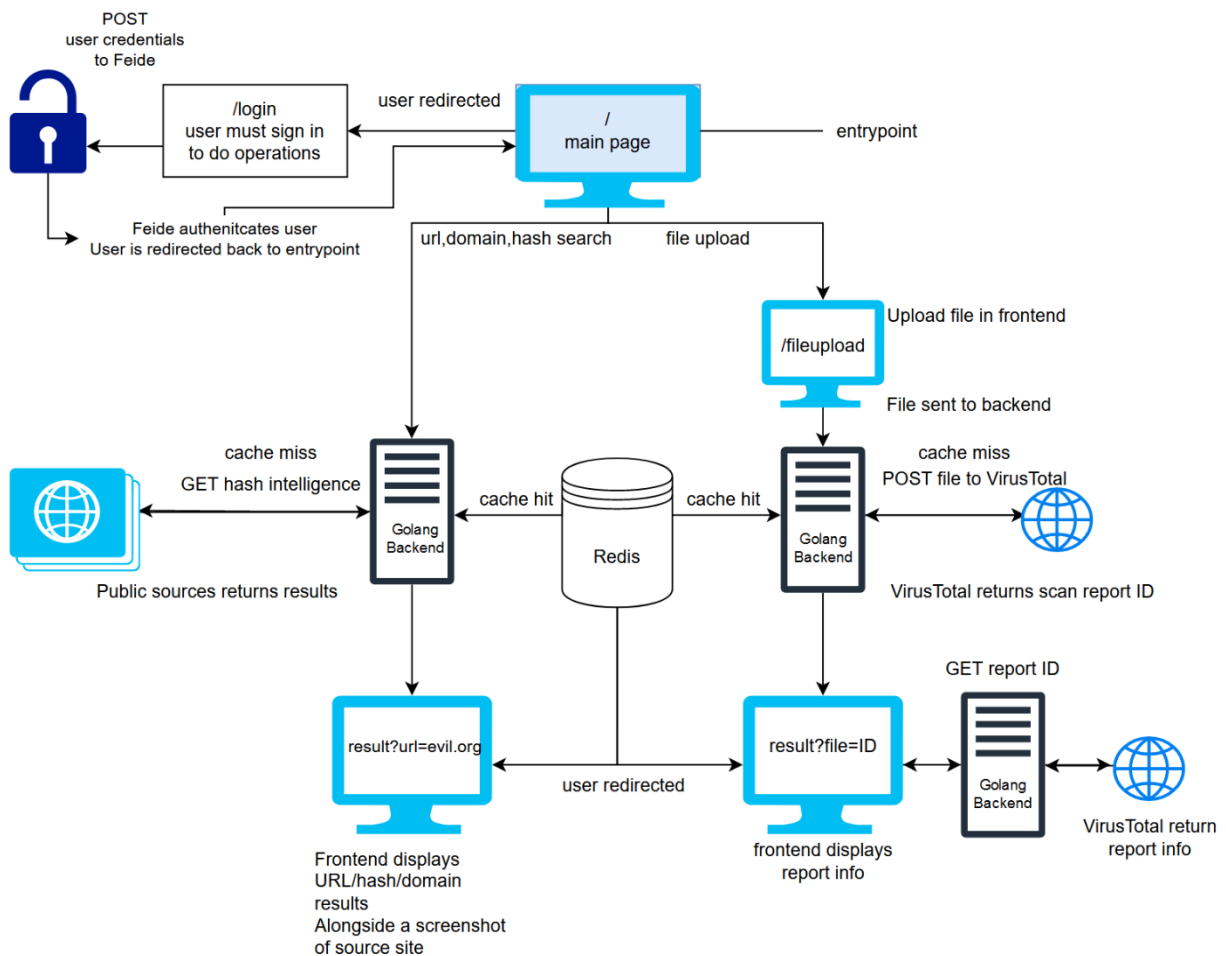


FIGURE 9: DETAILED OVERVIEW OF THE APPLICATION

6.2 Intelligence Sources

This application utilizes several public intelligence sources to collect information about searched, URL's, domains, file hashes and uploaded files. The intelligence we collect is collected through API endpoints. The endpoints we contact are shown in the table below:

Source name	Endpoint	What is collected?	REST Method
Google Safe Browsing API	https://safebrowsing.googleapis.com/v4/threatMatches:find?key=[API_KEY]	URL and Domain intelligence	POST
Hybrid Analysis	https://www.hybrid-analysis.com/api/v2/quick-scan/url	URL and Domain intelligence	POST
Hybrid Analysis	https://www.hybrid-analysis.com/api/v2/search/hash	File hash intelligence	POST
AlienVault	https://otx.alienvault.com//api/v1/indicators/url/[URL]/general	URL and Domain intelligence	GET
AlienVault	https://otx.alienvault.com//api/v1/indicators/file/[File hash]/general	File hash intelligence	GET
VirusTotal	https://www.virustotal.com/api/v3/files	Upload file to analysis	POST
VirusTotal	https://www.virustotal.com/api/v3/files/[ID]	Retrieve analysis data	GET

TABLE 15: API ENDPOINTS

6.2.1 Google safe browsing

Google Safebrowsing is an intelligence source that has been utilized to collect URL and domain intelligence. The threat data retrieved from the Google Safebrowsing API is divided into five main threat types: “malware”, “”, threat unspecified, “unwanted software”, and “potentially harmful application”. The API response received from the Safe Browsing API is based on matches to these threat types on all platforms. If a searched URL or domain has been marked as one of the five threat types, a response will be given from the Google Safe Browsing API containing information about which threat it has been associated with. Feedback to the frontend user will then be created based on which threat type was triggered. If the Google Safe Browsing has no entries for a given URL, or domain, the response object received will be empty. This makes it simple to filter the data received from the intelligence source, as when there is a response it can be assumed that there is an entry for the URL or domain, categorized under one

of the five threat types. The quota for lookups through the API is 10,000 requests per day, and 1,800 requests per hour. A bigger quota can be requested if needed.

6.2.2 Hybrid analysis

Hybrid Analysis is a threat intelligence source which is utilized to provide intelligence on file hashes, URLs, and domains. The Hybrid Analysis file hash intelligence is gathered from the Hybrid Analysis database of file samples and file hashes. The response from this API endpoint contains a verdict that states if the file is malicious, safe, or allow-listed or if Hybrid Analysis does not have any information of this file hash. This data along with the submitted filename if known, is utilized in the Threat Total application to generate a response to the frontend user about the file hash.

The URL intelligence gathered from Hybrid Analysis is information that Hybrid Analysis requests from external APIs, namely VirusTotal and Urlscan.io. The response data gathered from Hybrid Analysis contains a list of “scanners” namely VirusTotal and urlscan.io as well as a boolean stating if the analysis is completed or not. More on the individual scanner sources and information gathered can be read in the chapters 6.2.2.1 and 6.2.2.2

6.2.2.1 Hybrid Analysis – VirusTotal

Virus Total is an intelligence source that contains a large amount of data, and “VirusTotal inspects items with over 70 antivirus scanners and URL/domain block listing services, in addition to a myriad of tools to extract signals from the studied content.” [32]. This means that by utilizing VirusTotal as one of our intelligence sources, we with great certainty trust the analysis of an URL or domain. This is because of the large amount of data sources utilized by VirusTotal in their analysis. The downside of this utilization is that it takes time. On average the analysis of a new URL or domain that is not cached on their platform takes up to a minute. The team has however decided that the amount of information and the security the application is able to provide by including VirusTotal as one of our sources of intelligence is so valuable that it outweighs the cost of time.

In the response object that Hybrid Analysis provides on VirusTotal, there are a few fields that are utilized in our application. The “status” field which contains information about whether a domain/URL is malicious, safe, allow-listed or in awaiting analysis. The “total” field which contains how many sources the VirusTotal API has utilized in the scan of the submitted URL or Domain. And finally, the “positives” field which contains the number of intelligence sources that has triggered on malicious content on a URL or domain. All these fields are then handled by our backend functionality to deliver feedback on a URL or domain to the user as seen in figure 16.

The reason why we implemented the VirusTotal API through the Hybrid Analysis API, instead of directly contacting VirusTotal API, was because of the quota provided. The free quota for making requests to the VirusTotal API is four lookups per minute, with a daily quota of 500 lookups and a total monthly quota of 15 500 lookups. This quota would not be big enough to use

in a web application like Threat Total, which will be open to several thousands of students and staff. The free quota we receive from the use of Hybrid Analysis as a middleman, is limited to 200 requests per minute and 2000 requests per hour. In addition to, an unlimited number of total requests, outside of these restrictions are free. This quota along with the implementation of our caching solution, see [4.3.3](#), makes it possible to use Hybrid Analysis in the Threat Total application.

6.2.2.2 Hybrid Analysis – Urlscan.io

Urlscan.io is a service that scans and analyzes websites.

“When a URL is submitted to urlscan.io, an automated process will browse to the URL like a regular user and record the activity that this page navigation creates. This includes the domains and IPs contacted, the resources (JavaScript, CSS, et cetera) requested from those domains, as well as additional information about the page itself.” [\[37\]](#)

Urlscan.io has also been implemented as one of the intelligence sources we gain intelligence from and is along with VirusTotal, retrieved through the Hybrid Analysis API. Fetched data from the urlscan.io analysis response, is mainly the status field, which delivers information about whether a URL or domain is malicious, safe or has no classification. This data is processed in the backend to deliver feedback to the end user, as seen in figure [16](#).

The requests for urlscan.io analysis are coupled with the requests for analysis in VirusTotal through the Hybrid Analysis API. They operate on the same quota as mentioned above. Even though a request is made to both VirusTotal and Urlscan.io through Hybrid Analysis, this only counts as one request in the minutely and hourly quota.

6.2.3 AlienVault

AlienVault is a threat intelligence provider we utilize for URL, domain and file hash intelligence gathering. AlienVault is an open-source tool where users can register pulses which are a collection of traffic to URLs, domains, IPs, and files seen in relation to various network traffic mostly related to malicious activity. For example, a pulse registered on RAT (Remote Access Trojan) traffic may contain domains or IP addresses that have been utilized in C&C (Command and Control) traffic.

The utilization of this intelligence source varies slightly based on if it is searching for intelligence on file hashes, or URL/domain intelligence. The essence however is that it is looking for registered pulses.

When the AlienVault API is utilized for the retrieval of file hash intelligence the information, we are interested in mainly two fields. We are interested in if there are any registered pulses on the file hash and which type of malware, or potentially malicious program/utility this file hash is related to. These fields are taken from the response and handled in our backend to determine if a file is related to a malicious activity or not.

The utilization of AlienVault for URL and domain intelligence gathering, works much in the same way as file hash. The URL intelligence mainly utilizes the field that contains registered pulses and whitelists. The backend functionality determines if a URL or domain poses a risk or not based on if there are any registered pulses on the URL or domain, and on if it has been added to any whitelists. If the URL or domain are included in a pulse and not in a whitelist the backend translates this as a “risk” which it displays to the user. If a URL or domain is also added to any whitelist, the backend deems the URL or domain to be safe. It is worth to note that, because the intelligence that the application utilizes to deem something malicious or safe is heavily reliant on user generated pulses, there is a possibility of the generation of false positives. However, the generation of false negatives should not be possible, unless there is a lack of data from AlienVault. Moreover, the reason why we have multiple intelligence sources, to be able to deliver extra certainty in the analysis of URLs, domains and file hashes.

The quota for the AlienVault API is set to 10 000 requests per hour, which is more than enough for the Threat Total application, especially when coupled with the Redis caching solution, see [4.3.3](#).

6.2.4 VirusTotal

VirusTotal is a free intelligence platform that analyses files and URLs for malicious items. The endpoint utilizes the collaboration with antivirus engines to produce scanning results of malicious content, such as URL, domains, hashes, and files. We utilized the VirusTotal API v3 endpoint directly for the file analysis functionality in the Threat Total application. The reason why we chose this endpoint for our file analysis, is that the website follows the REST principle and returns predictable JSON data. Using the REST principles, it is possible to extract the necessary fields for creating a verdict to display in a high-level language. Additionally, it was relatively easy to implement using the API v3 documentation, as VirusTotal provides some basic code in various languages that allows for testing the API. Even though the free version of the VirusTotal API only limits our requests to four requests per minute, it is sufficient to display a proof of concept. Though this is not sufficient for production use. As the ideal way to create the file upload functionality, would be to send and run uploaded files in a local malware environment. This would limit the potential of a data leak. The files uploaded from users of the Threat Total application can be anything, and files uploaded to the VirusTotal malware scanner is outside of our control. File analysis was intended to be done in a local malware environment, accessed through the NTNU SOC’s API. However due to the situation resulting in us not having access to the API, we instead implemented similar functionality by using the VirusTotal API v3 to show a proof of concept.

6.3 Login Functionality

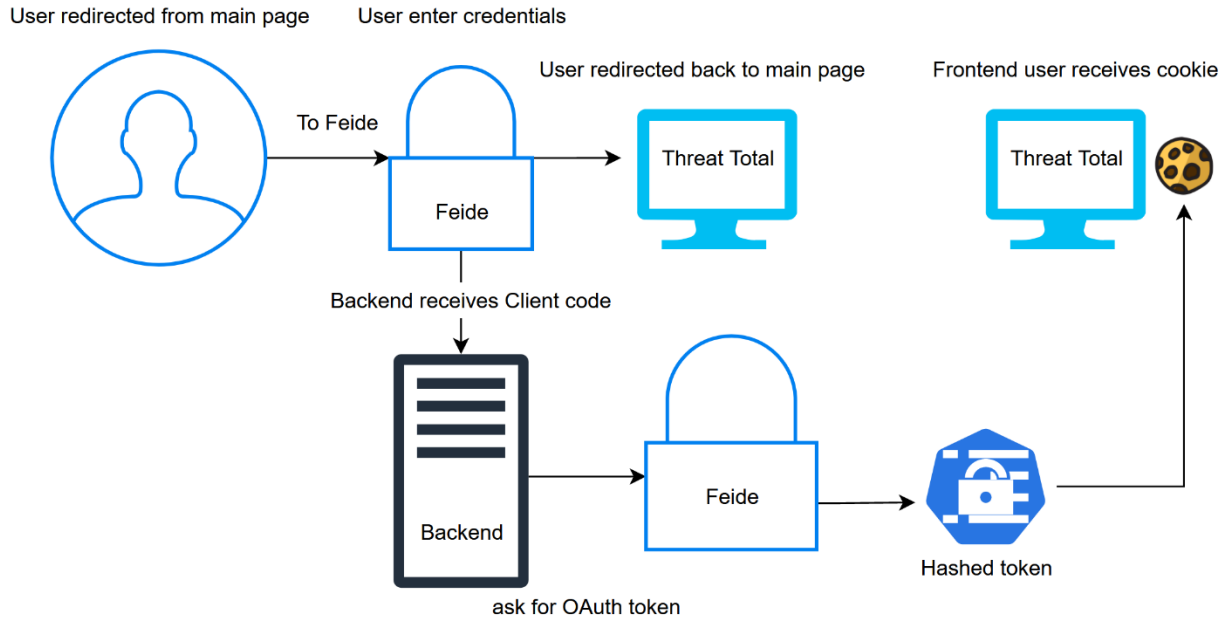
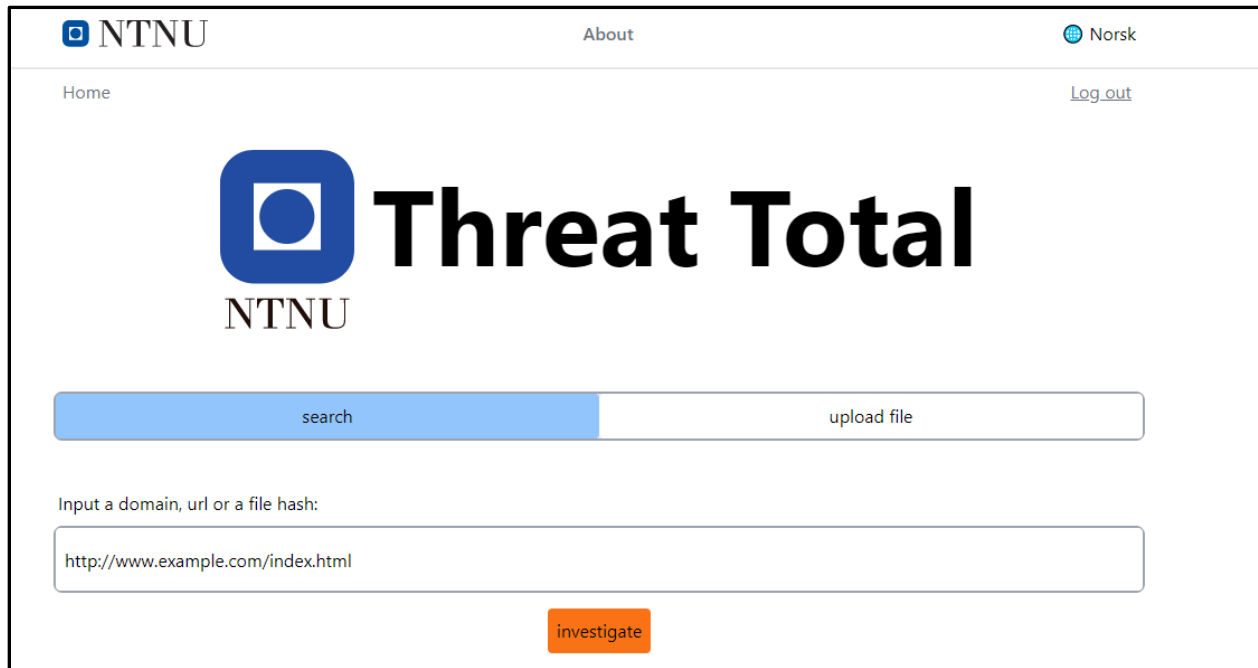


FIGURE 10: FEIDE AUTHENTICATION IMPLEMENTATION

Another requirement was to implement Feide authentication to secure the application. To log in, the user presses a log in button which redirects the user to Feide’s log in page. This page returns a code which is then forwarded to the backend. The backend uses the code to retrieve an id token and a JWT (JSON web token) with user information. The token is then hashed and stored in Redis with the user information. Finally, the hashed token is sent to the user again and stored as a cookie. This hashed token is then used to authenticate the user for the duration of the token’s lifespan.

6.4 URL and Domain Search



The screenshot shows the Threat Total web application interface. At the top left is the NTNU logo, and at the top right are links for "About" and "Norsk". Below the navigation bar, there are links for "Home" and "Log out". The main heading is "Threat Total" with the NTNU logo to its left. Below the heading is a search bar with two tabs: "search" (selected) and "upload file". Underneath the search bar is a text input field with the placeholder "Input a domain, url or a file hash:" and the example URL "http://www.example.com/index.html". Below the input field is an orange button labeled "investigate".

FIGURE 11: EXAMPLE OF INVESTIGATING A URL

When the user inputs a URL or domain in the search field as seen in figure 11 above and clicks “investigate”, the user is redirected to the “results” page. Input data is then fetched from the form by the “result” function in the frontend. The frontend then fetches the “URL-intelligence” endpoint in the REST API and passes the user authentication token and the submitted URL or domain as URL parameters. The API endpoint then checks whether the user that made the request is authenticated or not, before processing the actual URL or domain submitted in the request.

If the user has been authenticated, all functionalities of the web application will be unlocked for the user. The backend utilizes several public APIs to gather intelligence about the content. The backend uses the following APIs: “Google Safe Browsing”, “AlienVault” and “Hybrid Analysis”, which gathers data from “urlscan.io” and “VirusTotal”. This means that when you look up a URL or domain in the Threat Total application, the URL is cross-referenced with more than 70 antivirus agents, and several intelligence lists and blocklists. More on the data gathering methods from the different intelligence sources can be found in [chapter 6.2](#)

```

type HybridAnalysisURL struct {
  SubmissionType string `json:"submission_type"`
  ID              string `json:"id"`
  Sha256         string `json:"sha256"`
  Scanners       []struct {
    Name           string      `json:"name"`
    Status         string      `json:"status"`
    ErrorMessage   interface{} `json:"error_message"`
    Progress       int         `json:"progress"`
    Total          int         `json:"total"`
    Positives      int         `json:"positives"`
    Percent        int         `json:"percent"`
    AntiVirusResults []interface{} `json:"anti_virus_results"`
  } `json:"scanners"`
  Whitelist []interface{} `json:"whitelist"`
  Reports   []string      `json:"reports"`
  Finished bool          `json:"finished"`
}

```

FIGURE 12: GOLANG DATA STRUCTURE FOR HYBRID ANALYSIS

When the data has been retrieved from the different public intelligence sources, it is stored in data structures depending on the intelligence source. An example of a data structure can be seen in figure 12 above. This data structure is utilized when data is retrieved from the Hybrid Analysis’ API.

```

func SetResponseObjectGoogle(jsonResponse GoogleSafeBrowsing, response *FrontendResponse2) {
  if len(jsonResponse.Matches) != 0 {
    response.EN.Content = "This URL has been marked as malicious by Google Safebrowsing, visiting is NOT recommended"
    response.NO.Content = "Denne URLen har blitt markert som ondsinnet av Google Safebrowsing, besøk er IKKE anbefalt"
    switch jsonResponse.Matches[0].ThreatType {
    case "MALWARE":
      response.EN.Status = "Risk"
      response.NO.Status = "Utrygg"

      response.EN.Tags = "MALWARE"
      response.NO.Tags = "SKADEVARE"

    case "SOCIAL_ENGINEERING":
      response.EN.Status = "Risk"
      response.NO.Status = "Utrygg"

      response.EN.Tags = "SOCIAL_ENGINEERING"
      response.NO.Tags = "SOSIAL_MANIPULERING"
    }
  }
}

```

FIGURE 13: HOW THE USER RESPONSE IS MADE BASED ON VERDICT FROM GOOGLE SAFE BROWSING

The data in the structures are then parsed by helper functions which determines if the data indicates that the URL or domain is malicious or not. Separate functions for each public intelligence source handles this process and adds the processed data to a “Frontend response” data structure, which contains the data from all intelligence sources. The example code snippet above shows how the “SetResponseObjectGoogle” function processes different output from the Safe Browsing API, and deems a URL or domain malicious or not, before adding this data to the “Frontend Response” data structure.

Implementation and Production Process

The frontend response is the data structure that will be passed back to the frontend and displayed to the user. After all the individual sources has been parsed and the data has been added to the frontend response, the function “SetResultHash” is called. This function checks all the gathered data for any indication deeming that a URL or domain is malicious and stores the verdict in a result string which is displayed to the user. An image of this function is shown in the image.

```
func SetResultHash(Responses *ResultFrontendResponse, size int) {  
  
    for i := 0; i <= size-1; i++ {  
        if Responses.FrontendResponse[i].EN.Status == "Risk" {  
            Responses.EN.Result = "This file hash has been marked as malicious by atleast one of our threat intelligence sources."  
            Responses.NO.Result = "Denne filhashen har blitt markert som ondsinnet av minst en av våre trusselkilder."  
        }  
    }  
    if Responses.EN.Result == "" {  
        Responses.EN.Result = "We do not have any intelligence indicating that this file is malicious."  
        Responses.NO.Result = "Vi har ingen informasjon som tilsier at denne filen er ondsinnet"  
    }  
}
```

FIGURE 14: FUNCTION WHICH SETS RESPONSE BASED ON SOURCE STATUS

After all the parsing of data is complete, a screenshot is gathered from the URL or domain that is requested an analysis of. The screenshot is taken by using the “chomedp” GitHub package, which utilizes Chrome in headless mode to visit the URL or domain to take a screenshot. Information on this functionality and the risk can be read about in [chapter 8.9.2](#)

```
func checkIfIntelligenceComplete(jsonData utils.ResultFrontendResponse, size int) (complete bool) {  
    complete = true  
  
    for i := 0; i <= size-1; i++ {  
        if jsonData.FrontendResponse[i].EN.Status == "Awaiting analysis" || jsonData.FrontendResponse[i].EN.Status == "Error" {  
            complete = false  
        }  
    }  
  
    return complete  
}
```

FIGURE 15: FUNCTION FOR CHECKING IF THE PROCESS OF GATHERING THREAT INTELLIGENCE IS FINISHED

After the data has been set in the “Frontend response” structure and the screenshot has been gathered, it must be determined if the data is complete and ready to be cached or not. The function called, “checkIfIntelligenceComplete”, does this and can be seen in the figure above. If the function determines that one of the public endpoints has not yet completed their information gathering or has encountered an error during data gathering process, the data will not be cached before it is displayed to the front user. This is a measure we have implemented in case of smaller downtimes or minor errors in one or more of our public intelligence sources. Or if one of our sources uses too much time to gather the data. It would not be correct if we had a cache containing information about a data source having an error when the data source is able to give proper data.

After the data has been cached or not it is returned to the frontend “result” function which displays the data to the user. The image below shows an example of how a response can look

Implementation and Production Process

like when investigating a URL. The response includes an overall assessment if the visiting this URL is advised or not in the upper left corner. Next to it is a screenshot of how the user interface of the web site looks like. Below these sections is the response from all the antivirus agents that we use, and verdict based on their threat intelligence.

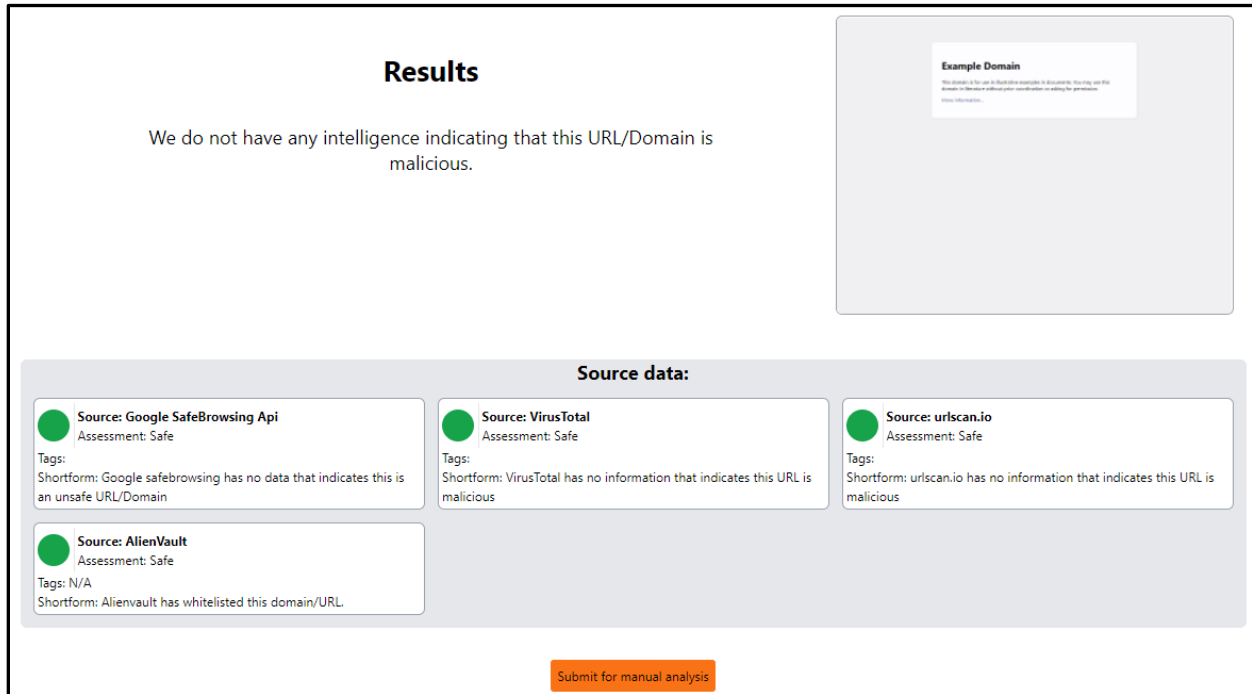


FIGURE 16: EXAMPLE OF HOW THE RESULT PAGE COULD LOOK LIKE

6.5 File Hash Search

The file hash search functionality is much like the search URL or domain functionality. The user adds the file hash they want to know information about in the search field on our frontend, as seen with a URL in [figure 11](#).

The content of the search field is then parsed by the frontend “result” function and a GET request is made to the “hash-intelligence” endpoint in our REST API. The REST API then checks if the user is authenticated before it parses the content of the request.

If the user has been authenticated, the backend gathers data from both Hybrid Analysis and AlienVault about the file hash. If the file hash is known to any of the intelligence sources, it will return which file and or malware the file hash has been associated with if any. The returned data from the intelligence sources are added to individual data structures before being parsed and added to the frontend response structure. This can be seen in the figure below an example of data parsing of file hash analysis from Hybrid Analysis.

```
if jsonResponse[0].Verdict == "malicious" {
    response.EN.Status = "Risk"
    response.EN.Content = "This file is recognized as malicious."

    response.NO.Status = "Utrygg"
    response.NO.Content = "Denne filen er gjenkjent som ondsinnet."
    //response.SourceName = jsonResponse.Submissions[0].Filename
} else if jsonResponse[0].Verdict == "whitelisted." {
    response.EN.Status = "Safe"
    response.EN.Content = "This file is known to be good - whitelisted."

    response.NO.Status = "Trygg"
    response.NO.Content = "Denne filen er hvitelistet av HybridAnalysis - Ikke ondsinnet."
    //response.SourceName = jsonResponse.Submissions[0].Filename
} else if jsonResponse[0].Verdict == "no specific threat" {
    response.EN.Status = "Safe"
    response.EN.Content = "According to HybridAnalysis does this file not pose any specific threat."
```

FIGURE 17: DATA PARSING FOR HYBRID ANALYSIS FILE HASH SEARCH

After being parsed individually, the data is checked for any indication of being malicious before the result string is set. The result string is displayed to the user and is a summary of the detections made by the intelligence sources. This is seen in the figure below.

```
func setResultHash(Responses *ResultFrontendResponse, size int) {
    for i := 0; i <= size-1; i++ {
        if Responses.FrontendResponse[i].EN.Status == "Risk" {
            Responses.EN.Result = "This filehash has been marked as malicious by atleast one of our threat intelligence sources visiting is not recommended"
            Responses.NO.Result = "Denne filhashen har blitt markert som ondsinnet av minst en av våre trussetetteretningskilder, besøk er ikke anbefalt."
        }
    }
    if Responses.EN.Result == "" {
        Responses.EN.Result = "We do not have any intelligence indicating that this filehash is malicious."
        Responses.NO.Result = "Vi har ingen informasjon som tilsier at denne filhashen er ondsinnet"
    }
}
```

FIGURE 18: "SETRESULTHASH" WILL SET THE USER RESPONSE BASED ON VERDICT FROM ANTIVIRUS AGENT

After the validation of the frontend response is complete, there is a final check to determine if the data is "complete". This means that the data does not contain any errors or incomplete analysis. This is checked to determine if the data should be cached or not. This is done to avoid the possibility of caching errors, or analysis that is not yet completed. This functions the same way for both hash intelligence and for URL/domain intelligence. See [figure 15](#). After the data has been cached, or not, the backend returns the frontend response struct to the frontend. This data structure is parsed into result data and displayed to the user. See figure below.

Implementation and Production Process

Resultater

Denne filhashen har blitt markert som ondsinnet av minst en av våre trusseletteretningskilder, hvis du har denne filen på datamaskinen anbefaler vi å slette filen og kjøre en full antivirus skann av maskinen.

Kilde data:

<div style="border: 1px solid #ccc; padding: 5px;"><div style="display: flex; align-items: center;"><div style="width: 15px; height: 15px; background-color: yellow; border-radius: 50%; margin-right: 5px;"></div><div><p>Kilde: Hybrid Analysis</p><p>Vurdering: Ukjent</p><p>Tags:</p><p>Kortform: Denne filhashen er ukjent for Hybrid Analysis.</p></div></div></div>	<div style="border: 1px solid #ccc; padding: 5px;"><div style="display: flex; align-items: center;"><div style="width: 15px; height: 15px; background-color: red; border-radius: 50%; margin-right: 5px;"></div><div><p>Kilde: AlienVault</p><p>Vurdering: Risk</p><p>Tags: Malicious</p><p>Kortform: Mimikatz</p></div></div></div>
---	---

[Send inn til manuell analyse](#)

FIGURE 19: FILE HASH SEARCH RESULTS

6.6 File Upload Process

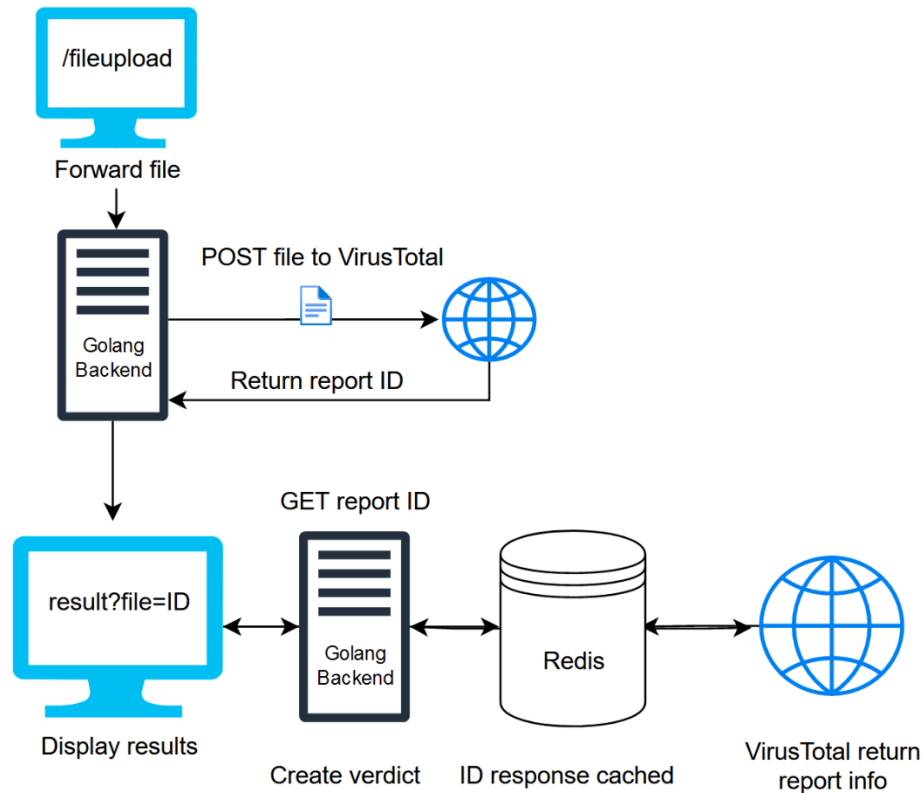


FIGURE 20: FILE UPLOAD PIPELINE

The file upload pipeline involves using the VirusTotal API v3 to receive a file scan report, which is then displayed on the frontend. After a user has logged in and navigated to the “upload” page, the user is presented the option to upload a file for analysis. Assuming the user has selected and clicked the “investigate” button, the pipeline for the file upload functions has started. As described in the image above, the file is forwarded to the backend. The backend prepares a POST request to the VirusTotal API. In the response, after the file has been analysed, the backend decodes the output, extracts, and returns an MD5 hashed ID. The id is then used for fetching scan reports from VirusTotal.

After the return, the user is redirected to the “result” page and the ID is sent back to the backend. If the data on the ID is not already cached, the ID is used to prepare a GET request to get the scan report information from VirusTotal. The backend then prepares a verdict, using the fields found in the report, to create a summary of the scan results in a high-level language. The verdict is based on different comparisons of the fields and is customizable. Meaning that developers who later pick up this tool, can also make more accurate assessments by adding more comparisons or create different verdict results. The results are then cached with Redis and then presented to the end user. The user receives a total verdict of the analysis and can observe results from every antivirus engine VirusTotal returned information from.

This example also presents how efficient caching is, as it shows how we can skip performing a request back to VirusTotal by using the cached data from previous requests.

6.7 Escalate to Manual Analysis

After an analysis, the results page contains an option for the end user to click the “Submit for manual analysis” button. See [figure 19](#). This button activates a function which retrieves the users email from the cache based on their user authorization token, and then sends a confirmation email back to the user. This email contains information about the request and how further handling will be done. In addition, the frontend displays an alert to the user that an email has been sent to their e-mail address. Further correspondence in reference to manual analysis can then be handled by e-mail.

6.8 Frontend Structure

The frontend was written using a combination of plain JavaScript, HTML, ReactJS and TailwindCSS. We structured the ReactJS project into pages, components, and resources. The pages contain the base structure of each of the different webpages. The components are reusable pieces of code which are used in different pages. For example, the navbar is a component which is included in every single webpage on the site. This component structure allows for modularity and simple reuse of parts where relevant. Having reusable components and shared imports makes the codebase simpler to view, manage and maintain. It also reduces duplicate code, as the duplicates can instead be turned into components which can then be used by several pages or components.

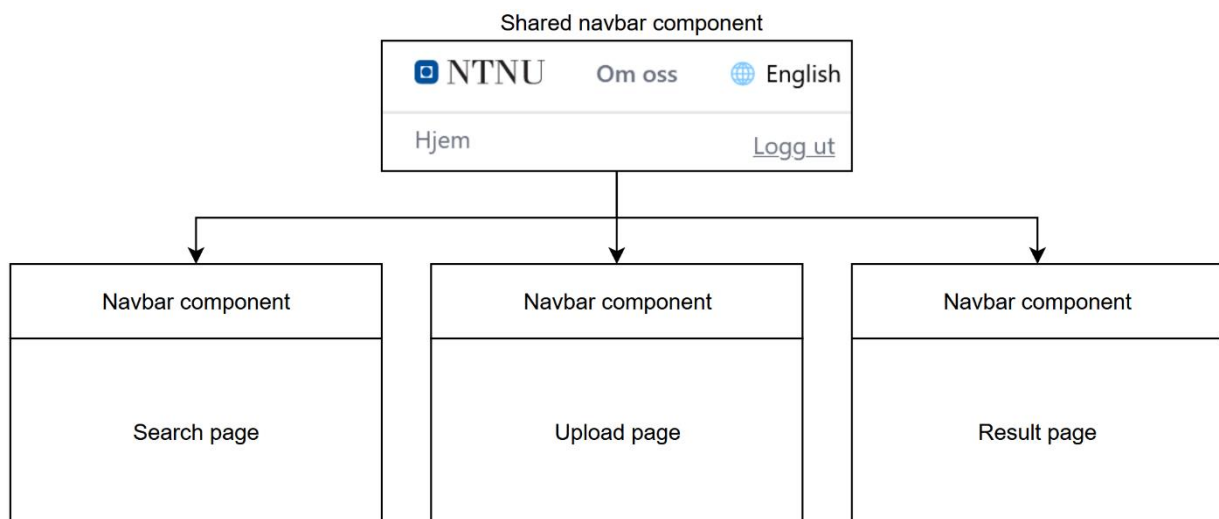


FIGURE 21: FRONTEND COMPONENT SHARING

6.9 Translation

Since one of the requirements of the application was to have both English and Norwegian versions of each of the different webpages, we ended up using the `i18next` package for translation. With this package each piece of plain text on the different webpages are replaced with text from a JSON object.

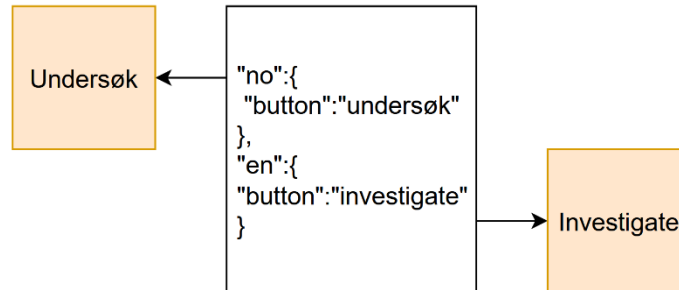


FIGURE 22: EXAMPLE OF A BUTTON BEING TRANSLATED WITH JSON DATA

This JSON object is further divided into namespaces and languages. So, each piece of text on the website has two different versions corresponding to the different languages we support, and the text will update in real-time upon the press of the language change button. The main advantage to this approach is that the translations are easy to manage as they are in a simple JSON data format, it is extendable, as more languages can simply be added to the JSON object and finally the translations can be applied without refreshing the webpage. Which leads to a better user experience and reduces the number of unnecessary requests to the backend.

7 Code Review and Code Quality

7.1 Introduction

This chapter contains information about the structure of the application when it comes to modularity within the project repository. Information about usage of computer threads, caching and code optimization can also be found under this chapter.

7.2 Component Structure

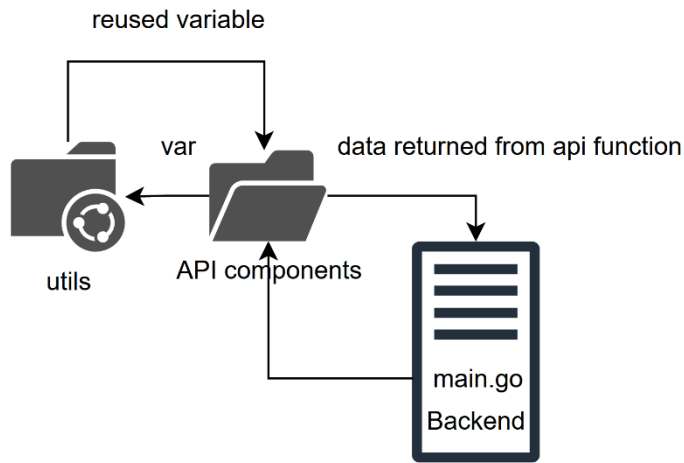


FIGURE 23: COMPONENT STRUCTURE, MADE IN DRAW.IO

As we wanted the project to have an understandable structure to make it accessible, we focused on structuring the code. In the backend we structured different parts of the project into logical packages. As an example, the API functions were structured in the “api” package, while the global variables such as constants and structs were gathered in the “utils” package. This allowed us to reuse variables across different packages and helped make the structure readable and accessible.

Code Review and Code Quality

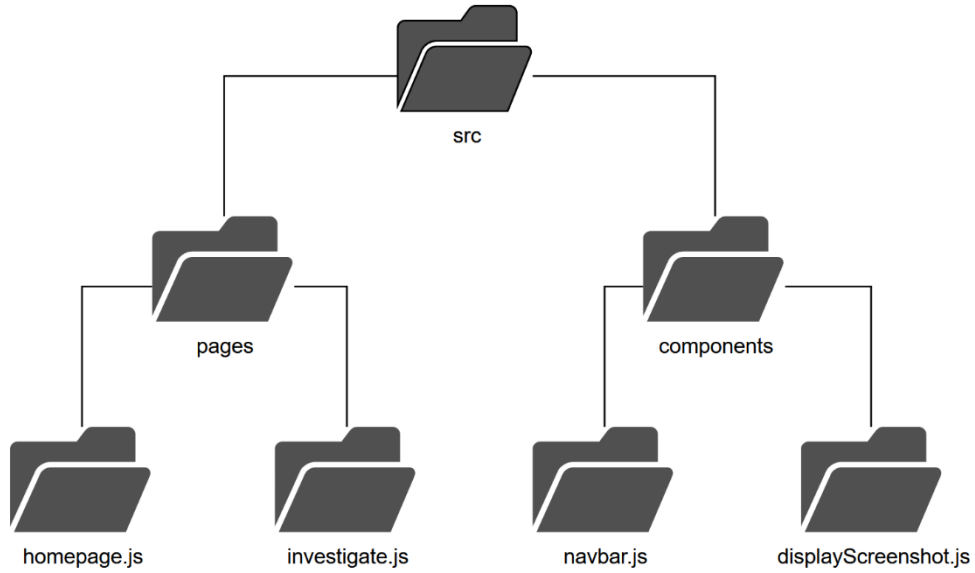


FIGURE 24: IMAGE OF STRUCTURE, MADE IN DRAW.IO

In the frontend the webpages were located together in the “pages” subfolder while components used by the different webpages were in the “components” folder.

7.3 Increasing Performance with Threading

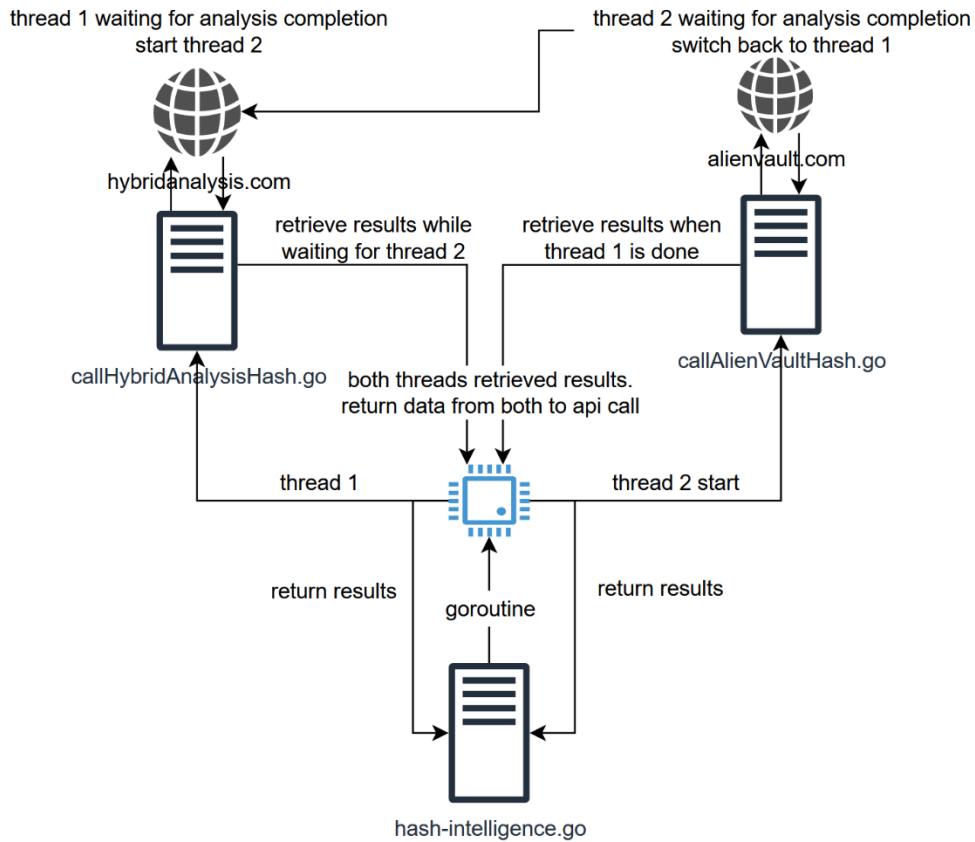


FIGURE 25: VISUALIZATION OF THREADING, MADE IN DRAW.IO

Because of the processing time of one of our data sources we had to implement a wait time, this in addition to processing the different data requests sequentially resulted in long wait times. To reduce wait times for the requests, we then ended up implementing goroutines which are lightweight threads in Golang. This allowed the different requests to run in parallel, which reduced the processing time.

7.4 Efficient Code with Caching

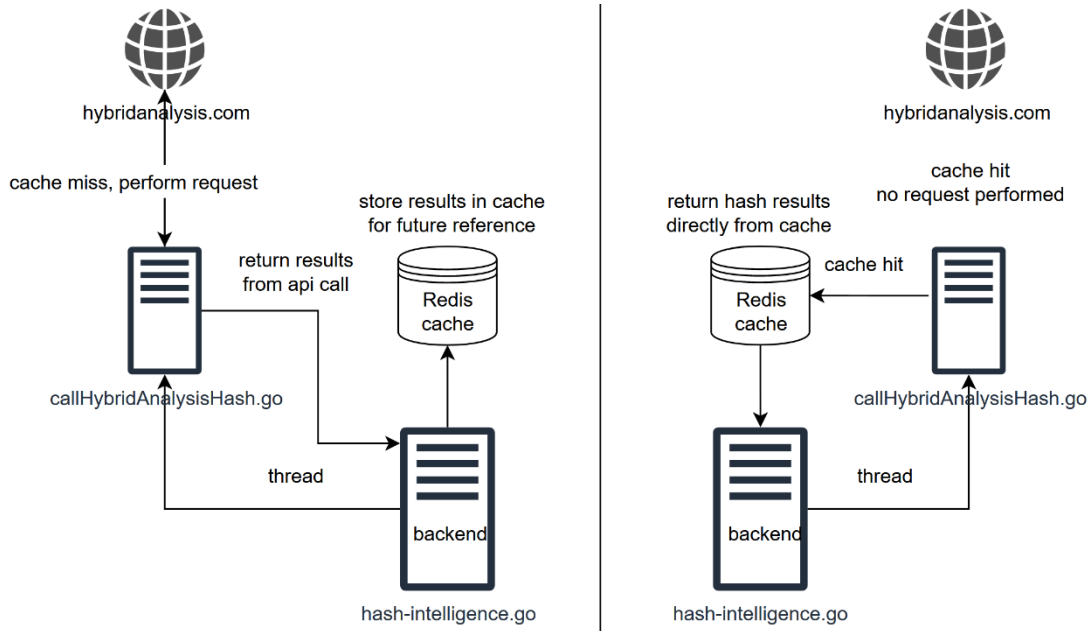


FIGURE 26: CACHE MISS VS CACHE HIT. VISUALIZATION MADE IN DRAW.IO

Another step we made to reduce wait times for the end user is to implement caching. Once a request has been made the response data is cached in Redis with a configurable cache duration. This makes it so that the wait time for common requests go down drastically, while also reducing load and requests to our data sources.

7.5 Code Modularity

```

209 r.POST("/upload", func(c *gin.Context) {
210
211     hash := c.Query("userAuth")
212
213     authenticated, _ := auth.Authenticate("", hash)
214     if !authenticated {
215         c.JSON(http.StatusUnauthorized, gin.H{"authenticated": "You are not authenticated. User login is invalid."})
216     } else {
217         log.Println("Fileupload worked")
218         logging.Loginfo("Fileupload worked")
219
220         uri := "https://www.virustotal.com/api/v3/files"
221         body := &bytes.Buffer{}
222         writer := multipart.NewWriter(body)
223
224         // fetch the file contents
225         file2, _ := c.FormFile("file")
226         // open the file
227         file3, _ := file2.Open()
228     }
229 } // 80 more lines of code below...
230
231
232
233
234
235

```

FIGURE 27: EXAMPLE OF CODE BEFORE REVISION

```

209 r.POST("/upload", func(c *gin.Context) {
210
211     hash := c.Query("userAuth")
212
213     authenticated, _ := auth.Authenticate("", hash)
214     if !authenticated {
215         c.JSON(http.StatusUnauthorized, gin.H{"authenticated": "You are not authenticated. User login is invalid."})
216     } else {
217         api.UploadFile(c)
218     }
219 })

```

FIGURE 28: EXAMPLE OF CODE AFTER REVISING CODE MODULARITY

Code modularity is also an important concept for making structured and understandable code. Using code modularity, in addition to actively revise the code with this method, contributes to making good quality code. Take the above picture as an example, where we initially had a sizeable chunk of code in which was difficult to read and follow. Separating the code packages allowed for better readable and understandable code. Sometimes this also deems as necessary for the overall programming logic. This is something which our project experienced, as the file upload functionality needed a handle for GET requests and another handle for POST. Separating this, not only made the code more readable, but also made the planned logic work as intended.

8 Testing and Quality Assurance

8.1 Introduction

This chapter contains information about how the team went forward about testing the application regarding usability and functionality, as well as analyzing the code. The code alone was analyzed by using a publicly available static analyzer and provides information about the code's security and quality. The best way of testing the usability of an application is through human interaction. The team have done this to get a clearer picture of our application's level of usability and a good user experience.

8.2 Static Code Analysis

Static code analysis is the process of debugging the source code of an application with automated tests and a set of rules for the tests to follow. [31] This type of code analysis is good to detect vulnerabilities in the source code, as most tools have these sets of rules already built in. In difference to unit testing, static analysis is performed before the program is being run and can provide the information about flaws and defects that way.

Static code analysis brings along some limitations, as well as pros and cons. Some built in rules that is used by the analyzer can interpret the commenting in the code according to a specific set of standards, while the commenting has been done according to another standard. Other scenarios can cause false positives and negatives. The analyzer will then probably report a possible defect in that area, but no more specific reasoning.

The greatest benefits of using such a tool are the speed, depth and accuracy compared with manual code reviews. Time is a valuable resource while developing a web application. By using automated tests, possible problems can be reported much earlier than by doing manual code reviews, and therefore also lead to less errors. Unlike manual code reviews where every possible execution path will both take time and knowledge, static code analyzers will execute every path and provide a report with good readability. Manual code reviews are prone to human errors. Static analysis tools are therefore also a more accurate solution than manual code reviews.

8.3 SonarQube

SonarQube is a popular static code analyser that we chose to use. The tool will provide the user with information about the security and code quality of the codebase which can be quite handy in code reviews. SonarQube supports 27 programming languages, and some of them are those that we use. [3] SonarQube provides two different solutions to deploy the service, either by downloading it and all its dependencies, or download a docker image and run it. After fiddling around with the first method for several hours, and not being able to make it work, we decided to try the docker image and instance which was a success. The next thing to do was to open the instance in the browser at “<http://localhost:9000>”, log in with the provided username and password and make a new project for which will store the finished reports. All the code files are then analysed with the tool called “sonar-scanner.bat”. This tool sends the results to the newly established project in SonarQube’s web browser interface. The image below is an example of a SonarQube report:

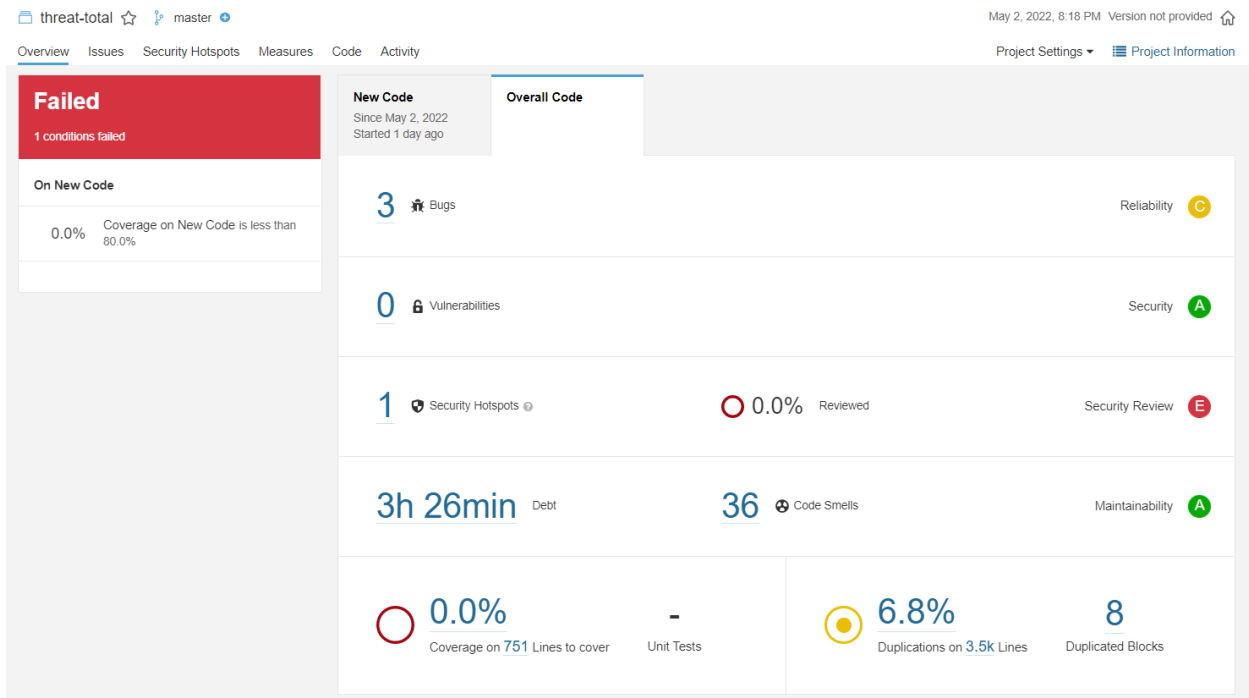


FIGURE 29: EXAMPLE OF SONARQUBE REPORT.

SonarQube will also have more extensive notes for each of these categories by clicking on the respective category that the user wants to know more about. By clicking on the “1” in the same column as “Security review”, the user can sort out why the report received an “E” in that category:

The screenshot shows a SonarQube security hotspot interface. At the top, a message reads: **"Password" detected here, make sure this is not a hard-coded credential.** Below this, it states: "Hard-coded credentials are security-sensitive" with a link to `go:S2068`. The status is **TO REVIEW**. A description says: "This security hotspot needs to be reviewed to assess whether the code poses a risk." There is a **Change status** dropdown and an **Assignee: Not assigned** field with an edit icon.

Below the description are four tabs: **Where is the risk?**, **What's the risk?**, **Assess the risk**, and **How can you fix it?**. The **Where is the risk?** tab is active, showing a code editor for `storage/redis.go`. The code contains the following lines:

```
7      "github.com/gomodule/redigo/redis"  
8    )  
9  
10   // Alternatively use https://github.com/go-redis/redis  
11  
12   // TODO: move these to the utils library when done testing  
13   const Host = "localhost"  
14   const Port = "6379"  
15  
16   // TODO: password in authentication file or as system args?  
17   const Password = "admin"
```

A red box highlights the line `const Password = "admin"` with the message: **"Password" detected here, make sure this is not a hard-coded credential.** A **Comment** button is visible to the right of the message.

FIGURE 30: EXAMPLE OF A DETECTED SECURITY FLAW.

In this case, SonarQube detected the variable, “Password”, and a hardcoded value to it, which is a problem with high priority according to SonarQube. This value was used for testing and development and was later changed to an environment variable so that it was not hard-coded and not accessible from the git repository. Another reason for the failed test, was the lack of unit testing. A built-in rule is the coverage of unit testing. If this percentage is not past a certain level, the analysis will also automatically receive a “Failed”.

During the project development, the team stumbled upon an example of a not so uncommon security incident, which is data leakage. In the context of testing, there was performed some hardcoding of API keys into variables, without necessarily realizing the potential consequences of this later. After performing the required change in the code, the code was committed and pushed to the team’s GitLab repository. The team then realized the API key was now leaked through the commit history in GitLab. In the hands of an attacker, this can lead to API abuse. The team handled this incident professionally, by removing the hardcoded variables and deactivating the account linked to the leaked API keys. Additionally, the team moved on to using key files for API keys. These files were put in the “.gitignore” so they were not a part of the version control system. Later this was replaced with using environment variables for the different secrets and API keys. As described above, using tools such as SonarQube, can aid in detecting mistakes like this and help prevent similar incidents from occurring.

8.4 Usability Testing

Testing is a critical part of web application development as it will provide feedback in form of things that can be improved for a better user experience. [12] Usability testing is quite relevant part of testing when it comes to feedback on the user experience. This type of testing is a non-functional testing method and is part of a user-centred design philosophy. The testing is done with real users in real scenarios. Better understanding of issues and concerns regarding the usability from the test subject, the better the developer or designer will become at correcting them.

Usability testing comes with its pros and cons. On the one hand, testing helps the developer or designer understand the users' needs and points at frustrating things as mentioned earlier. [4] The discoveries of hidden issues and anomalies can easily be unrevealed with real users in real scenarios. On the other hand, selecting a target group can be tricky. As a rule of thumb, the testing should be done with a sizeable audience, both when it comes to age and technical experience. Usability tests are also more tedious and time demanding than questionnaires as they need to be done manually.

8.4.1 Results From Usability Testing

By performing user testing we were able to uncover some errors regarding logging in with Feide, investigating a URL and investigating a file hash. View the appendices for the complete spreadsheet from the usability testing. The table below contains the result from the testing:

Person	Age	Technical knowledge	Successful test	Comments
1	22	Intermediate	Yes	Overall good and structured layout. Easy to navigate. An improvement to implement could be to add a text when loading investigation results. The text should state the expected amount of time for the user to be waiting before receiving a result. There were no problems during the tasks for the usability testing.
2	25	Intermediate	Yes	The web page is not hard at all to navigate. Everything is provided in a clean way on the homepage. There were no problems during the tasks for the usability testing.
3	26	Advanced	Yes	No problems with executing the tasks given in the usability testing. The application's homepage looks clean and can easily be navigated. As of now the application is not well optimized for a wide computer monitor ratio.

TABLE 16: USABILITY TESTING

8.4.2 Weaknesses

The testing was done with a limited audience and will hence not provide as broad of response, as if the audience consisted of a wider selection based on age and technical knowledge. This can be seen as a source of error. This is thus explained previously as some of the general weaknesses with this kind of testing. By having such a limited selection, we could also miss features which different aged audiences would appreciate.

8.5 Regression Testing

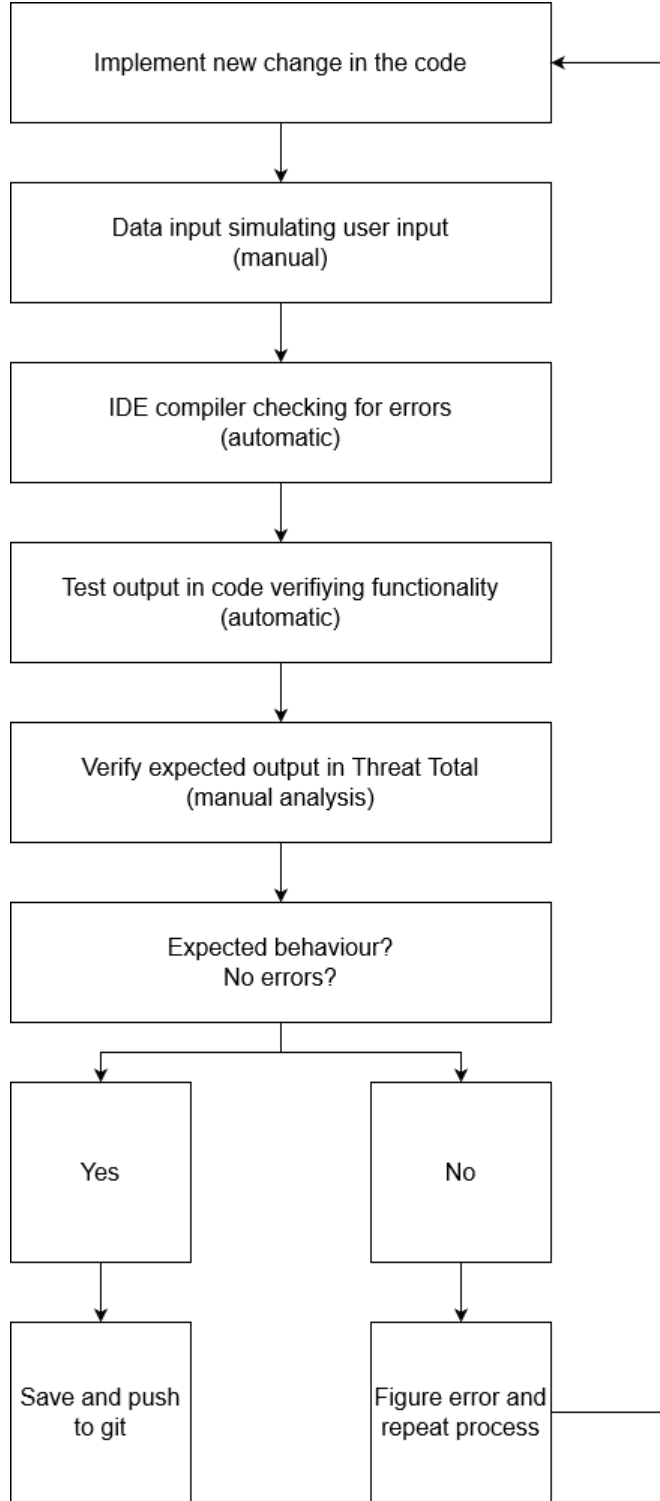


FIGURE 31: REGRESSION TESTING FLOWCHART - MADE IN DRAW.IO

During the development phase of the project, most of the feature implementations were tested by semi-automated regression testing. This involves the process of running a test of the function before and after an implementation change. Evaluating a component with regression testing, involved using the same input data, as this allowed for a measurable consistency. This way, it was easy for the developer to know when a change would break the functionality of a component. The regression testing process was semi-automated, and the workflow is visualized in the image above. A team member would implement a new change to the code and the compiler checks for any syntax errors. The member would then navigate to the already booted up application in the browser, and input data that would make sense as an end user. Followingly would the team member check the log for any unexpected output or crashes and verify if the dataflow went through the application was as expected. If the application did not crash or have any unexpected output, the developer can then safely commit and push to the git repository with the updated change. If the application did not behave as expected, the developer would instead start troubleshooting and repeat the process.

8.6 Destructive Testing

Destructive testing is the process of actively trying to break the application. The goal here is to figure out what input data leads to breaking the application. This can be used to remediate possible bugs and vulnerabilities that may uncover from application breaking inputs. An example of destructive testing that we found during the implementation of the URL, domain and hash uploading functionality, was that if we added any trailing whitespaces after the data input, the dataflow would cause a false-negative return output on one of the public antivirus engines. This was because we did not have any functions that would correctly parse the input of the query parameter, for instance a hash to search for. Thereafter, the way we handled the parameter, it would then send an invalid string to search for through the public antivirus sources and resulted in an error that would cause the results from Hybrid Analysis to become invalid. This is an example of how effective destructive testing can be, as we would have never found this bug with using valid inputs only. In addition, this would be a common mistake for the average front-end user, where the user could potentially enter the data input and mistakenly press spacebar to add a whitespace at the end of the input. In production, this bug would have been a negative impact back to the end-user, as it would return a false negative as a result. This would potentially present false data to the end user and create a deficient verdict. This could have been critical, as if the file would have been indeed malicious, the user would not have known. In the end, this bug was simply fixed by using Golang's built-in functions for parsing input data correctly. Another way to find such errors is to use property tests, where the test inputs random data to try to find errors in the functions tested.

8.7 Unit and API Testing

This project has incorporated API tests that both test components of our functionality and the full functionality of our API endpoints. The implemented testing focuses primarily on the “/url-intelligence” endpoint, the “/hash-intelligence” endpoint and on authorization. These endpoints contain the functionality of delivering an analysis on submitted URLs and Domains and delivering an analysis of a submitted file hash respectively.

The full list of our implemented tests can be viewed in the table below:

Test function name	Type	Description	What is tested?
TestUrlIntelligenceOK	API	Test that checks if the url-intelligence endpoint returns the expected return code 200 if the user requests analysis as an authenticated user.	Return code 200 as an authorized user.
TestUrlIntelligenceUnauthorized	API	Test that checks if the authorization block on the endpoint functions properly, this test attempts to perform a request to the URL-intelligence endpoint unauthenticated and expects a return code of 401.	Return code 401 as an unauthenticated user.
TestHashIntelligenceOK	API	Test that checks if the hash-intelligence endpoint returns the expected return code 200, if the user requests analysis as an authenticated user.	Return code 200 as an authenticated user
TestHashIntelligenceUnauthorized		Test that checks if the authorization block on the endpoint functions properly, this test attempts to perform a request to the hash-intelligence endpoint unauthenticated and expects a return code of 401.	Return code 401 as an unauthenticated user.
TestUrlIntelligenceValidOutput	API	This is a test that tests the validity of the output received from the URL-intelligence endpoint. Because our	- If status code is 200 - If the data can be unmarshalled

Testing and Quality Assurance

		<p>backend code is structured in packages this test will indirectly also function as unit tests for the subfunctions. This is because each part of the expected output is generated by multiple smaller functions, and by viewing the output it is possible to determine where an error lies.</p>	<p>to the struct ResultResponse</p> <ul style="list-style-type: none"> - If data can be accessed in the struct - If the first sourceName is "Google Safebrowsing API" as expected -If there is a screenshot of the requested URL -If status or content is not set in any of the data objects from the intelligence sources
<p>TestHash_IntelligenceValidOutput</p>	<p>API</p>	<p>This is a test that tests the validity of the output received from the hash-intelligence endpoint. Because our backend code is structured in packages this test will indirectly also function as unit tests for the subfunctions. This is because each part of the expected output is generated by multiple smaller functions, and by viewing the output it is possible to determine where an error lies.</p>	<ul style="list-style-type: none"> - If status code is 200 - If the data can be unmarshalled to the struct ResultResponse - If data can be accessed in the struct - If the first and second sourceName is "Hybrid Analysis and AlienVault" respectively - If status or content is not set in any of the

			<p>responses from the intelligence sources both in English and Norwegian.</p> <p>- If the status of AlienVault is risk as expected.</p>
TestNotSpecifiedEndpoint	API	This test checks if requests towards a random non existing endpoint on our API returns 404.	If the return code is 404 if the endpoint does not exist.

TABLE 17: TEST FUNCTIONS

It can be seen here that we have not implemented testing for anything related to file uploads. This is because of the complexity of the file upload endpoint. From our experience it has been easier to perform manual testing on this endpoint, and debugging is simple because of well executed logging functionality and thorough error handling, see [chapter 8.8](#). The same can be said for unit testing of smaller utility functions in our Golang backend. Most of our utility functionality handles the output data that is displayed to the user or forwards the requests for intelligence to external APIs. By checking if the output is valid through the test functions, “TestUrlIntelligenceValidOutput” and “TestHash_IntelligenceValidOutput”, we are indirectly testing the smaller utility functions, and can determine if the utility functions work as intended.

8.8 Logging in Threat Total

The Threat Total application has been implemented with both command line and global file logging of both errors and information. To create this functionality, the standard “log” library in the Golang programming language is utilized. We have defined our own “logging” package in the Golang backend which is included in every package we have created.

The error logging function utilized in the application is the “Logerror” function in the “logging.go” file, see figure below. This function takes an error and a message string as parameters. When the “Logerror” function runs, it opens and appends text to the file “errorlog”. If the file does not exist, it will be created. This function handles all types of errors in the application, such as marshalling errors and request errors.


```
//Function to handle logging of errors to errorlog file with message
func Logerror(err error, msg string) {
    // log to custom file
    LOG_FILE := "./logs/errorlog"
    // open log file
    logFile, err := os.OpenFile(LOG_FILE, os.O_APPEND|os.O_RDWR|os.O_CREATE, 0644)
    if err != nil {
        log.Panic(err)
    }
    defer logFile.Close()

    // Set log output file
    log.SetOutput(logFile)

    // log date-time, filename
    log.SetFlags(log.Lshortfile | log.LstdFlags)

    log.Println(msg, err)
}
```

FIGURE 32: LOGERROR FUNCTION

To perform information logging to a file, the function “Loginfo” is utilized. The “Loginfo” function is also defined in the “logging.go” file, see figure below. This function is very similar to “Logerror” but takes different parameters and logs to a separate file. This function takes a string as a parameter and writes logs to the “infolog” file. When the function runs, it opens and appends to the file “Infolog.txt”, and in the case of a non-existing file it creates a new “infolog” file. The information logged is the information log message, along with a timestamp. An example of a typical informational log input is when the application starts up.

```
//Function to handle information logging to infofile
func Loginfo(msg string) {
    LOG_FILE := "./logs/infolog"
    // open log file
    logFile, err := os.OpenFile(LOG_FILE, os.O_APPEND|os.O_RDWR|os.O_CREATE, 0644)
    if err != nil {
        log.Panic(err)
    }
    defer logFile.Close()

    // Set log output
    log.SetOutput(logFile)

    //log date-time, filename
    log.SetFlags(log.Lshortfile | log.LstdFlags)

    log.Println(msg)
}
```

FIGURE 33: LOGINFO FUNCTION

Logging is an asset to have in every application, and the Threat Total application is implemented with both command line logging and file logging. This makes it both possible to pipe the output from the application to a file when started from a shell, and just view errors and general information in the “errorlog” and “infolog” files respectively. The main advantages gained from logging, are for work with debugging and maintenance, as well as error tracing.

8.9 Known Vulnerabilities

8.9.1 Potential RCE vulnerability in “file.filename”

There was identified a possible remote code execution vulnerability (which will be henceforth referred to as RCE) in the file upload function. Referring to the Gin-Gonic web framework [33] and one of their issues [34], it was discovered that the “file.filename” property is not to be trusted, when handling form files. When handling a form file type, you are issued with different properties to display, such as the file size, header and filename. In the issue, there was discovered that when uploading a file to a specific destination, the file is also uploaded in the parent directory of the function. In the Threat Total code, the “file.filename” property is used when calling a function to a convenience wrapper in “CreateFromFile()”, to fetch file contents. [35] With this wrapper, the file contents are also copied and stored in memory. The multipart-form-data header is handling a file that is separated by delimiters or boundaries and is sent as a binary [36]. Hypothetically, if a threat actor were to change the filename to anything that has the capabilities of sending malicious code, this could be a potential RCE vulnerability in the code. This vulnerability is a concept of a scenario that is highly unlikely to happen but is worth noting as a possibility.

8.9.2 Potential malicious downloads from screenshot functionality

Another potential vulnerability in the application, is usage of the screenshot functionality. As the screenshot functionality is a good tool for displaying to the user what kind of website they have visited, there is a potential flaw here. The application uses the library, “chromedp” [39] for this functionality. Followingly, the library runs Chrome engine in headless mode, meaning the chrome browser is run without a user interface. The potential security flaw here, is running a browser which is visiting potentially malicious sites, on the same machine as the web application’s backend which runs important functions. The ideal setup would be to have a malware environment and run the screenshot functionality from there. This would minimize the damage from any potential infections from viewing malicious websites.

9 Installation and Realisation

9.1 Introduction

This chapter contains information about running the application and information about deploying the application into production.

9.2 Installation

To run the application, the user needs to go through some configuration steps. It is assumed that the user has already installed “node.js” and Redis. The first thing to do is to clone down a local version of the GitLab repository through SSH (Secure Shell) with the command: “git clone [git@git.gvk.idi.ntnu.no:Johannesb/dcsg2900-threattotal.git](https://git.gvk.idi.ntnu.no:Johannesb/dcsg2900-threattotal.git)”. Secondly, the node dependencies can be downloaded with “npm -i” from within the “threat-total” folder, then environment variables need to be set for both the frontend and backend as described in the project README.md file. Finally, Redis needs to be configured with a password. When the setup is complete, the frontend can be run using “npm start” from the “threat-total” folder. The backend can be run using “go run main.go”, which will also install the backend dependencies. After starting the front- and backend, the website can be accessed through a web-browser at “http://localhost:3000”. This whole setup process is described in greater detail in the readme file found in the project repository.

For deploying the code into production, the frontend can be optimized using the command “npm run build” which will build an optimized version of the whole website into the “build” folder. This data can then be used with any website hosting software such as Nginx, Apache, or any website hosting provider. As we have not implemented TLS, we recommend setting up the backend, frontend and Redis instance behind reverse proxies which will then handle TLS to secure the network-traffic in transit. Nginx for example can easily be set up as a reverse proxy with TLS. This will encrypt the traffic when it moves between the different components, which will stop simple network snooping attacks and man-in-the-middle attacks.

10 Ending Chapters

10.1 Introduction

This chapter contains closing thoughts and reflections on the overall project work, followed by an evaluation of the project. The chapter will be rounded off with a general project conclusion.

10.2 Results

The team collectively learned a lot throughout the development and documentation of the project. This includes deeper knowledge in full stack application development, limitations to different technologies we used, cache-implementations, OAuth 2.0 authentication and structuring of data flow. The final product is a functional full stack application which uses external API's, implements caching and has a user-friendly interface. The interface makes it accessible, and the aggregation of information makes using it more convenient than the alternative of visiting several different websites for the same information.

10.3 Task Critiques

10.3.1 Internal API access

We were originally told that we would get access to intelligence data on URLs, domains, file hashes and a malware environment through NTNU SOC's API. We were to base our searches on this, as well as publicly available anti-virus agents. A variety of circumstances led to this not being possible. So as of now, we only use publicly available data for testing URL's, domains, and files through hashes and uploads. The NTNU SOC had to focus resources elsewhere due to the changed threat picture from the ongoing war in Ukraine. The API access was also not secure enough, and due to the lack of resources available at the time, this error was not fixed in time. Our scope included NTNU SOC data as one of the primary elements of the project. But due to the circumstances discussed earlier, we ended up not receiving API access in time. This led to some elements of the scope being impossible to implement. To counteract this, we implemented most of the functionality which would be integrated with the NTNU SOC API as functionality connecting to third party sources, or with our own implementations. This includes for example filtering of requests where we created a proof-of-concept model with a list of URLs to filter for one of the sources as safe. This shows that the functionality can easily be implemented in the future when the API is ready. We also created a simple case system which sends email to users with information about the item sent to manual analysis, as well as information on how future contact will be made. In the end we implemented what was possible to implement given the constraints that were in place. We implemented proof-of-concept versions of what was not possible to implement fully due to these constraints.

10.3.2 Worklog time tracking

Another critique to the project work, is the method of tracking working hours. During our project, we went simplistic and attempted to track our hours within an Excel document. The idea was to take a note of when we started working, until we called it a day and to write it in the timesheet document along with a description of the work. The problem with this method, was that the team was inconsistent on noting working hours and therefore leading to inaccurate tracking of hours. Inconsistency usually appeared from not taking note of when the work period started and ended or forgetting to log the hours in the timesheet document. Looking back, the team realize they should have been using a software tool for automatic time tracking. For instance, a stamp in and stamp out service. This would have eliminated the need for a manual process of hour logging and therefore making it more consistent. The result of our current solution is an estimate of the hours committed to work and what was done. This spreadsheet can be viewed in the appendices.

10.3.3 Sprint durations

Additionally, following the scrumban framework more strictly could have made for a better plan and overview of the work-task structure. At the beginning of the project, the team followed the newly learned framework to the book. This included things such as assigning simple tasks and completing them. Though as the project began to grow larger, some assigned tasks consisted of many smaller and more “abstract” tasks. This is where the team had their own sense of whether these smaller tasks should be a new task to assign, or the task was simply too obvious bother making a new ticket on. This is where the team should have trusted the framework a bit more and started assigning new tasks and delegating them amongst the members. The essential core of scrumban is the availability for team members to perform smaller tasks over the whole board, during a sprint, without having to delegate larger main areas for the individual members to focus on.

Followingly, better planned and followed deadlines, could have contributed to an improved project structure. As mentioned in the sprint summaries, at some point we had to keep extending our sprints, because the deadlines were usually too short, or the tasks were too time consuming to meet the deadline. To fix this we should have set the sprints to a longer duration. A typical sprint in the scrum model is around two to four weeks, and most of our sprints consisted of one to two-week sprint periods. This led to the team having to keep extending the ongoing sprints and made the work messy and unstructured at times.

In conclusion, critiques to our project sprints are clearly something that have enveloped from lack of experience using the framework. This has been a great learning experience that the team will bring into future projects.

10.3.4 Automated testing

Learning more about automated testing and how to implement it earlier on, could potentially have made the application development process more efficient. As mentioned earlier, the project development had a lack of unit testing, which could have proven to be helpful if implemented. For instance, to catch bugs we would not have been able to identify as fast or easily otherwise. Writing unit tests, in which are easily automated, is a key part of continuous integration software development. Having more knowledge of this earlier on in the project, could have contributed to making better quality code. In addition, automating time-consuming manual processes, such as using scripts for testing and pushing code to Git, could have contributed to time savings when we produced the code.

10.3.5 User testing and usability testing

Lack of user testing early on in software development can lead to large setbacks in producing code. This is an element that the project could potentially have benefitted a lot from, if the team had prioritized this earlier. A main point in usability testing, is to test as early as possible and preferably before implementing any code. This is because it is much easier to change the design before implementing code, than it is to implement changes to the design through changing the code. This project should have followed the modern software development process more strictly, and for instance, focused more on user-testing our wireframes to verify if the design is in fact user-friendly for a wide audience. In our case, we ended up performing most of the user testing at the end stages of the product development, which could have led to some potentially costly changes.

10.4 Future Work and Development

Future work might include implementing TLS for Redis, the webserver and the backend to secure the requests. This will also help to prevent MITM (Man-In-The-Middle), and traffic snooping attacks. Another task is implementing the NTNU data sources. The screenshot function should also be implemented differently as it is now a possible attack vector. Thus, a vulnerability of the software due to it simply downloading and viewing the URLs in a local headless chrome instance. Another potential improvement would be to use a proper database solution for the user authentication and the allow list. A final potential improvement would be the implementation of more extensive automatic unit testing. As it now stands, the sub-functionality of the application is only automatically tested through the output from the API endpoints, and manual testing. The implementation of automatic unit testing would increase certainty in the consistency of various sub-functionality in the application

10.5 Evaluation

10.5.1 Organization

Group work was organized in work sessions where the team met digitally and worked on the project collaboratively. The work for each week was planned on Monday and reviewed again the following Monday through short stand-up meetings with our supervisor. This way we had a good structure and plan for the work each week, which helped with assessment of progress. In the later stages of the project, we had physical meetings and daily work sessions at NTNU. Meeting physically ended up being a good motivator for the team, and increased productivity drastically. It also helped in making the work more organized.

10.5.2 Work distribution

Progress on the project was mostly consistent throughout the work period, although there were some periods where work with other deliverables was prioritized. Throughout the project, all team members were able to maintain their assigned role. In the end stages of the project phase, workload also picked up a bit as the project and report had to be finalized.

10.5.3 Project as a form of work

Project work is a good way to organize work. As with project work there is a defined task and result which makes it simple to assess if the work is successful, and of if it is of good quality. As you can compare the result to the task, including scope and requirements. Having a clear end goal is also good inspiration for the team members as it is possible to visualize the end result and therefore get inspired to work towards a common goal.

10.6 Conclusion

Threat Total is a full stack web application for displaying threat intelligence information in a high-level language. The frontend has a user-friendly interface, using NTNU branding colours and logos. The application's target group is a wide audience of NTNU staff and students. The backend is a robust Golang application with caching in Redis, for reducing the load on our third-party data providers as well as the request time for our users. Threading is also used to reduce the processing time for requests where appropriate. Due to integration with Feide authentication, logging in and out is as simple as with any of the NTNU services, which makes using Threat Total a seamless experience for members of NTNU staff and students.

The development framework the team chose to use was structured using a combination of the agile frameworks, Scrum and Kanban. The combination, "scrumban", utilizes sprints and distribution of tasks to aid teamwork while not relying on any hard deadlines.

We used a variety of tests, including both real user scenarios, static code analysis and API tests among other things. By utilizing different testing methods, we were able to both receive feedback on the usability of the application, as well as more technical answers when it came to discovering bugs and errors in the code.

Overall, the Threat Total project has been a great learning experience, impacting positively in both teamwork organizing and programming knowledge.

References

- [1] Enginss. (2014, November 3rd). *The 6 Principles Of Design, a la Donald Norman*.
<https://www.enginss.io/insights/6-principles-design-la-donald-norman>
- [2] GitLab. *About GitLab*. <https://about.gitlab.com/company/>
- [3] SonarQube. *About SonarQube*. <https://www.sonarqube.org/about/>
- [4] UTOR. (2021, March 31st). *Advantages of Usability Testing and Some Drawbacks You Should Know*. <https://u-tor.com/topic/advantages-of-usability-testing>
- [5] Microsoft. (2022, February 18th). *Authentication vs. authorization*.
<https://docs.microsoft.com/en-us/azure/active-directory/develop/authentication-vs-authorization>
- [6] Amazon Web Services. *Caching Overview*. <https://aws.amazon.com/caching/>
- [7] W3Schools. *CSS Media Queries - Examples*.
https://www.w3schools.com/css/css3_mediaqueries_ex.asp
- [8] MDN Web Docs. (2022, May 2nd). *CSS: Cascading Style Sheets*.
<https://developer.mozilla.org/en-US/docs/Web/CSS>
- [9] Feide. *Feide*. https://docs.Feide.no/general/Feide_overview.html
- [10] Gin. (2022, May 14th). *Gin Web Framework*. <https://github.com/gin-gonic/gin>
- [11] Johnston, J. (2019, April 23rd). *How to build a web app: A beginner's guide (2021)*.
<https://budibase.com/blog/how-to-make-a-web-app/>
- [12] Roose, J. *How to Conduct Usability Testing in Six Steps*.
<https://www.toptal.com/designers/ux-consultants/how-to-conduct-usability-testing-in-6-steps>
- [13] Lane, C., & Krüger, N. (2021, December). *How to Write a Software Requirements Specification (SRS Document)*. <https://www.perforce.com/blog/alm/how-write-software-requirements-specification-srs-document>
- [14] MDN Web Docs. (2022, May 2nd). *HTML: HyperText Markup Language*.
<https://developer.mozilla.org/en-US/docs/Web/HTML>
- [15] DELL Technologies. (2021, February 21st). *Hva er minne (RAM)?*
<https://www.dell.com/support/kbdoc/no-no/000148441/hva-er-minne-ram>

- [16] Trend Micro. *Indicators of compromise*. <https://www.trendmicro.com/vinfo/us/security/definition/indicators-of-compromise>
- [17] Interaction Design Foundation. *Interaction Design*. <https://www.interaction-design.org/literature/topics/interaction-design>
- [18] Microsoft. (2022, April 16th). *OAuth 2.0 and OpenID Connect (OIDC) in the Microsoft identity platform*. <https://docs.microsoft.com/en-us/azure/active-directory/develop/active-directory-v2-protocols>
- [19] Nielsen, J. (2003, April 13th). *Paper Prototyping: Getting User Data Before You Code*. <https://www.nngroup.com/articles/paper-prototyping/>
- [20] Redis. *Redis*. <https://redis.io/>
- [21] Friedman, V. (2018, August 11th). *Responsive Web Design - What It Is And How To Use It*. <https://www.smashingmagazine.com/2011/01/guidelines-for-responsive-web-design/>
- [22] usability.gov. *Use Cases*. <https://www.usability.gov/how-to-and-tools/methods/use-cases.html>
- [23] Gorbachenko, P. (2021, April 9th). *What are Functional and Non-Functional Requirements and How to Document These*. <https://enkonix.com/blog/functional-requirements-vs-non-functional/>
- [24] Software Engineering. (2018, September). *What are the differences between server-side and client-side programming?* <https://softwareengineering.stackexchange.com/questions/171203/what-are-the-differences-between-server-side-and-client-side-programming>
- [25] Red Hat. (2020, May 8th). *What is a REST API?* <https://www.redhat.com/en/topics/api/what-is-a-rest-api>
- [26] GeeksforGeeks. (2022, January 18th). *What is full stack development?* <https://www.geeksforgeeks.org/what-is-full-stack-development/>
- [27] W3Schools. *What is Full Stack?* https://www.w3schools.com/whatis/whatis_fullstack.asp
- [28] Radigan, D. *What is kanban?* <https://www.atlassian.com/agile/kanban>
- [29] OpenID. *What is OpenID Connect?* <https://openid.net/connect/>
- [30] Microsoft. (2021, May 14th). *What is Scrum*. <https://docs.microsoft.com/en-us/devops/plan/what-is-scrum>

Postlude

- [31] Bellairs, R. (2020, February 10th). *What Is Static Analysis? Static Code Analysis Overview*. <https://www.perforce.com/blog/sca/what-static-analysis>
- [32] VirusTotal. *How it works*. <https://support.virustotal.com/hc/en-us/articles/115002126889-How-it-works>
- [33] Gin Web Framework. (2022, April 29th). *Single file*. <https://gin-gonic.com/docs/examples/upload-file/single-file/>
- [34] ganlvtech. (2018, December 11th). *file.FileName should not be trusted. There should be a sanitize function, or give a warning in docs*. <https://github.com/gin-gonic/gin/issues/1693>
- [35] Golang. *Package multipart*. <http://www.golang.pw/pkg/mime/multipart/#Writer.CreatePart>
- [36] Golang By Example. (2021, January 19th). *HTTP- Understanding multipart/form-data content-type*. <https://golangbyexample.com/multipart-form-data-content-type-golang/>
- [37] urlscan.io. *urlscan.io A sandbox for the web*. <https://urlscan.io/about/>
- [38] Cisco. *What is CI/CD?* <https://www.cisco.com/c/en/us/solutions/data-center/data-center-networking/what-is-ci-cd.html>
- [39] Martí, D., & Shaw, K. (2021, April 27th). *About chromedp*. <https://github.com/chromedp/chromedp/blob/master/README.md>
- [40] Check Point. *What is Remote Code Execution (RCE)?* <https://www.checkpoint.com/cyberhub/cyber-security/what-is-remote-code-execution-rce/>

Appendices:

Appendix 1: User Testing Spreadsheet

Appendix 2: Original Gantt Scheme

Appendix 3: Updated Gantt Scheme

Appendix 4: Project Plan

Appendix 5: Collaboration Agreement

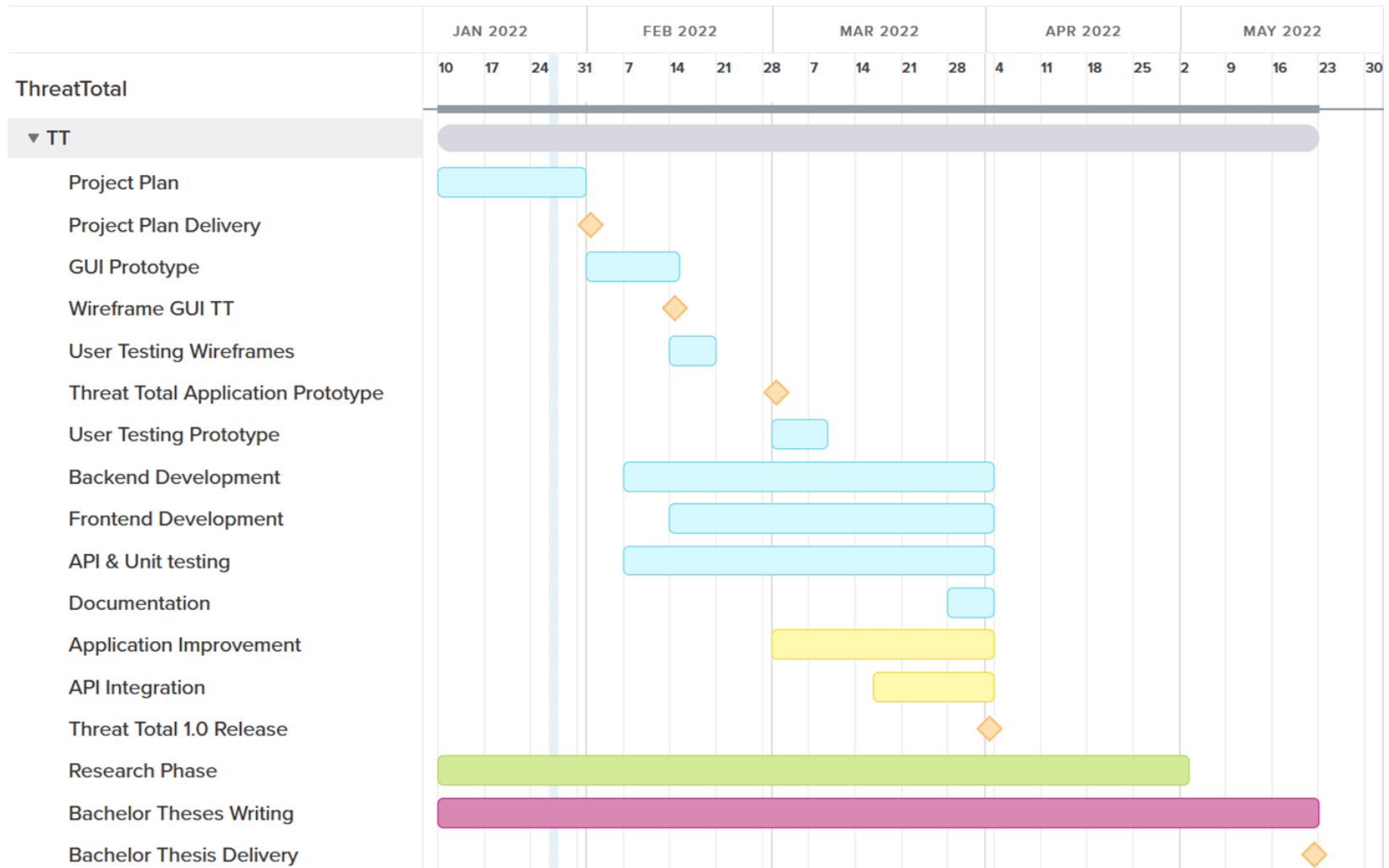
Appendix 6: Time Log

Appendix 7: Meeting Log

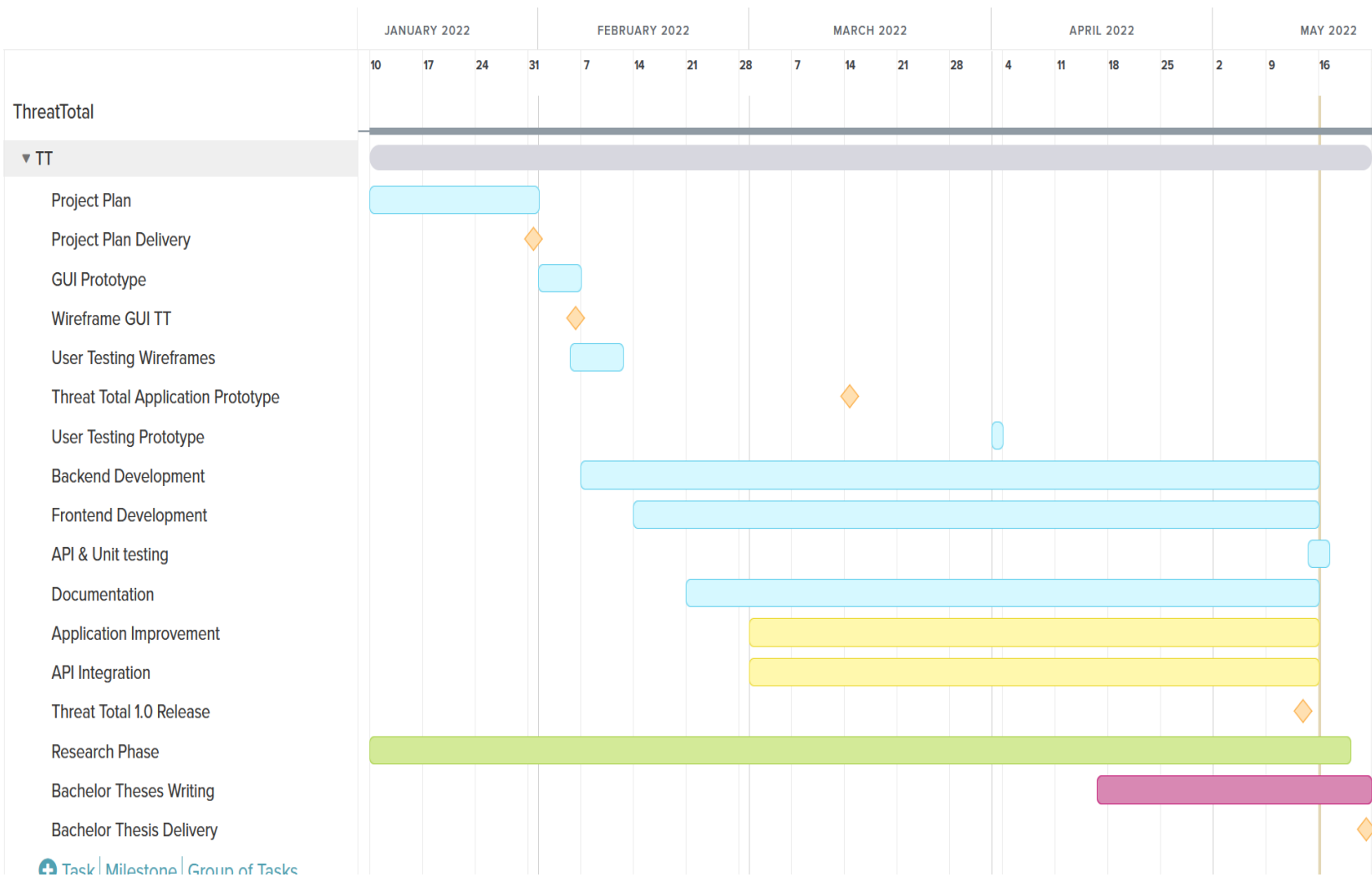
Appendix 8: Wireframes

Person	Age	Technical knowledge (Bad - Basic - Intermediate - Advanced)	Log in	Investigate URL	Investigate file	Log out	Change language	If some of the tasks were unclear or not finished, comment on this (Mark sections on the left green or red based on if task was successfully done)
1	22	Intermediate						
2	25	Intermediate						
3	26	Advanced						Layout of website does not scale good to wide screen

USER TESTING SPREADSHEET



ORIGINAL GANTT SCHEME MADE IN APP.TEAMGANTT.COM



UPDATED GANTT SCHEME SHOWING THE ACTUAL TIMELINE MADE IN APP.TEAMGANTT.COM



NTNU

Kunnskap for en bedre verden

Project plan:

Group 116

Authors:

Johannes Madsen Barstad

Odin Korsfur Henriksen

Jonas Kjærandsen

Peder Andreas Stuen

January, 2022

Contents

1 Goals and framework.....	3
1.1 Background.....	3
1.2 Project goals.....	3
1.3 Framework.....	4
2 Scope.....	5
2.1 Scope.....	5
2.2 Task description.....	7
3 Project organization.....	8
3.1 Roles and responsibilities.....	8
3.2 Routines and rules.....	9
4 Planning and reporting.....	10
4.1 Main division of project.....	10
4.2 Plan for status meetings and decision points.....	10
5 Risk analysis and organization of quality assurance.....	11
5.1 Risk analysis at project level.....	11
5.2 Risk reduction measures.....	13
5.3 Resources.....	15
5.4 Plans for inspections and testing.....	15
6 Plan for implementation.....	16
6.1 Main activities.....	16
6.2 Milestones.....	16
6.3 Gantt chart.....	17

Tables and Figures

Table 1: Weekly meeting schedule.....	9
Table 2: Risk analysis table.....	11
Table 3: Incident descriptions.....	12
Table 4: Risk reduction measures.....	14
Table 5: Risk analysis after mitigations.....	14
Figure 1: Gantt Chart.....	17

1 Goals and framework

1.1 Background

The background for this project is a task that is made by the Security Operation Centre (SOC) at NTNU in Gjøvik. They need a web-based solution which can be used by all authorized NTNU users to check if a file, a suspicious URL, or a domain has any malicious intents.

After careful consideration of the different Bachelor thesis tasks, we could choose from, all the team members agreed upon this task. The reason behind this is our interest with the field, as well as the relevancy of it.

The team has had a meeting with one of our contact persons at NTNU's SOC to get a better understanding of the task and some key points such as programming language, framework for development and collaboration tools. The team have had a meeting with our supervisor as well. During this meeting we discussed matters such as weekly scheduled meetings, the scope of the Bachelor thesis and what kind of framework development to use.

1.2 Project goals

Our goal is to create a functional product, which includes an application with an accessible API, Norwegian and English support, documentation, and automated tests. The team has ambitions on getting a **B** for the Bachelor thesis, but this will require a good amount from all members, both when it comes to work amount, as well as work precision. We think however if these criteria are met, the grade **B** should be achievable.

An overview of the time usage, resource usage and milestones/ sub-goals can be found under [chapter 6](#) which contains a Gantt chart for the entirety of the project period.

1.3 Framework

The framework we have decided to use for this bachelor project is scrum. The scrum methodology is divided in four meetings, sprint planning, stand-up, review, and retrospective.

- Sprint planning:

In the Sprint planning it is discussed what the next sprint is going to achieve and who is doing what. A sprint is a task.

- Stand-up meetings:

Stand-up is a small meeting where we give a report on how we are doing, what we have done, what we are going to do, and if we are going to make the deadline for the sprint.

- Review:

During the review, the results are delivered, and we receive feedback on the sprint.

- Retrospective:

During the retrospective meeting we are looking back on the process of the sprint and look at what can be improved for the next sprint.

A sprint can be defined as a small task / sub-project of our Threat-Total project which is each a part of the complete Threat-Total product we will be developing. To help with tracking of sprints and assignments we will be using Atlassian Jira.

2 Scope

2.1 Scope

The scope of this bachelor thesis is to develop the application “Threat Total” for NTNU’s SOC department, which will be used to check URLs, Domains, and file-hashes against several databases of IOCs both public and private.

Our scope is to develop a user-friendly application which will be used by NTNU students and employees to lookup if a URL, domain, or file hash is malicious. We will develop the website to support both English and Norwegian. The application will use publicly available reputation sources and hash databases, as well as NTNU’s private reputation database to check if the domain, URL, or file hash has a reputation of being malicious. The application will be utilizing a REST API to communicate and access data from NTNU SOC. The website application will also be utilizing the FEIDE portal login system and will be able to retrieve contact information. This will be used whenever a hash, domain or URL is unknown for NTNU, it will be possible to create a case for NTNU SOC. Security analysts can then further check it out, analyze the case, and proceed with further communication with the user.

We will develop the backend in Golang, and the front-end will be developed using JavaScript.

To increase the efficiency of our application it will also be important to look at implement support for caching to store the most recently searched domains, URLs, and file hashes.

Functionality:

- Functionality to search for a domain, URL, or file hash.
- Login functionality using FEIDE.
- Retrieval of contact information from FEIDE.
- The possibility to create an event for analysis to NTNU SOC.
- Utilize a REST API to retrieve information from NTNU SOC.
- The application should be able to show disposition for indicators (Domain, IP, etc.)
- Gather reputation data from both public and private (NTNU's) reputation sources.
- Submit file, URL, domain.
- Block/allow list functionality for indicators.
 - Filter for what can and cannot be displayed to the user.
- Submitted files should first be scanned automatically, and if no result is given the file may be submitted for manual analysis. If the file previously has been analyzed, the previous analysis should be submitted to the frontend user.
- [OPTIONAL] The application should be able to collect and display a screenshot of the domain or URL that is sent in for analysis.

Content:

- A user-friendly easy to understand GUI.
- GUI supports both Norwegian and English.
- Text to display if this domain, URL, or file hash is known to be malicious.
- Display what reputation the domain or URL has.
- Display what file the file hash corresponds to.
- Must use NTNU's color and branding.

What is not included in our scope:

- Discussed with the client that this will have to be determined as we are starting to work. It is too early in the project to assign what we won't be handling.

2.2 Task description

The task is to develop a full-stack application “NTNU Threat Total”. The application is a self-help solution in the form of a web portal where registered NTNU users can check if a link or a file is dangerous, get feedback from the operation center about the risk of the file or the site and finally view the status of eventual measures taken. If the file or page is unknown there will be an option to send it to analysis and to create a case. In addition to NTNU sources the application should retrieve information from open sources on the internet. The application will be developed with automated tests and an accessible API to make integration with other programs possible.

3 Project organization

3.1 Roles and responsibilities

Information about client and supervisor:

- Client: NTNU SOC
- Contact person: Christoffer Vargtass Hallstensen, Group leader SOC
christoffer.hallstensen@ntnu.no
- Contact person: Frank Wikstrøm, Security analyst
frank.wikstrom@ntnu.no
- Supervisor: Espen Torseth, Senior advisor
espen.torseth@ntnu.no

The team members of the project:

- Johannes Madsen Barstad, Project Leader
johanmba@stud.ntnu.no
- Odin Korsfur Henriksen
odinkh@stud.ntnu.no
- Jonas Kjærandsen
jonakj@stud.ntnu.no
- Peder Andreas Stuen
pederas@stud.ntnu.no

Our clients / stakeholders are Christoffer Vargtass Hallstensen and Frank Wikstrøm. Our supervisor is Espen Torseth. In order to have a final decision if conflict arise, we will have a project leader. The project leader we have chosen is Johannes Madsen Barstad. In email and similar forms of communication our main contact will therefore be Johannes Madsen Barstad.

3.2 Routines and rules

- Be professional.
- Be a team player.
- Give notice in advance when you are unable to attend a meeting.
- It is expected of each of the team members to work an average of 25-30 hours a week.
- If a member works less a day or a week it is expected for the member to compensate for the missed work individually.

To organize the work throughout the week we will have daily scheduled meetings:

Meeting schedule:						
Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
10:30	12:30	12:30	11:00	14:30		

Table 1: Weekly meeting schedule

4 Planning and reporting

4.1 Main division of project

The project will be divided into two main parts. First part being building the application, and secondly documenting the work through the main project report. These two parts will be progressed in parallel throughout the project period.

4.2 Plan for status meetings and decision points

The team thinks that weekly status meetings with supervisor is sufficient. By doing this we can maintain a close communication and keep the supervisor updated on our progress. The weekly status meeting will be held Mondays, from 12:00pm to 12:30pm. Minutes of meeting should always be written and added as an attachment in the bachelor thesis report. We will also have meetings with the clients where it is relevant, such as when a prototype is completed, or another milestone is reached.

5 Risk analysis and organization of quality assurance

5.1 Risk analysis at project level

Risk Analysis table:				
Impact				
Likelihood		Low (1)	Medium (2)	High (3)
	High (3)			
	Medium (2)		No. 2	
	Low (1)		No. 6	No. 1, No. 3, No. 4, No. 5, No. 7

Table 2: Risk analysis table

Incident descriptions:			
Incidents		Risk score	Reasoning
No. 1	A team member gets sick and unable to work.	3	Somewhat elevated likelihood during the current pandemic. Though can lead to backlog as sickness will slow down project work
No. 2	A team member is unable to work due to personal matters, such as going to their job.	4	3 out of 4 students in the group has a job as SOC or student assistant. Therefore, if not properly planned, this can lead to a lot of backlogged hours of work. Potentially leads to exhausting all-nighters to catch up lost work.
No. 3	Due time for delivery is not met.	3	Leads to project failure.
No. 4	Confidentiality is violated.	3	Could lead to potential project fail
No. 5	Unexpected specification adjustments from client	3	Not likely that the SOC changes the specifications, though if it were to happen it would be critical, as this would lead to a lot of do-over work

No. 6	Conflict between team members.	2	Low likelihood as we have worked as a team over the whole course of the bachelor, though there is always a chance of conflict. Would create some overhead as we spend time on the problem instead of work but should be manageable since it's expected to happen rarely.
No. 7	Applications used for work becomes deprecated.	3	Very low likelihood for modern tools, though the possibility still exists. If a tool we're using were to be deprecated, the group would have to re-organize work and create a lot of work-overhead.

Table 3: Incident descriptions

5.2 Risk reduction measures

Incidents:			
Incidents		Mitigation	Rest risk
No. 1	A team member gets sick and unable to work.	Hard to mitigate this incident as it is often out of our control, good planning, and collection of notes to not make us fully dependent on a missing team member would limit the damage this has. Good notes would reduce the impact of this incident.	2
No. 2	A team member is unable to work due to personal matters, such as going to their job.	Hard to mitigate this incident as it is often out of our control, but we would try to plan around this to the best of our ability to not decrease our product-ability if one team member is missing. Possible mitigation here includes keeping our notes collected so we will not be fully dependent on a missing team member.	2
No. 3	Due time for delivery is not met.	Weekly status meetings and organized sprints to make sure work is completed on time and delivery dates are met. This will reduce the likelihood of the incident happening, though impact stays the same, which is project failure.	3
No. 4	Confidentiality is violated.	By creating a policy that clearly states what should be shared with whom, reduces the risk of sharing sensitive information to unintended parties. Maintaining routines related to the policy will further reduce the likelihood of this risk. Impact stays the same.	3

No. 5	Unexpected specification adjustments from client	By keeping an open channel for dialogue throughout the project and having meetings where necessary we hope to mitigate this risk. Reduces likelihood and impact, as a good dialogue would give better odds in finding potential adjustments	2
No. 6	Conflict between team members.	Open dialogue can mitigate the risk of escalation of conflicts, allowing members to view different point of views. Also having the team leader to have the final say, if discussion leads to decisions. This would reduce the impact of the risk, as the final word will end discussion without further escalation.	1
No. 7	Applications used for work becomes deprecated.	Evaluation of the tool can mitigate the risk of it being deprecated while the project is ongoing. The impact of this incident stays the same.	3

Table 4: Risk reduction measures

Risk Analysis table after mitigations:				
Impact				
Likelihood		Low (1)	Medium (2)	High (3)
	High (3)			
	Medium (2)	No. 2		
	Low (1)	No. 6	No. 1, No. 5,	No. 3, No. 4, No. 7

Table 5: Risk analysis after mitigations

5.3 Resources

README.md or GitLab wiki with detailed explanation of application usage, setup, and deployment. The code will be documented and commented during the development period. In addition to documentation and comments we will be writing automated tests for the software to verify functionality where it is relevant.

Tools:

- Microsoft Teams, email, and Discord for communication.
- Jira for support with scrum and sprints.
- OpenStack for hosting VM's for testing the web application and belonging functions.
- GitLab and git for code collaboration and version control.

Resource specifications:

In NTNU's virtualization platform SkyHigh we have been allotted the following resources:

- 32 VCPU cores
- 64 GB RAM
- 300 GB Storage

These are resources we will use to setup a common testing and hosting environment for Threat Total.

5.4 Plans for inspections and testing

We will develop unit tests for the software so that you will be able to run automated tests for different functions in the software. We will also perform user tests, if possible, to gauge the usability of the frontend.

Quality in the report will be assured using scrum sprints in Jira. This is something we will continuously work on, as well as to the end of the project period.

6 Plan for implementation

6.1 Main activities

We have divided the project into the following main activities:

- **Project plan:** Writing this document, a precursor focused on planning the main project work.
- **GUI prototype:** A frontend prototype for the website, used for receiving feedback on and improving the frontend.
- **User Testing:** Two different activities for receiving feedback on the prototypes.
- **Backend Development:** Developing the backend of the software, API integration, data handling and such.
- **Frontend Development:** Developing the frontend of the software, the website design.
- **API & Unit testing:** Creating unit tests for the software and testing the API endpoints.
- **Documentation:** Writing documentation to the software, including how to use it, the API endpoints and how to deploy it.
- **Application improvement:** Using feedback received from the clients and possibly user testing to improve the application.
- **API Integration:** Integrating with external APIs as data sources.
- **Research Phase:** Researching relevant technology, methodology and such for the project.
- **Bachelor Thesis writing:** Writing the bachelor thesis, the final report and relevant attachments.

6.2 Milestones

The milestones we have set for the project include:

- **Project plan delivery:** The delivery of this document, this signals the start of the main phase of the project where we work on the software we are developing.
- **Wireframe GUI:** Where we have finished a prototype of the frontend of the web application which will be, if possible, tested on users, to receive feedback and start on improvements.
- **Threat Total Application Prototype:** A prototype of the software which will allow us to test the software, receive feedback and plan changes and improvements.
- **Threat Total 1.0 Release:** The final 1.0 release of the software, signaling a working product.
- **Bachelor Thesis Delivery:** A finalized bachelor thesis, signaling the end of the project.

6.3 Gantt chart

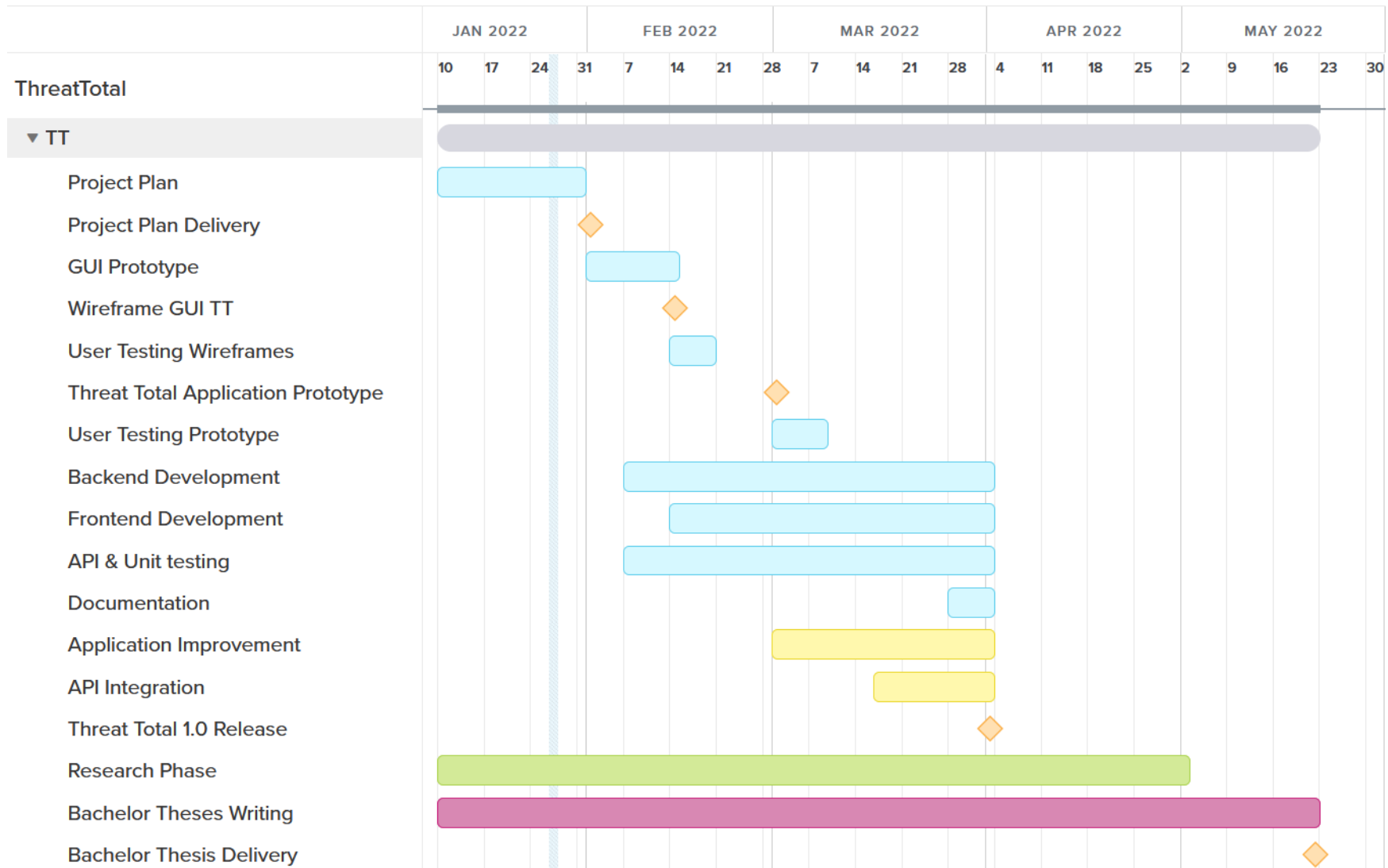


Figure 1: Gantt Chart

Fastsatt av prorektor for utdanning 10.12.2020

STANDARDAVTALE

om utføring av studentoppgave i samarbeid med ekstern virksomhet

Avtalen er ufravikelig for studentoppgaver (heretter oppgave) ved NTNU som utføres i samarbeid med ekstern virksomhet.

Forklaring av begrep

Opphavsrett

Er den rett som den som skaper et åndsverk har til å fremstille eksemplar av åndsverket og gjøre det tilgjengelig for allmennheten. Et åndsverk kan være et litterært, vitenskapelig eller kunstnerisk verk. En studentoppgave vil være et åndsverk.

Eiendomsrett til resultater

Betyr at den som eier resultatene bestemmer over disse. Utgangspunktet er at studenten eier resultatene fra sitt studentarbeid. Studenten kan også overføre eiendomsretten til den eksterne virksomheten.

Bruksrett til resultater

Den som eier resultatene kan gi andre en rett til å bruke resultatene, f.eks. at studenten gir NTNU og den eksterne virksomheten rett til å bruke resultatene fra studentoppgaven i deres virksomhet.

Prosjektbakgrunn

Det partene i avtalen har med seg inn i prosjektet, dvs. som vedkommende eier eller har rettigheter til fra før og som brukes i det videre arbeidet med studentoppgaven. Dette kan også være materiale som tredjepersoner (som ikke er part i avtalen) har rettigheter til.

Utsatt offentliggjøring

Betyr at oppgaven ikke blir tilgjengelig for allmennheten før etter en viss tid, f.eks. før etter tre år. Da vil det kun være veileder ved NTNU, sensorene og den eksterne virksomheten som har tilgang til studentarbeidet de tre første årene etter at studentarbeidet er innlevert.

1. Avtaleparter

Norges teknisk-naturvitenskapelige universitet (NTNU) Institutt: Institutt for Informasjonssikkerhet og Kommunikasjonsteknologi
Veileder ved NTNU: Espen Torseth, Seniorrådgiver Epost: espen.torseth@ntnu.no Tel: 91690629
Seksjon for Digital sikkerhet, NTNU IT Christoffer Vargtass Hallstensen, Faggruppeteleder SOC Epost: Christoffer.hallstensen@ntnu.no Tel: 61135145
Student: Odin Korsfur Henriksen Fødselsdato: 02.05.1999
Student: Jonas Kjærandsen Fødselsdato: 28.06.2000
Student: Johannes Madsen Barstad Fødselsdato: 23.05.2000
Student: Peder Andreas Stuen Fødselsdato: 26.05.2000

Partene har ansvar for å klarere eventuelle immaterielle rettigheter som studenten, NTNU, den eksterne eller tredjeperson (som ikke er part i avtalen) har til prosjektbakgrunn før bruk i forbindelse med utførelse av oppgaven. Eierskap til prosjektbakgrunn skal fremgå av eget vedlegg til avtalen der dette kan ha betydning for utførelse av oppgaven.

2. Utførelse av oppgave

Studenten skal utføre: (sett kryss)

Masteroppgave	
Bacheloroppgave	X
Prosjektoppgave	
Annen oppgave	

Startdato: 14.01.2022
Sluttdato: 20.05.2022

Oppgavens arbeidstitel er: NTNU Threat Total

Ansvarlig veileder ved NTNU har det overordnede faglige ansvaret for utforming og godkjenning av prosjektbeskrivelse og studentens læring.

3. Ekstern virksomhet sine plikter

Ekstern virksomhet skal stille med en kontaktperson som har nødvendig faglig kompetanse til å gi studenten tilstrekkelig veiledning i samarbeid med veileder ved NTNU. Ekstern kontaktperson fremgår i punkt 1.

Formålet med oppgaven er studentarbeid. Oppgaven utføres som ledd i studiet. Studenten skal ikke motta lønn eller lignende godtgjørelse fra den eksterne for studentarbeidet. Utgifter knyttet til gjennomføring av oppgaven skal dekkes av den eksterne. Aktuelle utgifter kan for eksempel være reiser, materialer for bygging av prototyp, innkjøp av prøver, tester på lab, kjemikalier. Studenten skal klarere dekning av utgifter med ekstern virksomhet på forhånd.

Ekstern virksomhet skal dekke følgende utgifter til utførelse av oppgaven:
--

Dekning av utgifter til annet enn det som er oppført her avgjøres av den eksterne underveis i arbeidet.

4. Studentens rettigheter

Studenten har opphavsrett til oppgaven¹. Alle resultater av oppgaven, skapt av studenten alene gjennom arbeidet med oppgaven, eies av studenten med de begrensninger som følger av punkt 5, 6 og 7 nedenfor. Eiendomsretten til resultatene overføres til ekstern virksomhet hvis punkt 5 b er avkrysset eller for tilfelle som i punkt 6 (overføring ved patenterbare oppfinnelser).

I henhold til lov om opphavsrett til åndsverk beholder alltid studenten de ideelle rettigheter til eget åndsverk, dvs. retten til navngivelse og vern mot krenkende bruk.

Studenten har rett til å inngå egen avtale med NTNU om publisering av sin oppgave i NTNUs institusjonelle arkiv på Internett (NTNU Open). Studenten har også rett til å publisere oppgaven eller deler av den i andre sammenhenger dersom det ikke i denne avtalen er avtalt begrensninger i adgangen til å publisere, jf. punkt 8.

5. Den eksterne virksomheten sine rettigheter

Der oppgaven bygger på, eller videreutvikler materiale og/eller metoder (prosjektbakgrunn) som eies av den eksterne, eies prosjektbakgrunnen fortsatt av den eksterne. Hvis studenten

¹ Jf. Lov om opphavsrett til åndsverk mv. av 15.06.2018 § 1

skal utnytte resultater som inkluderer den eksterne sin prosjektbakgrunn, forutsetter dette at det er inngått egen avtale om dette mellom studenten og den eksterne virksomheten.

Alternativ a) (sett kryss) Hovedregel

<input type="checkbox"/>	Ekstern virksomhet skal ha bruksrett til resultatene av oppgaven
--------------------------	--

Dette innebærer at ekstern virksomhet skal ha rett til å benytte resultatene av oppgaven i egen virksomhet. Retten er ikke-eksklusiv.

Alternativ b) (sett kryss) Unntak

<input checked="" type="checkbox"/>	Ekstern virksomhet skal ha eiendomsretten til resultatene av oppgaven og studentens bidrag i ekstern virksomhet sitt prosjekt
-------------------------------------	---

Begrunnelse: Seksjon for Digital sikkerhet beholder eiendomsretten til resultatet for å sikre at ressurser betalt av offentlige midler skal gå til fellesskapets beste etter NTNUs strategi om «Kunnskap for en bedre verden». Seksjon for Digital sikkerhet forplikter seg til å lisensiere kode og rapport som åpen kildekode/creative commons slik at studentene kan ta med seg arbeidet nedlagt i prosjektet videre etter studier, men samtidig ivaretar at fremtidige studenter og andre kan bygge videre på arbeidet.

6. Godtgjøring ved patenterbare oppfinnelser

Dersom studenten i forbindelse med utførelsen av oppgaven har nådd frem til en patenterbar oppfinnelse, enten alene eller sammen med andre, kan den eksterne kreve retten til oppfinnelsen overført til seg. Dette forutsetter at utnyttelsen av oppfinnelsen faller inn under den eksterne sitt virksomhetsområde. I så fall har studenten krav på rimelig godtgjøring. Godtgjøringen skal fastsettes i samsvar med arbeidstakeroppfinnelsesloven § 7. Fristbestemmelsene i § 7 gis tilsvarende anvendelse.

7. NTNU sine rettigheter

De innleverte filer av oppgaven med vedlegg, som er nødvendig for sensur og arkivering ved NTNU, tilhører NTNU. NTNU får en vederlagsfri bruksrett til resultatene av oppgaven, inkludert vedlegg til denne, og kan benytte dette til undervisnings- og forskningsformål med de eventuelle begrensninger som fremgår i punkt 8.

8. Utsatt offentliggjøring

Hovedregelen er at studentoppgaver skal være offentlige.

Sett kryss

X	Oppgaven skal være offentlig
---	------------------------------

I særlige tilfeller kan partene bli enige om at hele eller deler av oppgaven skal være undergitt utsatt offentliggjøring i maksimalt tre år. Hvis oppgaven unntas fra offentliggjøring, vil den kun være tilgjengelig for student, ekstern virksomhet og veileder i denne perioden. Sensurkomiteen vil ha tilgang til oppgaven i forbindelse med sensur. Student, veileder og sensorer har taushetsplikt om innhold som er unntatt offentliggjøring.

Oppgaven skal være underlagt utsatt offentliggjøring i (sett kryss hvis dette er aktuelt):

Sett kryss

Sett dato

<input type="checkbox"/>	ett år	
<input type="checkbox"/>	to år	
<input type="checkbox"/>	tre år	

Behovet for utsatt offentliggjøring er begrunnet ut fra følgende:

Dersom partene, etter at oppgaven er ferdig, blir enig om at det ikke er behov for utsatt offentliggjøring, kan dette endres. I så fall skal dette avtales skriftlig.

Vedlegg til oppgaven kan unntas ut over tre år etter forespørsel fra ekstern virksomhet. NTNU (ved instituttet) og student skal godta dette hvis den eksterne har saklig grunn for å be om at et eller flere vedlegg unntas. Ekstern virksomhet må sende forespørsel før oppgaven leveres.

De delene av oppgaven som ikke er undergitt utsatt offentliggjøring, kan publiseres i NTNUs institusjonelle arkiv, jf. punkt 4, siste avsnitt. Selv om oppgaven er undergitt utsatt offentliggjøring, skal ekstern virksomhet legge til rette for at studenten kan benytte hele eller deler av oppgaven i forbindelse med jobbsøknader samt videreføring i et master- eller doktorgradsarbeid.

9. Generelt

Denne avtalen skal ha gyldighet foran andre avtaler som er eller blir opprettet mellom to av partene som er nevnt ovenfor. Dersom student og ekstern virksomhet skal inngå avtale om konfidensialitet om det som studenten får kjennskap til i eller gjennom den eksterne virksomheten, kan NTNUs standardmal for konfidensialitetsavtale benyttes.

Den eksterne sin egen konfidensialitetsavtale, eventuell konfidensialitetsavtale den eksterne har inngått i samarbeidprosjekter, kan også brukes forutsatt at den ikke inneholder

punkter i motstrid med denne avtalen (om rettigheter, offentliggjøring mm). Dersom det likevel viser seg at det er motstrid, skal NTNUs standardavtale om utføring av studentoppdage gå foran. Eventuell avtale om konfidensialitet skal vedlegges denne avtalen.

Eventuell uenighet som følge av denne avtalen skal søkes løst ved forhandlinger. Hvis dette ikke fører frem, er partene enige om at tvisten avgjøres ved voldgift i henhold til norsk lov. Tvisten avgjøres av sorenskriveren ved Sør-Trøndelag tingrett eller den han/hun oppnevner.

Denne avtale er signert i fire eksemplarer hvor partene skal ha hvert sitt eksemplar. Avtalen er gyldig når den er underskrevet av NTNU v/instituttleder.

Signaturer:

Instituttleder:	<i>Pi vegne av instituttleder</i>
Dato: <i>8/2-22</i>	<i>Katrin. Moe</i> <i>Nils Kallestad</i>
Veileder ved NTNU:	<i>Jon Eivind</i>
Dato: <i>1/2-2022</i>	
Seksjon for Digital sikkerhet, NTNU IT:	<i>C. Vindheim</i>
Dato: 01.02.2022	
Student: <i>Johannes Barbed</i>	NORGES TEKNISK-NATURVITENSKAPELIGE UNIVERSITET
Dato: 01.02.2022	
Student: <i>Peder A. Stuen</i>	
Dato: 01.02.2022	
Student: <i>Jonas Kjørandsen</i>	
Dato: 01.02.2022	
Student: <i>Odin k. Henriksen</i>	
Dato: 01.02.2022	

Time logs

Team time log

Date:	Hours:	Type:	Description:
14/01/2022	2	Team	Meeting with the client, and work after.
17/01/2022	2.5	Team	Starting work with the project plan.
18/01/2022	2	Team	Work on the project plan, scheduling future meetings.
19/01/2022	3.5	Team	Work on the project plan and the gantt chart.
20/01/2022	2	Team	Meeting and work on the project plan.
21/01/2022	2	Team	Meeting with supervisor.
24/01/2022	2.5	Team	Meeting with supervisor and project plan work.
25/01/2022	3.5	Team	Implemented scrum sprints with Jira, project plan work.
26/01/2022	1	Team	Gantt chart work.
28/01/2022	3	Team	Meeting with client and project work
31/01/2022	3	Team	Meeting with supervisor and project plan work. Wireframe work.
01/02/2022	4	Team	Setting up a basic web server. Looking into css framework, will most likely use tailwind. Looked into web framework for golang (gin). Signed documents.
02/02/2022	2	Team	Setting up our development environment for tailwind and prototyping.
03/02/2022	3	Team	Further prototyping with tailwind. Preparation for meeting with Frank regarding wireframes and frontend development. Ubuntu server set up in skyhigh.
04/02/2022	1	Team	Meeting with Frank regarding front-end development. Organization of sprint for next week.
07/02/2022	2	Team	Status meeting with supervisor. Improvement of project plan. Discussing ReactJS implementation vs og templates
08/02/2022	2	Team	Frontend and backend work
09/02/2022	2	Team	Frontend and backend work
14/02/2022	1	Team	
18/02/2022	1	Team	Discussing further work
21/02/2022	1	Team	Standup meeting and work planning

Time logs

03/03/2022	5	Team	URL search functionality, checking its validity with a regular expression. Translating meeting logs. Making a skeleton for main report.
04/03/2022	2	Team	TT development
07/03/2022	1	Team	Meeting with supervisor and TT development
16/03/2022	3	Team	Work on external api's
21/03/2022	1	Team	Monday meeting, API from NTNU has been confirmed done, though the access has not been set up yet.
01/04/2022	1	Team	Meeting with NTNU SOC regarding demo and API access
04/04/2022	1	Team	Meeting with supervisor and creation of new sprint
07/04/2022	1	Team	Status updates and how we're doing on the current sprint
23/04/2022	1	Team	Meeting with supervisor and division of tasks
28/04/2022	4	Team	Collaborative work on implementing OpenID Connect
29/04/2022	5	Team	Colaborative work on creating reverse proxy in golang, and file upload and handling
02/05/2022	7.5	Team	Setting up sonar qube for code quality assurance and security testing, implementing FEIDE authentication, finished file uploading for backend
03/05/2022	7.5	Team	Main report work, FEIDE and OIDC work, started with implementing logging to separate log file, file upload work
04/05/2022	4	Team	Main report work, FEIDE and OIDC work, representation of file analysis result work
05/05/2022	5	Team	Main report, FEIDE authentication, upload file and show response.
06/05/2022	5	Team	Main report work, tying up FEIDE authentication, implementin of logging functionality
07/05/2022	4	Team	Internal discussion and work with main report, dividing fileupload functions into separate file, securing fileupload endpoint

Time logs

08/05/2022	6	Team	Main report work, finishing up code and implementing TLS for redis and backend
09/05/2022	6	Team	Implementation of URL screenshots, implementation of simple filter functionality for URL's, main report work
10/05/2022	6	Team	Finishing URL screenshot, implementation of simple email system for case escalation, main report work
12/05/2022	4	Team	Main report work, minor bug fixes (log in with FEIDE, wait group for url-filter check, fetching of file hash information and displaying to user)
13/05/2022	6	Team	Main report work
14/05/2022	4	Team	Main report work
15/05/2022	4	Team	Main report work
16/05/2022	6	Team	Main report work, merging React-branch into main branch
18/05/2022	5	Team	Main report work, commenting code and removing unused functionality
19/05/2022	9	Team	Finalizing report and delivering

Summarized time log

Estimated weekly summary of hours per person, considering both team and individual work		
Week nr	Hours:	Description:
Week 2	12	Bachelor task startup meetings, Meeting with the client and work after.
Week 3	20	Project plan work
Week 4	21	Jira, scrum and Gantt
Week 5	25	Prototyping
Week 6	20	Frontend and backend work, getting used to the tools
Week 7	25	More frontend and backend work
Week 8	20	More frontend and backend work
Week 9	25	Various functionality work, skeleton for main report
Week 10	31	Threat Total development
Week 11	29	External API integration
Week 12	30	More development
Week 13	32	More development and live demo prototype for NTNU soc
Week 14	19	Further development after feedback from SOC
Week 15	15	Threat Total development
Week 16	17	Development and delegating more tasks
Week 17	12	On site development working
Week 18	39	See team time log
Week 19	36	See team time log
Week 20	22	See team time log
Sum	450	

Logs of all the meetings

Meeting with supervisor, 21.01.2022:

- Agile stand-up / weekly meeting: 5–10-minute meeting where we discuss what we have done, what needs to be done and organize future work. Stand-up because meeting where people stand are often quicker or more efficient.
- Use Kanban boards to organize.
- With advisor after twelve on Mondays, or before twelve on Thursdays.
- Do not spend a lot of time finding the perfect framework, see what works for your team and follow it.
- A part of the bachelor is writing why we used a framework / method, how we used it and such.
- The project is one of the most useful projects in real life.
- It is important for the interface to be accessible, easy to view and understand.

FRAMEWORK:

Find a form that fits us best as a group

Check out agile methods

Find a method/framework that works best for us as a group.

Check out agile methods.

SCOPE:

Agree with client the scope from the hours we have available.

Can adjust what will be done and what will not be done, and the quality of this. What is the quality requirements?

Important to state what will not be done, define scope accordingly.

WEEKLY STATUS MEETINGS:

Espen – Mondays after 12 and Tuesdays before 12

meeting log 21.01

start writing the main report AT ONCE.

Writer down points and all we have done!

Scope – what we SHOULD do, what we SHOULDN'T do

Let client know how many hours that has been used

Iterative model

What are the quality requirements?

Time, quality, resources, pick two

Most important thing about task is to show that it works

Basic agile → regular short meetings (a common understanding of the situation)

- What was done since last time?
- What do we plan to do next time?
- What can stop us from doing this?

5 – 10 minutes

A part of the bachelor thesis is to describe how the method is used in the task, therefore a simple and intuitive model should be implemented for better task execution

Meeting with supervisor, 24.01.2022:

Organize meetings where we declare / decide upon expectations. Decide on what can be done within the allotted time and what the client wants done. This is an important bullet point for the scope and time plan. We should document that we have had dialog with the client and resulting adjustments. It is not necessarily important to document everything said in the meeting, but changes that come as a result from meetings are important to note. This is a give and take process. Think about the meeting beforehand, prepare. What do we know and what can we easily learn?

Meeting 28.01.2022 with clients, mostly focused on defining the scope of the project and contracts for the project plan:

Include staff as the main target audience of the project.

Add a point under scope: “The application should be able to show disposition for indicators (Domain, IP, etc.)”

Providing a screenshot of the URL / webpage provided will be optional.

Blacklist functionality, don't return data on certain indicators.

Meeting with SOC v/Christoffer and Frank, 28.01.2022:

- Further defined Scope as we were missing some information here
- Clarified when we're getting the agreement signatures
 - Christoffer will go through this in the weekend
 - We'll further finish this on Monday
- Figured that it's hard to define what we're NOT doing in the scope, this early in the project.
 - Would have to define this later when we've figured out what implementations we're doing.

Meeting with supervisor, 31.01.22:

- Informed supervisor about our progress with the project plan as well as communication with NTNU SOC
- Agreed with supervisor about sending the finished project plan on email by the end of the day.

Meeting with SOC / Frank 04.02.2022

Information if several people have searched this URL and other statistics

Admin interface/ advanced mode

Historical data, we have seen the link before and screenshot

All steps in redirects if possible

Show the user all the different parts via redirects from start to finish

There is a page on "innsida" about graphics and stuff

File hashes and information about malicious files are stored on an internal system at the SOC

04.02.2022

notes frontend meeting

1. Do we run Anti-Virus agent? How many?

- We will handle reputation firstly
- It does exist sand boxes which will handle analysis if needed

2. Special demands about NTNU branding?

- Is enough with what we got

3. Special colour demands?

- Should be within NTNU's colour profile
- Information about this on NTNU's web pages

4. Should we include a help page which explains the use of different web page functionalities?

- Don't do anything big out of it, keep it simple stupid

5. Do we get access to FEIDE-login?

- We will get this eventually

6. What information about file hashes are stored?

- NISP, AlienVault, can also fetch externally

07.02.2022, meeting with supervisor

Add risk mitigating steps for our risk evaluation

Hold notes and thoughts in one place

Frode will go further into report writing

Important to start with the main report early!

Important to store all used sources! NTNU has a very strict policy regarding this

11.02.2022, decisions about frontend modules

Lit elements pros:

- +use of bootstrap
- + simpler to transfer data from frontend to backend
- + should not be problematic to send JSON data around

Currently using lit elements, can use tailwind inside lit elements

➔ Using node stack

21.02.2022, meeting with supervisor

- A lot of work with cloud assignment has prevented Peder and Johannes from Bachelor work
- Changed front-end library from lit elements to ReactJS
- Still waiting for API access
- Should start thinking about backend organization
 - Two backends

Something things to start with:

- How to organize project and code regarding API access
- Look at how to transfer JSON data between web server and Golang

07.03.2022

- Our plan ahead:
 - Waiting for API access
 - Started to integrate third party API's
 - Start with testing
 - User testing?
- Any other big barriers?
 - No, just API access
 - Some trouble with use of react
- Should start with the report

- Deliver first draft of main report before easter break
 - Espen will look through this and come with his taught

Meeting with supervisor, 14/03/2022

- How to get out of a black hole
 - Take a walk outside
- What we have done this week:
 - Main report work
 - File handling in front-end
 - About page
- What our plan is forward:
 - Continue working with previous week's tasks
- Espen will talk to the SOC regarding API access

Meeting with supervisor, 21/03/2022

- What have do done since the last meeting?
 - Implementation of public API's functionality.
 - Backend development, frontend development.
- Our plan ahead
 - Main report work
 - Integration with NTNU SOC's tailormade API (when we get access to it)
- Do we see anything that can stop us from continuing the work?
 - No, not really. It really comes down to us team members and how we work
- NTNU SOC said that we will get API access by the end of this week.

Meeting with supervisor, 28/03/2022

- How is the progress since last meeting?
 - Tried to divide tasks between team members
 - Still waiting for API access
 - Starting to get finished with frontend development
 - The plan is to get a meeting with the SOC by the end of this week
- Greatest obstacles until now:
 - Poor communication with SOC
 - We were not given access to the API within the timeframe we were promised originally
- Plan ahead:
 - Continue working on current tasks
 - Get access to APIs
 - Continue working on main report
- What should be included in the first draft of main report?

- Structure of the entire report
 - One finished chapter, if possible
- Main report work should not be in the way of getting API access

Meeting with NTNU SOC, 01/04/2022

- Colour scheme on result page:
 - Remove red colour when result come back as safe (placeholder for now)
- Feide login:
 - We can make a template ourselves via dataporten
 - Kent will fix feide connection, or documentation
- Branding assets:
 - Use what we're using now, despite that this is png and not svg
- Caching:
 - Memcache will work just fine
 - 1 hour cache time as default, but this should be a configurable parameter
- Languages:
 - Cookie solution and URL will work fine, this decision is up to us
 - Try to use URL first
- Blackboxes:
 - Threatgrid, their malware sandbox
 - Will use about 5 minutes to analyse the input file
 - If this doesn't return any result, this sample should be sent to manual analysis
 - Check if this file has been analysed previously to prevent duplicates
 - Misp, open-source threat database
 - Php website with database as backend
 - We can setup our own misp server in docker for testing
 - Huge variety in reported values
 - Lack of security when testing
 - Threatresponse:
 - Commercial service
- Test instance:
 - Pushes out data we can use for reputation
 - JSON
- Filtering:
 - It is quite easy to filter data based on access level
- Secure file handling:
 - Best practice regarding GDPR
 - File will be analysed at NTNU SOC. No other 3. Parties
- Case tracking:
 - Button to send email and create a case with a case number
 - Details about user will be fetched via the login with FEIDE
- Screenshots of URL's webpage (OPTIONAL):

- Christoffer posted a GitHub repo which offers this functionality
- We can work at the SOC if we want to
 - We have access to meeting rooms and free coffee!

Meeting with supervisor, 04/04/2022

- Access to SOC API yet?
 - Have not gotten access to SOC API's yet, but information about them
 - Have gotten information and clarification around question in the meeting with the SOC
- What to do:
 - Implement caching
 - Work with the feedback from SOC
 - Peder and Johannes are nearly finished with cloud assignment, more focus on bachelor work
 - Should deliver a first draft of main report to Espen before easter break (absolute deadline: 12th april)

Meeting with supervisor, 22/04/2022

- API access:
 - Originally finished, but vulnerable (which is why we haven't got access yet)
 - We have a docker instance already running with MISP on it
 - Use this as the SOC use I similar instance to store internal data.
- We have already implemented checks against public antivirus agents.
- Explaining reasoning of why communication with internal APIs is not implemented.
- Include in report what is necessary to go from local running version to version in production.
- Demo on the presentation can affect final grade.
- 1 slide = 3 – 5 minutes

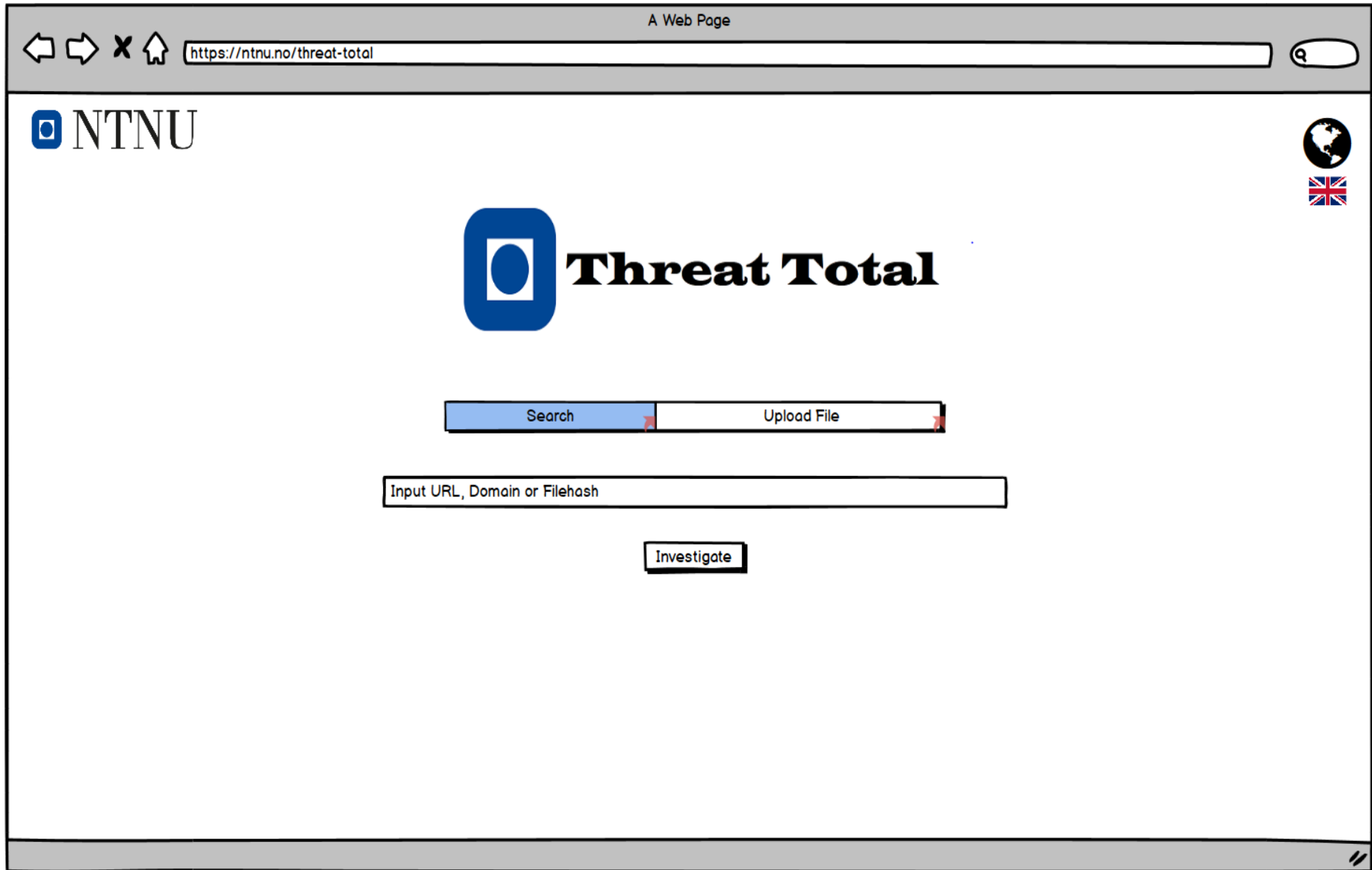
Meeting with supervisor, 02/05/2022

- Plan:
 - Get finished with things we have started to implement
 - File upload
 - FEIDE integration
 - Caching
 - Then starting to write the report full time
- Either live demo or recorded for the presentation
- We must expect questions off topic in the presentation

Meeting with supervisor, 09/05/2022

- What have we done:
 - Tying up the code
 - Main report work
- Deliver final draft to Espen, May 14th
- What will we be doing:
 - Main report work
 - Finish last code functions and commenting
- Demonstrate live language change without new API call

Wireframes



Wireframes

