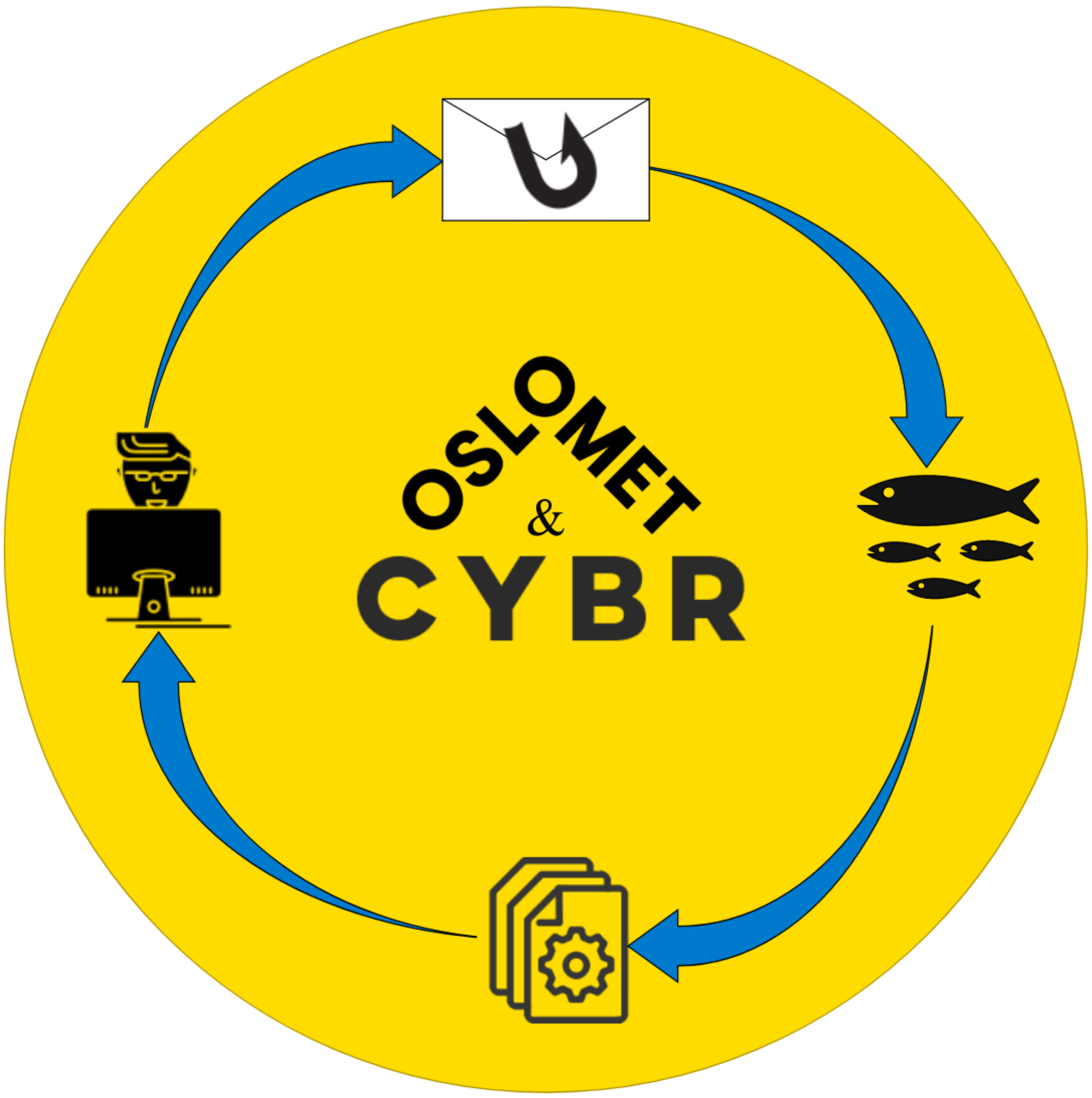
Prototyping of an email phishing emulation system



Martin Bang, Frithjof Brate, Kristian Marison Haugerud,

M. Tayyab Khalid, Jørgen Sandnes

PROJECT NR. 23
AVAILABILITY Open – except source code

Phone: 22 45 32 00

Post address: Postboks 4 St. Olavs plass, 0130 Oslo
Visiting address: Holbergs plass, Oslo

Bachelor Project

PROJECT TITLE Prototyping of an email phishing emulation system	DATE **26.05.2020** NUMBER OF PAGES: **131** ATTACHMENTS: **7**
GROUP MEMBERS Martin Bang s315602 Frithjof Brate s336406 Kristian Marison Haugerud s331048 Mohammad Tayyab Khalid s319229 Jørgen Sandnes s331423	INTERNAL SUPERVISOR Dr. Lothar Fritsch

CLIENT CYBR	CONTACT PERSON Jaan Kitsuk

SUMMARY

Prototyping of an email phishing emulation system is a bachelor project done at OsloMet in collaboration with CYBR. The team has developed a PoC that can perform emulated phishing attacks and track actions performed by phishing targets. The product supports multiple companies at once and has administrator functionality to manage the different companies and comes with 20 remade phishing attacks with support for adding more. It is made after the API-First approach, and includes a simple GUI frontend to visualize the potential of the product.

3 KEYWORDS. Phishing	Proof of Concept	API-First

About the report

The report is optimized for digital reading, with hyperlinks, and cross referencing. Digital reading is recommended, but it also supports printing and reading in paper format.

Following expressions listed below has been used throughout the report:

- "the team"
  - student group of five from OsloMet
- "the product"
  - The rest api, the frontend, and the phishing attacks produced by the team.
- "the project"
  - everything regarding the project and its surroundings.

At the discretion of the client, the source code is not disclosed to the public, but the project provider agreed to show code snippets in this report.

CYBR is a english speaking company with international employees. All communication with the client has been in English hence the team felt it best to write the report in english.

We would like to thank the team at CYBR for helping us with the project from start to finish. Especially thanks to Prahkar Srivastav and Jaan Kitsuk for providing invaluable insights on technical questions and useful suggestions to improve the product. They spent a lot of time with us to make sure that the product and project was as good as possible.

We would also like to thank Dr. Lothar Fritsch for helping us as a supervisor with the various problems and questions regarding the project.

Oslo Metropolitan University, May 2021

Reading Guide and Chapter Introduction

This introduction serves as a guide to who should read what chapter and is the recommended reading order of the documentation. It includes a short introduction of each chapter and describes who the chapter is intended for. Each chapter in the report also has a short introduction informing the reader what to expect.

Due to the size of the test report, user guide, phishing-attack, and API-documentation they have been separated out of the main document.

1 Presentation and software requirement specification

Software requirement specification with introduction of the project and stakeholders. Puts context for who we chose to solve the task and worked with the client.

2 About phishing

This chapter is a short introduction to phishing, different types of phishing and some real life examples. The aim is to give the readers with little experience about the topic context for the rest of the report. It also describes some of the phishing methods the team has implemented.

3 Process Documentation

Provides insight to the team's working methods and tools used. Explains the development process and what decisions were made during the project.

4 Production Documentation

Describes the product developed in detail. It is intended for developers working, maintaining or developing the product's future. The chapter shows how the different components of the product work together. It includes figures and uml with complementary explanations. Code snippets are shown with some examples of full function. Longer code chunks are shown as pseudo code with comments to explain.

5 Appendix / Attachment

1. **Test API Report**
   Test of all the endpoints. Tell the reader about how the test was performed and findings of the test. Readers interested in how the product performed should read this chapter. The chapter is intended for people with basic knowledge of REST APIs. Readers should have read other chapters before reading this for better context.
2. **User guide**
   This is designed for developers and for someone potentially taking over the project and continuing working on it. It is a step-by-step guide on how to set up the backend Django project and frontend. Made at the request of the client.
3. **API Documentation**
   Documentation of all the API endpoints.
4. **Phishing AttackTemplates**
   Overview of all the phishing attacks created during the project.
5. **Frontend source code**
6. **Backend source code**
7. **Insomnia configuration file**
8. **Bibliography**

Glossary

Technical words and acronyms used in the report.

Terminology	Description
API	Acronym for Application Programming Interface. API is a software that allows two applications to talk to each other.
API-first	"An API-first approach means that for any given development project, your APIs are treated as "first-class citizens." That everything about a project revolves around the idea that the end product will be consumed by mobile devices, and that APIs will be consumed by client applications." (Wagner, n.d.)
AttackTemplate	Combination of landing page and email template from Gophish. This object also has additional fields for tracking risk score and phishing type. More info in 4.5.1.2.6. Template & AttackTemplate
Companies	Potential customers of CYBR to perform phishing attacks on.
Docker	A framework for making container based solutions.
Dockerfile	A file that builds the container image.
Docker image	Docker image contains the application and all its dependencies and what makes the container run.
DRF	Django Rest Framework.
Employees / Targets	Employees of the companies taking part in the phishing emulation. More info in 4.5.1.2.2. Employee.
Gophish	Open source phishing framework. The backend API is a wrapper around Gophish.

GUI	Graphical User Interface.
KPI	Key Performance Indicator is used to gauge a company’s overall performance.
KPMG	One of the worlds largest auditing and consulting companies working with auditing, consulting, tax, fees and business law.
Phishing	A cybercrime where the goal is to gather personal information from its victim. More info about phishing in 2.1.1. Regular phishing.
Queryset	Describes collection of objects in the database.
React	JavaScript framework for creating single page applications.
REST API	**RE**presentational **S**tate **T**ransfer.
Server	Django backend.
Spear phishing	A type of phishing that targets individuals. More info about spear phishing in 2.1.2. Spear phishing.
Team	Our development team - a student group of five from Oslo Metropolitan University.
UX	User experience. Usually referred to when talking about universal design for a good user experience.
Whaling	A form of spear phishing that targets high level targets, like CEOs or big celebrities. More about whaling in 2.1.3. Whaling.

Table 1. Glossary list.

Table of Content

List of Figures

List of Tables

1. Introduction and presentation

Introduction of the project, background and the stakeholders involved.

1.1. Project Description

Prototyping of an email phishing emulation system that can be utilized to detect human weaknesses in an organization and estimate the cost of the breach. The phishing emails template should be based on the research of effective techniques used in today's industry. Main part of the project is to create a prototype of a phishing emulation system to analyze and present the data. The templates created will be used to launch phishing campaigns with the phishing tool produced.

1.2. Presentation of the team

Martin Bang, s315602	Martin was in charge of handling the cloud deployment for the software. He had to make sure that the code would run properly when deployed. He also worked on the backend developing some main functionality in the early stages. During the development cycle he was also in charge of the project diary and made sure that it was updated after each meeting.
Frithjof Brate, s336406	Frithjof had multiple roles throughout the project. However, the following were the most profound ones: Scrum master, backend research & design, and one of the developers who focused on creating the backend API.

Kristian Marison Haugerud, s331048	Focused mostly on the backend development in django throughout the project. Active in coming up with different ideas for features of the product. Spent time learning and finding out how the team could utilize Gophish. Last month he made the frontend and docker-compose setup. Responsible for that the Kanban board was up to date and that every group member had something to do and kept the project on track.
Mohammad Tayyab Khalid, s319229	Tayyab was in charge of managing the google drive. Created many of the templates. And was the initiator for creating API-endpoints for landingpages. He also cleaned up a bit in the backend and implemented some simple functionality. And was responsible for the web development for the documentation project website. Tayyab was otherwise a bit everywhere where a helping hand was needed.
Jørgen Sandnes, s331423	Jørgen has worked with the risk score system. Both the strategy of how it should work and with the backend. He has worked with and documented the research, and made some attack templates. Otherwise he has helped wherever it has been needed.

Table 2. Introduction to the group members.

1.3. Presentation of the client

CYBR is a corporation whose business is within the field of Cyber Security. One of their goals is to gamify security training to increase awareness of potential threats within the industry. They also provide security tools that scan for weaknesses within their customers' websites, network configurations, and servers. The goal is to create security tools that are automated, so that companies don't need to hire expensive consultants. The collected data of training and threats can then be reported back to the customers, who then can decide on the necessary precautions.

1.3.1. Project stakeholders at CYBR

CYBR,	Hovfaret 13, 0275 Oslo,	https://cybr.no
Joe Jones,	Director of Operations,	joe@cybr.no
Jaan Kitsuk,	Project Owner,	jaan@cybr.no
Prahkar Srivastav,	Technical Supervisor,	prakhar@cybr.no

1.4. Current situation

CYBR is very heavily involved in commercialisation of its newly released product. Due to this, the resources they can use for prototyping are limited. They want to use us as a team to help them improve their phishing emulation product, that's not released on the market yet.

They want to help their customers remove the need for consultants to help them with cybersecurity and automate the process. Currently, the main focus of the CYBR is to create a training platform, where each employee has a tailored training program. The training program is generated based on the employees simulated email phishing fail rates, and other key performance indicators (KPIs) to gauge the company's overall long-term performance. CYBR's goal is to release a phishing emulation product during the first half of 2021. This product should check how good the employees are at detecting phishing emails based on the position, industry, and Phishing KPIs .

1.5. Software Requirements.

This software requirement was the baseline of the project. It was used as a non-final control document to guide us during the project. Originally, we received an initial specification which evolved during the project. After feedback on the pre-project report and exploratory work . The team finalized the specification as follows, per 01.02.2021. We will discuss implementation and solution of the requirements in chapter 4.8. Compliance between requirements and the product.

1.5.1. Research Requirements.

- **R1:** Find the most efficient phishing methods
  - **R1.1:** Use this research to make the most efficient attack templates.
- **R2:** Research the possible cost of breach if you are a victim of a phishing attack.
- **R3:** Research different possibilities to make emulated phishing attacks. Which tools to use etc.
- **R4:** Research how phishing awareness training of employees improves the cyber security of an organization.
- **R5:** Research how a phisher works. What his methods are and how he plans his attacks.

1.5.2. Technical Requirements

- **R6:** Phishing servers will be whitelisted in firewalls, bypassing any security measures enforced by companies.
- **R7:** Open-sources phishing framework as the backbone of the project.
- **R8:** Frontend should be a minimalistic GUI for data presentation.
- **R9:** It should be easy to import email and or other types of information
- **R10:** A showcase of statistics for the different campaigns that were used.
- **R11:** The possibility to create custom user groups.
  - **R11.1:** Ability to create premade groups as well based on position and departments.
  - **R11.2:** Groups for each company.
- **R12:** A database with pre-made email templates.
  - **R12.1:** Use the most efficient templates based on the research.
  - **R12.2:** Can be stored in the system or a github repo.

- **R13:** Store metric data over different campaigns separately for each company/department. Including but not limited to:
  - **R13.1:** Emails sent.
  - **R13.2:** Emails accepted.
  - **R13.3:** Emails delivered.
  - **R13.4:** Emails opened.
  - **R13.5:** Links clicked.
  - **R13.6:** Emails reported.
  - **R13.7:** Data gathered.
- **R14:** An option that allows for scheduling before a campaign is launched.

1.5.3. Additional features (if time)

- **R15:** Long term engagement with high priority targets.
- **R16:** A grading system (based on how many clicked the link, which can be helpful later when they are going to calculate the costs. e.g., Risk level.)
- **R17:** Email import: import from other management tools, such as azure active directory

1.6. Framework requirements

1.6.1. Security

Securing the system has multiple aspects:

1. **R18:** In the test cases no sensitive data will be stored, and all information are test cases and imaginary numbers. Therefore, security of the databases is not a priority.
2. **R19:** Authentication and verification of users from different companies will be implemented to demonstrate data separation in a multi-tenant system.
3. **R20:** Only the django server will be exposed to the internet. Gophish servers and api keys will be hidden behind the exposed REST API.

1.6.2. Scalability.

R21: The full system will be containerized in separate docker containers for the backend, frontend, and Gophish. **R22:** Each of these components can be load balanced. **R23:** The Gophish server supports having multiple companies.

1.6.3. Usage.

R24: The main aspect of this software will be the REST API. Therefore, the team will provide an API Documentation which states what endpoints are available, their functionality and expected output.

2. About Phishing

Phishing is a cybercrime where the goal is to gather personal information from its victim. The term phishing is derived from the word fishing. You can say that the attacker is fishing for sensitive information. The targets are contacted by email from an attacker posing as a legitimate institution to lure the victims into providing sensitive data such as personally identifiable information, banking and credit card details, and passwords.

Figure 1. Phishing (Embla.net, 2020)

A phishing email usually tries to make the victim click on a link or download an attachment. The link usually directs to a web page that looks completely like a familiar webpage. By this they want the victim to use their login details because they might think it's the real page. However, in reality, the victims give their personal information away to the attacker.

Another type of phishing attack is that the attacker makes the victim download a spyware without knowing. For example, by clicking a link or disguising it as something else. This way the attacker can track the user's activity and for example get his private information.

Phishing attacks do often utilize methods that there is a lot of attention around. One example is the COVID-19 pandemic where frauds regarding this have been popular. Examples of COVID-19 scams are emails telling the victims that they can register to get the vaccine, by submitting personal information during the registration. Or emails telling the victims that they may have gotten infected by COVID-19, so they need to log in to register themselves. COVID-19 occupies most people's minds, and therefore mails regarding this is something a lot of victims would be interested in. (Abnormal Security, 2020)

2.1. Phishing methods

2.1.1. Regular phishing

This is the most common form of phishing. Here are the emails sent to a large group of more or less random people. Emails are often gathered by scraping the web. The emails used in regular phishing are usually generic so they can lure a lot of users. This could be emails acting as they are from a big company, social media platform or another service that is widely used. These services have a lot of users and therefore there is likely that some of the targets use these. Emails could for example ask the victims to update their personal information on the service. The success rate on regular phishing is very low, but since it is sent out to a large group of people it will always be someone who believes in it and does what they are told, and even a 0,01% success rate could be good if the email just goes out to enough people.

2.1.2. Spear phishing

Spear phishing is more complicated than regular phishing. It is a personalized phishing attack where an attacker collects information in advance about the potential victims which then is used to create emails that look legitimate and are appealing to the target. Spear phishing attacks do often target companies. To gain the recipients' trust, the email will often refer to internal information known to the recipient. In a spear phishing attack against a company, this information could be information about their customers, suppliers, partners, employees, or other agreements they have. This is done to build credibility in the phishing email, Spear phishing has become very common against organizations and companies, and in 2019 did 88% of organizations worldwide say that they were subjects for spear phishing attempts. (Proofpoint, 2020)

2.1.3. Whaling

Whaling is a form of Spear Phishing that targets high profile individuals. For example, business leaders, politicians, and celebrities. To target these types of people high on the food chain is usually difficult, so a successful whaling attack requires a lot of research and planning.

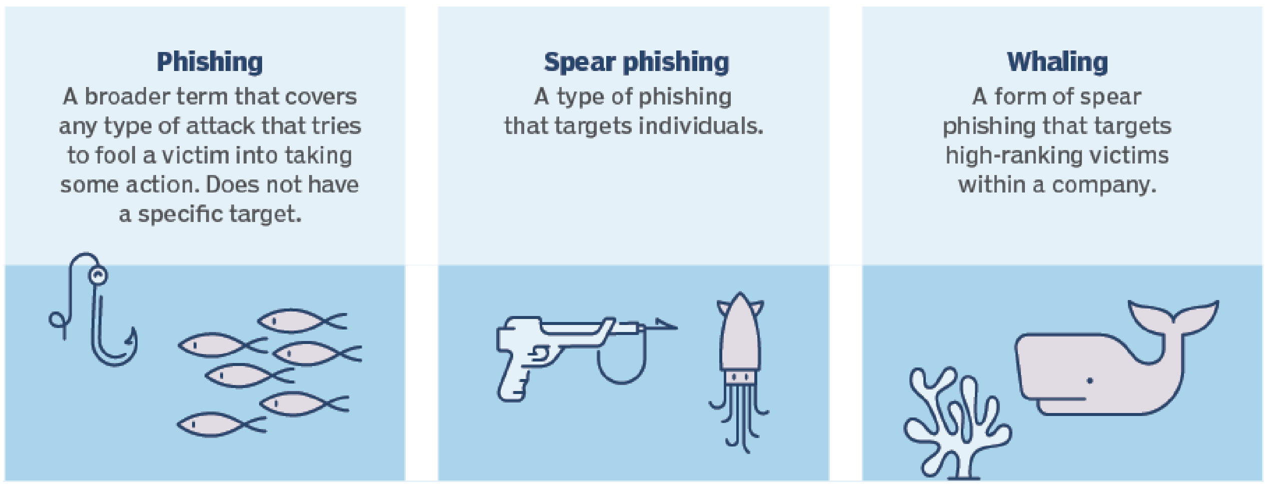
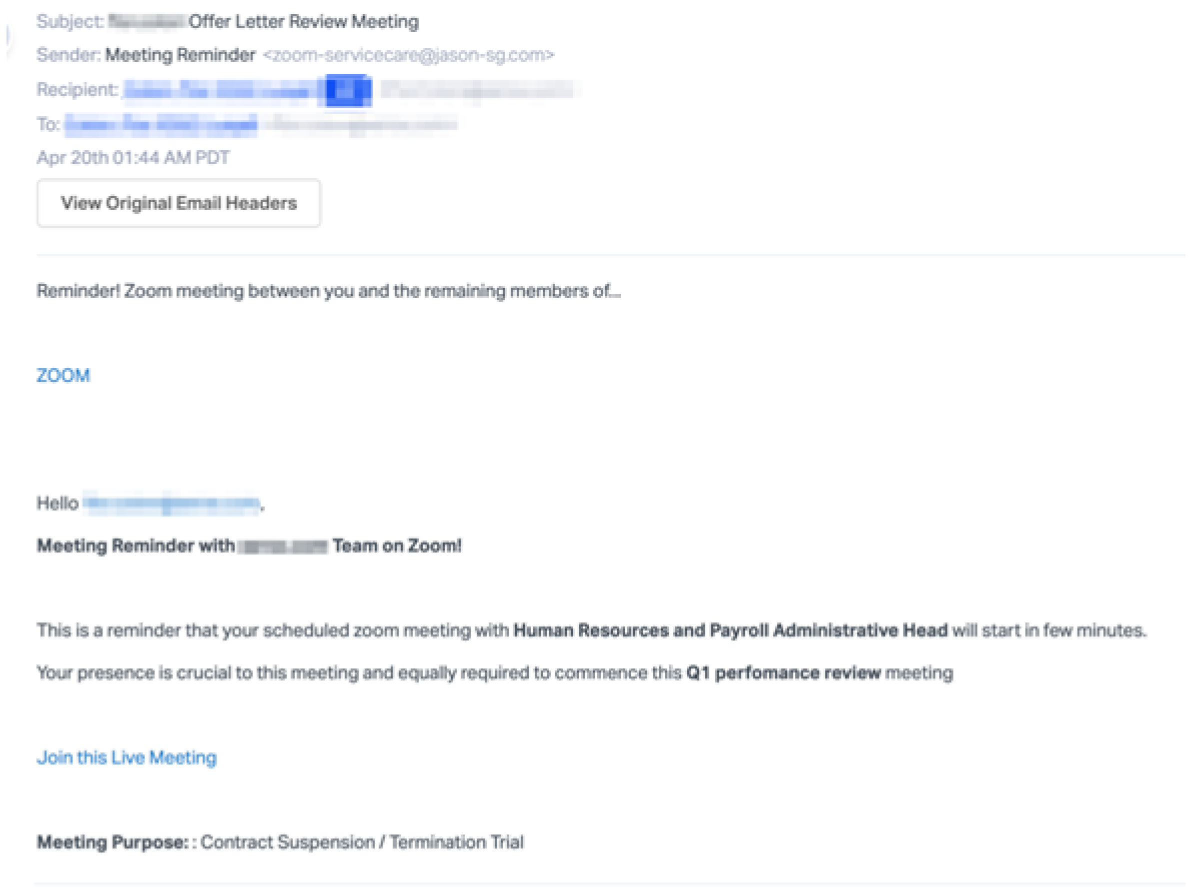

Phishing
A broader term that covers any type of attack that tries to fool a victim into taking some action. Does not have a specific target.
Spear phishing
A type of phishing that targets individuals.
Whaling
A form of spear phishing that targets high-ranking victims within a company.

Figure 2. Phishing types (Lutkevich, 2021)

2.2. Real life phishing attacks

2.2.1. Regular phishing attack

A typical scam is Zoom phishing emails. The attackers are sending out fake Zoom meeting notifications, telling the targets that they must join a zoom meeting. When the receivers click the link, they get sent right to a fake Zoom login page where they have to login before they can access the meeting. But as said, this page is fake, and it will steal the credentials they type in instead. (Abnormal Security, 2020). An example of a zoom phishing email is shown in Figure 3.



Subject: [redacted] Offer Letter Review Meeting
Sender: **Meeting Reminder** <zoom-servicecare@jason-sg.com>
Recipient: [redacted]
To: [redacted]
Apr 20th 01:44 AM PDT

View Original Email Headers

Reminder! Zoom meeting between you and the remaining members of...

ZOOM

Hello [redacted],

Meeting Reminder with [redacted] Team on Zoom!

This is a reminder that your scheduled zoom meeting with **Human Resources and Payroll Administrative Head** will start in few minutes.

Your presence is crucial to this meeting and equally required to commence this **Q1 perfomance review** meeting

Join this Live Meeting

Meeting Purpose: : Contract Suspension / Termination Trial

Figure 3. Example of a zoom phishing email

2.2.2. Spear phishing attack

In 2017 the Canadian university MacEwan University was scammed for 11.8 million dollars in a spear phishing attack. The attackers sent emails to the university pretending that they were from a construction company where they requested them to change their payment details. Those who were responsible for handling this at the university didn’t verify if the email was real, and they ended up paying three payments to the scammers. The University didn’t realize that they were scammed before the real construction company contacted them to inquire about why they hadn’t paid their bills. (Nohe, 2017)

Later investigations have shown that as many as 14 construction companies in the area were exploited as the attackers built fake websites and trustworthy emails. The university had to work with national authorities to try to get the money back. In April 2018 the university reported that they had managed to get 10,92 out of the 11,4 million dollars back. No one was fired after this incident, but three persons were put on paid suspension for a while. (Robertson, 2018)

2.2.3. Whaling attack

In 2017 the French film production and cinema chain Pathé was victim of a whaling attack. (Schwartz, 2018) It ended up with them firing their CEO and their CFO for their subdivision in Amsterdam after they lost around 19 million euro. The Dutch subdivision had a revenue of 209 million euros in 2017 which means that the scammers managed to steal nearly 10% of it. (DutchNews.nl, 2020)

It started on the 8th of march when the chief of Pathé’s dutch division got an email which seemed to be sent by the CEO of the whole chain. The mail asked if he had been contacted by KPMG. After discussing with the financial director, they answered no, and asked for contact details to KPMG. Then they received an answer telling them that the company was involved in a takeover, so they had to pay money to a bank account in Dubai. In the coming weeks they paid even more money in a total of five transactions before they realized that they had been victims of fraud. This ended up with both the financial director and the chief director getting fired.

2.3. What Phishing method are we using?

We are mostly using a combination of 2.1.1. Regular phishing and 2.1.2. Spear phishing. Initially we wanted to make templates for spear phishing only, but because we were not allowed to get any information about the CYBRs clients this proved to be difficult. Instead CYBR wanted us to make as many phishing attacks as possible. They want to have a premade library of templates they can give their customers when they buy the product.

The product should be usable in different sectors within different industries. Therefore, the templates in the library have to be generic. The templates must be usable for everyone after minor adjustments, therefore are social media, google meet and Bank ID scams relevant as this is something almost everyone is familiar with. Minor adjustments can be to add your CEO as the sender for the specific attack.

Then we identified some potential types of attacks that would be interesting to make templates of. We identified these four types of potential attack methods:

- Clicked link.
- Clicked link and then credential harvesting.
- Download an attachment.
- Bank ID credential harvesting.

A Bank ID scam is essentially credential harvesting, but we choose to make the category of templates just for this.

2.4. Phishing Awareness training

So many out there are being phished without even realizing it. Not everyone has the knowledge to avoid becoming a victim of an attack. Furthermore, there are far too many options out there and so many tools to really create sophisticated and believable scams. The only way we can avoid it is by ensuring people are equipped to tackle these kinds of attacks and that's where the solution comes in. Using various kinds of templates, the users will be able to emulate a real-life phishing attack to gain experience which will be of immense value when a real attacker decides to drop a fishy message in their inbox.

Phishing attacks against organizations isn't just about the money they lose. Other factors like downtime hours for their users, their employees needing to work overtime and damage to their reputation can also damage their business.

3. Process documentation

3.1. Summary

Our research has made us realize a lot of things. Maybe the biggest surprise was how big of a problem phishing is for organizations today. We have found numbers of how many phishing attacks that are happening yearly, and how fast that number has increased in the last years. (FBI, 2021)We have also seen how social trends like the COVID-19 pandemic affects how the phishers operate. (Abnormal Security, 2020)

The research has provided us with a good understanding of phishing and how big of a problem it is. This has made it easier for us to plan how our program should work, and to understand which functions it needs to be an effective phishing emulator. We have also found sources about how phishing awareness training should be done, this has helped us when making the program and to understand what to focus on. For more about this topic. See section 2.4. Phishing Awareness training and 3.7.1. Security awareness training.

We have struggled a bit with making the best possible templates for the users. Our research has shown that spear phishing is the most effective type of phishing against an organisation. This means that ideally each company should have their own templates made just for them, but this is not possible for us as we don’t know who the customers will be. Our templates are therefore more generic, this way they will hopefully be useful for a lot of different companies and organizations.

3.2. Preface

In this project the team has researched and developed a tool to perform phishing attacks. The project was assigned by the company CYBR. We came in contact with them after a long search through different companies. They seemed genuinely interested in working with us and we thought their business concept was interesting. CYBR wanted a phishing tool, and had some plans for how it was going to perform and look. They gave us some guidelines to follow, but wanted us to come up with a solution which solved the requirements and other functionality we thought were fitting. The only requirements from them was to do proper research on the concept of phishing and develop a tool which could build upon their original ideas and guidelines.

We started the work process with researching and collecting data on various phishing attacks, and quickly realised that finding the right information was difficult. Most of the time companies would withhold information about their recent attacks, so we were never quite sure about the overall damage caused by these attacks. We consulted the supervisor about the issue and he gave us a list of Norwegian data security organizations which could provide us with more coherent data.

The biggest task was developing the phishing tool for CYBR, which would take up most of the time on this project. We made sure to plan the development cycle ahead and to use an agile workflow to make sure everything was working out. This project would not be possible without the open source tools available and the people supporting them, our product is therefore a combination of the skills and creativity derived from these tools.

3.3. Introduction

3.3.1. Description of CYBR.

CYBR is a company that focuses on creating a gamified cybersecurity training platform that is both fun and interactive to increase awareness of potential threats within an industry. This helps them to provide effective and simple means of security testing to businesses. That further allows businesses to take matters into their own hands and check how vulnerable they are. Allowing them the option to learn and improve their overall system for breaches and other unpleasantries.

They are very heavily involved in commercialization of its newly released product. Due to this, the resources they can use for research are limited. Hence, they wanted to use us to help them improve their Phishing emulation product, which is not released on the market yet. CYBR wants to help their customers remove the need for consultants to help with cybersecurity and automate the whole process. Their goal is to create a gamified cybersecurity training platform, where each employee has a tailored training program. The training is generated based on each individual employee's assessments, emulated phishing email fail rates, and other KPIs.

3.3.2. Project Overview

Prototyping of an email phishing emulation system that can be utilized to detect a weakness in an organization (human links) and estimate the cost of breach. The AttackTemplates are based on the research of effective techniques utilized in today's industry (3.7. Research of phishing attacks). Finally, the product will analyze and present the data. We were encouraged to use available tools in the open-source community.

CYBR wanted us to use the most time on the backend. For the frontend we used ReactJS and for the backend we went with Django alongside Python because both worked well with each other and were relatively easy to work with. We went with Gophish for our open-source phishing framework and utilized its REST API. The request library in Django was used to extract and analyze data from Gophish to be stored in a database, which will be later displayed in a minimalistic GUI. The team adapted the agile workflow with elements from SCRUM and Kanban adjusted to our workflow.In conclusion, we were able to fulfill all of CYBR demands and didn't have to cut out the core parts they wanted us to implement. We have a functional minimalistic front-end and we have created API-endpoints for all the required functionality. Different phishing AttackTemplates were created and formatted in JSON to be supported by our backend.

3.4. Planning and Working method

3.4.1. Planning of the project

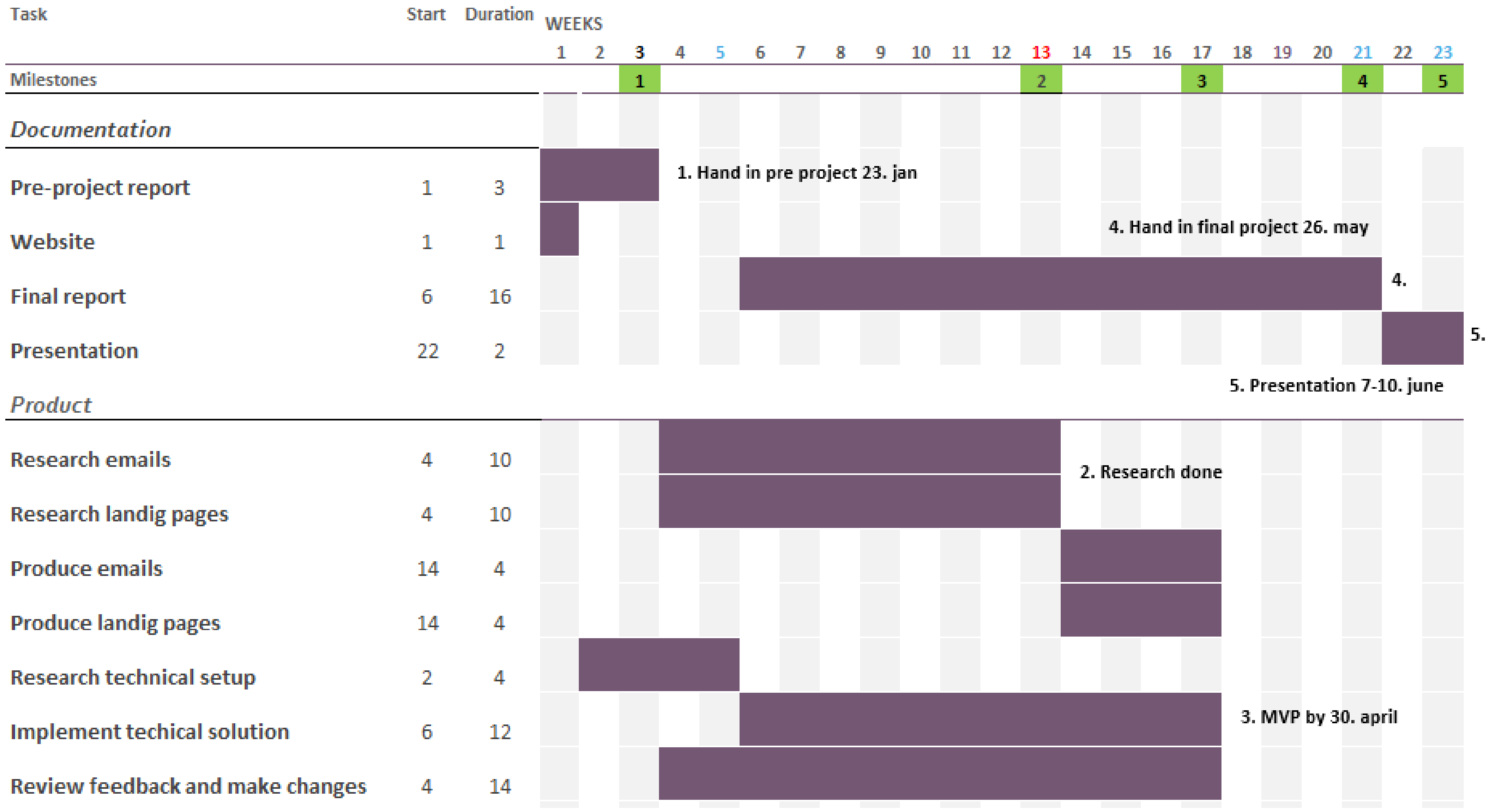
In December we got the initial software requirement specification from CYBR. Internally we had as few meetings discussing the task and brainstorming many different ideas. Before Christmas 2020 we had a few meetings with CYBR to ask questions and clarify anything about the project. The official start of the project was 04.01.2021.

The first task was to find the scope of the project and set a realistic timeframe. Both CYBR and the team wanted to focus on the backend API and make a simple GUI if we had time. Therefore, the team focused on backend development and good documentation of the REST API.

We spent the first time planning and brainstorming to understand the task at hand. The first thing and most important thing we had to choose was an open-source phishing framework. We had a look at all the different frameworks we could find, but we quickly found out that Gophish was the best option for the project. It supported both a REST API and a python API client we could utilize.

As a team we wanted to work with industry standard tools and framework. CYBR gave us free rein to choose the programming languages and tools we wanted to use. The team had some experience with Python and all team members wanted to learn more about Python. So, we found a fitting web framework that was built upon Python. Django and Django Rest Framework come early as a candidate. After reading through the documentation we found out that it fitted the needs. We asked the technical supervisor from CYBR and he also agreed that it would be appropriate.

Figure 4. Gantt scheme of the project shows how we planned the project after feedback from the pre-project report. In chapter 3.5. Development process we discuss in detail our development process based on our plans.



Task	Start	Duration
Documentation		
Pre-project report	1	3
Website	1	1
Final report	6	16
Presentation	22	2
Product		
Research emails	4	10
Research landig pages	4	10
Produce emails	14	4
Produce landig pages	14	4
Research technical setup	2	4
Implement techical solution	6	12
Review feedback and make changes	4	14

Figure 4. Gantt scheme of the project

3.4.2. Software used and why

3.4.2.1. Reading product documentation

Reading the official production documentation was always the go to solution when implementing code. We choose well known and documented frameworks and they usually come with a getting started mini-tutorial from the developers. The first step was always to read and try this out before we implemented it in our own project.

They give recommended ways of doing this and we always follow these tips in the code base. If we still had a bug or a problem we could find out it resulted in using third-party sites like *StackOverflow* or developers writing blog posts about their findings.

3.4.2.2. General software tools

Ubuntu 20.04	Ubuntu 20.04 virtual machine for those not using Linux. To reduce bugs and errors, and spend less time fixing them. More time on development.
GitKraken	Visualize the git timeline. Helpful program to manage pulling, pushing, branches, merge conflicts, and removes the need to remember every git command. Branches utilized when working implementing new features to the product.
IDE	IDE(Integrated Development Environment) of personal choice, PyCharm or VSCode. Everything that gets the job done and what each team member is most comfortable with using.

Insomnia	Used to test API endpoints. Makes it easy to request and see responses. Setup with automatic fetching of tokens. Can easily switch between different companies.
Docker / Docker-compose	Make containers for each service and run as one network with the help of docker-compose. Makes it easy to run the project. Just one person needs to set up the project. Then the rest can run the project with one command.
VM, Google Cloud	Host Gophish server and REST API. Provided by CYBR. For testing and prototyping.
Mailgun	SMTP server provided by CYBR. Makes sending emails to targets easy.
Thunderbird mail	Test different phishing mails. See how phishing emails look in a mail client.

Table 3. List of general software tools used

3.4.2.3. Backend software tools

Django and DRF	Used to create a REST API. Well known and documented. Allows the team to focus on the development. Django handles most of the configurations and at the same time allows for custom settings.
Gophish	Open-source phishing framework used to send emails to target and track their actions. Top most advanced and documented phishing tools that are also open-source.

Table 4. List of backend software tools

3.4.2.4. Frontend software tools

Create React App	CRA framework that uses React JavaScript library. Used to bootstrap the project. Less configurations, lets the team focus on development. Automatic production builds. Easy to maintain and upgrade to newer versions. One the most popular frontend frameworks. Well documented and an industry standard.
Node.js	JavaScript runtime environment. Npm keeps track of packages and dependencies. Used to initialise projects like “Create-react-app”.
React query	Takes care of state, error, and status messages. Handles caching, background updates and stale data. Do not need to manage the global state. Just use a function that resolves or throws an error on the data.
Axios	Used in combination with react query to get and post data. Promised based HTTP client for node.js and the browser. Used to handle the request to the REST API. Transform it into useful data. With a try and catch any errors.
Material UI	React UI framework and react components for web development. To get a website fast up and running for a prototype. Used to get a vizual solution of the REST API, and show the functionality and logic of the website.

Table 5. List of frontend software tools.

3.4.2.5. Communication and planning tools.

Slack	Day to day communications. Host meetings and share screens. Both group chats and one on one calls. Different channels for different topics. Can

	integrate and get reminders from apps like google docs, GitHub, and google calendar. “Every” update in one place.
Trello	Used to host multiple Kanban boards. Visual display of all the tasks we are working on, that are done and are in planning.
Google Calendar	The team made a shared google calendar to note down internal meetings , meetings with CYBR and the internal supervisor from OsloMet.
Google Docs	Host all documentation. Easy to give feedback on each other's work with comments. Every team member has access to every document needed.
diagrams.net	Used for creating uml and flowcharts. Easy to share figures and modify them.

Table 6. List of planning and communication tools.

3.4.3. The team's typical work week.

Stand-ups every 3 times a week, Monday, Wednesday, and Friday at 0900 am. With flexible working hours due to team members working part time jobs. We kept the stand-ups short and to the point. If there were any problems, we divided into smaller groups to save time. We had meetings with CYBR weekly or bi-weekly where we gave them a status report and discussed our findings.

3.4.4. Work method. SCRUM and Kanban

From the very beginning the team adapted an agile workflow based on SCRUM principles and we utilized a Kanban board to keep track of the different tasks. We kept a project diary on google docs from the first meeting back in October for future reference and an overview of what everyone was doing. The project diary was structured with an agenda, meeting minutes report and actions points. Every team member had the opportunity to add something to the meeting's agenda.

This allowed the meetings to be more efficient and we had an ordered agenda for the different topics we wanted to discuss. That made sure that we did not forget anything. As soon as the implementation phase of the project started, we moved the action points to the Kanban board.

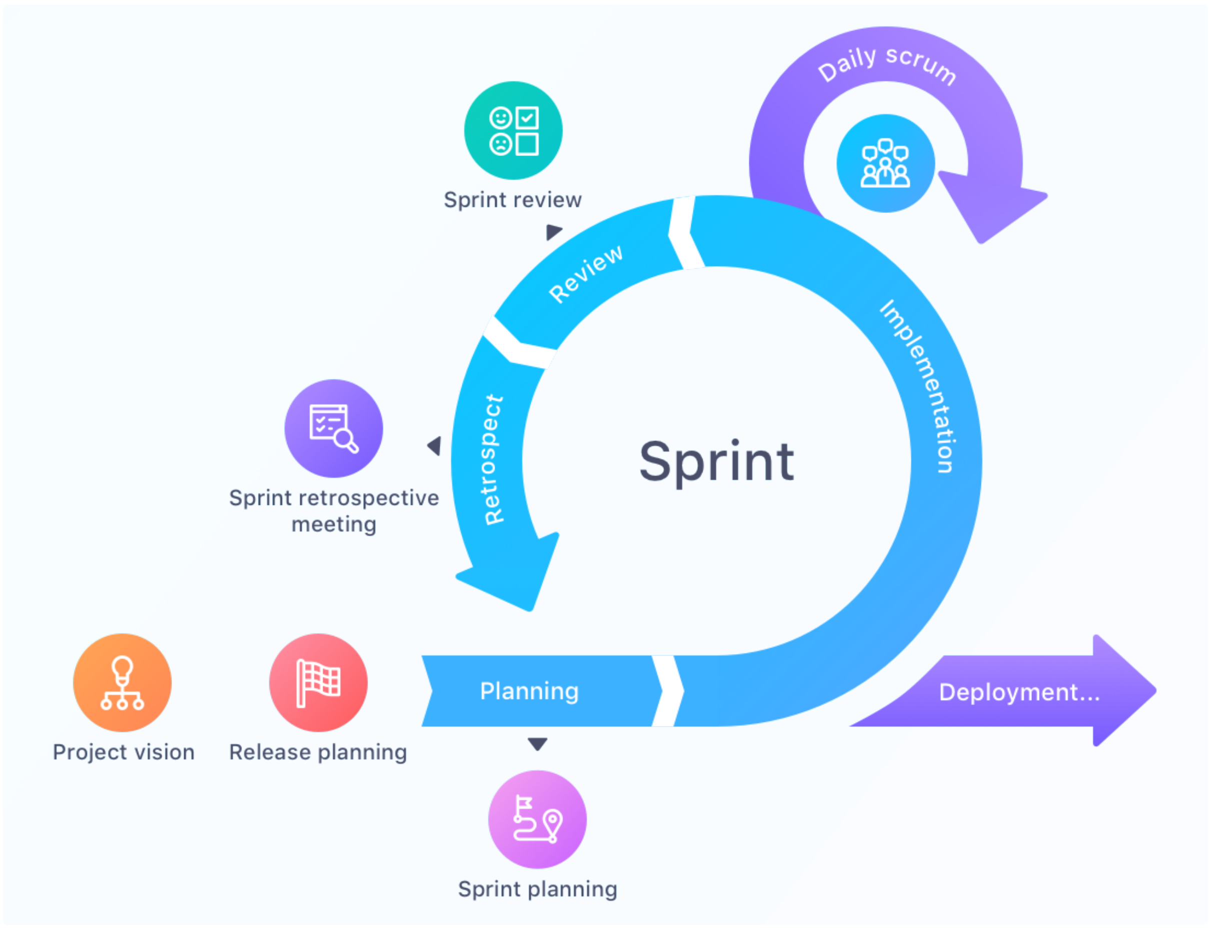


Figure 5. Example of SCRUM workflow (Eclee, 2019)

Initially we started out doing sprint planning, review and retrospective and followed the principles strictly as shown in Figure 5. This was very helpful early in the project and we got to know the different team members' workflow. As the project progressed we could see the path of the product more clearly, so we decided to do shorter weekly "sprints". As the team grew more confident in each other we did not follow sprint meeting principles closely and moved more towards a Kanban workflow. At this point we had most of the backlog cards in

Kanban board and we knew what to do and work on. It was all about putting the work in. We still reviewed each other's work and kept up the tree weekly stand-ups to give each other feedback or help clear roadblocks.

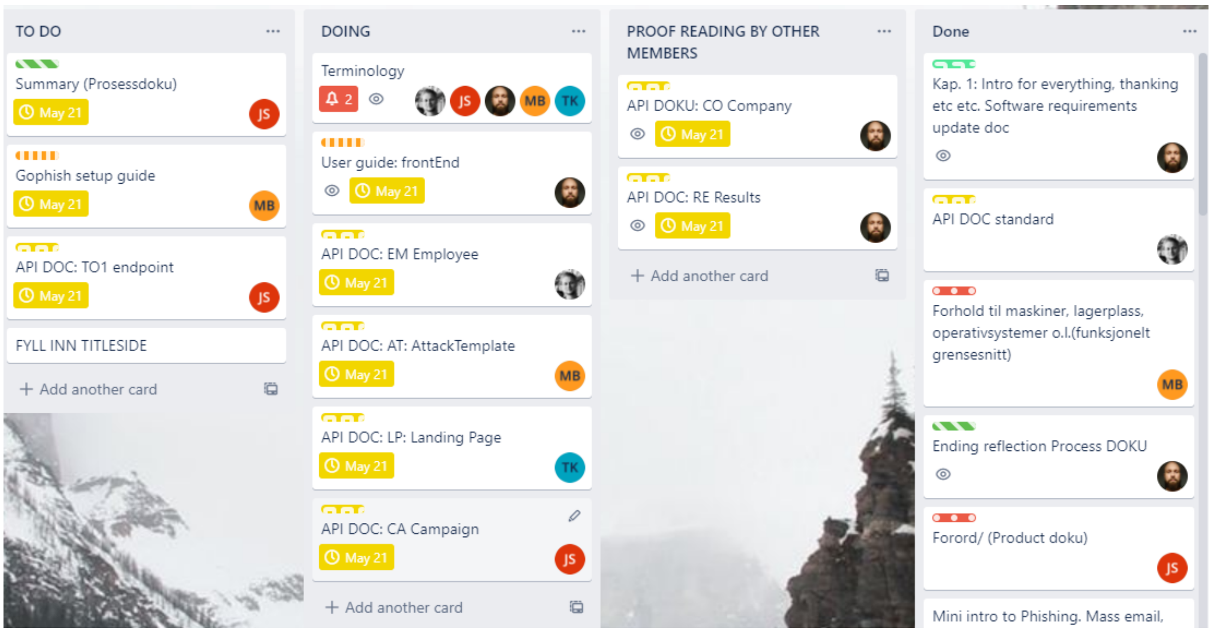
3.4.5. Kanban on Trello

The most useful planning method we found was a Kanban board on Trello. We kept a backlog with all the software requirement specifications. They were ordered in a prioritised list from top to bottom. One team member was given the responsibility to always make the trello board up to date so that the other team member always had something to do.

Each card in the backlog was, if needed, made into smaller and more specific cards. We made sure to always have some tasks on the *TODO* list. So there was always something to do if a team member finished a task earlier than expected. This way members could pick tasks and at the same time inform the team of what they were doing with minimal effort.

After picking a task and marking the card with their name, the card was moved to *Doing* and stayed there as long as the team member was working on it. This gave an overview of what every team member was working on and made sure that we did not do any double work. It also made sure that no task was forgotten. Only tasks in the *Doing* list were worked by the team members, and had to be completed before moving to a new task.

After the task was finished the card was moved to the *Review/Proofreading by other members* column. Here other team members could test or review the work. Then move the card to the done list. If any bug or the task was partly complete. We made a new card in the *TODO* list. Figure 6 is an example of how a typical Kanban board looks.



TO DO

Summary (Prosessdoku) May 21 JS

Gophish setup guide May 21 MB

API DOC: TO1 endpoint May 21 JS

FYLL INN TITLESIDE

+ Add another card

DOING

Terminology 2 JS MB TK

User guide: frontEnd May 21

API DOC: EM Employee May 21

API DOC: AT: AttackTemplate May 21 MB

API DOC: LP: Landing Page May 21 TK

API DOC: CA Campaign May 21 JS

+ Add another card

PROOF READING BY OTHER MEMBERS

API DOKU: CO Company May 21

API DOC: RE Results May 21

+ Add another card

Done

Kap. 1: Intro for everything, thanking etc etc. Software requirements update doc

API DOC standard

Forhold til maskiner, lagerplass, operativsystemer o.l.(funksjonelt grensesnitt) MB

Ending reflection Process DOKU

Forord/ (Product doku) JS

Mini intro to Phishing. Mass email,

Figure 6. Example of our Kanban board during the report writing.

The team decided to make multiple for different topics. As the kanban board quickly got crowded. We had one board for development, research, and final report documentation. This made it easier to have an overview of the different topics we worked on.

3.4.6. Pair programming & sharing screen

Pair programming via sharing screen was utilised throughout the whole project. As the team and CYBR worked remotely due to COVID-19 this became very important in the project. When implementing a hard piece of code for solving bugs, pair programming sped up the process. A fresh set of eyes always helped debugging the problem and taking things step by step to solve it.

Figure 7. Example of Pair Programming. (Sorsa, 2017)

When major steps of functionality, new tools or framework were implemented we often shared the findings with the team. Here every team member could ask questions and clarify anything. This resulted in that team members could easily get up to speed on new information.

3.5. Development process

This section features what the team did on a week by week basis. Highlighting the most important steps of the development process. It is based on the project diary kept throughout the project.

3.5.1. Week 0 - Finding the project

After the group was settled, we discussed what kind of projects we were interested in and started contacting potential employers. The goal was to find something interesting within fields such as AI, Blockchain, or Cyber Security. We were a diverse group with different interests but there was a common goal to find a project which mainly focused on backend development. We narrowed down the list of potential projects to this list.

3.5.2. Week 1, 2& 3 - Pre-project

When CYBR was selected as the project provider we started the project in the first week of January 2021. The first 2 weeks were dedicated to writing the pre-project report and establishing a common understanding between CYBR and us for the scope of the project. During this phase we designed an overview of the program and decided what frameworks, languages, and technologies to use. We also decided to use kanban boards and scrum for organizing the progress and tasks. Tayyab at this time was in charge of developing the project website.

During this initial phase we communicated with CYBR about what specifications we had for the technical and research sections of the project.

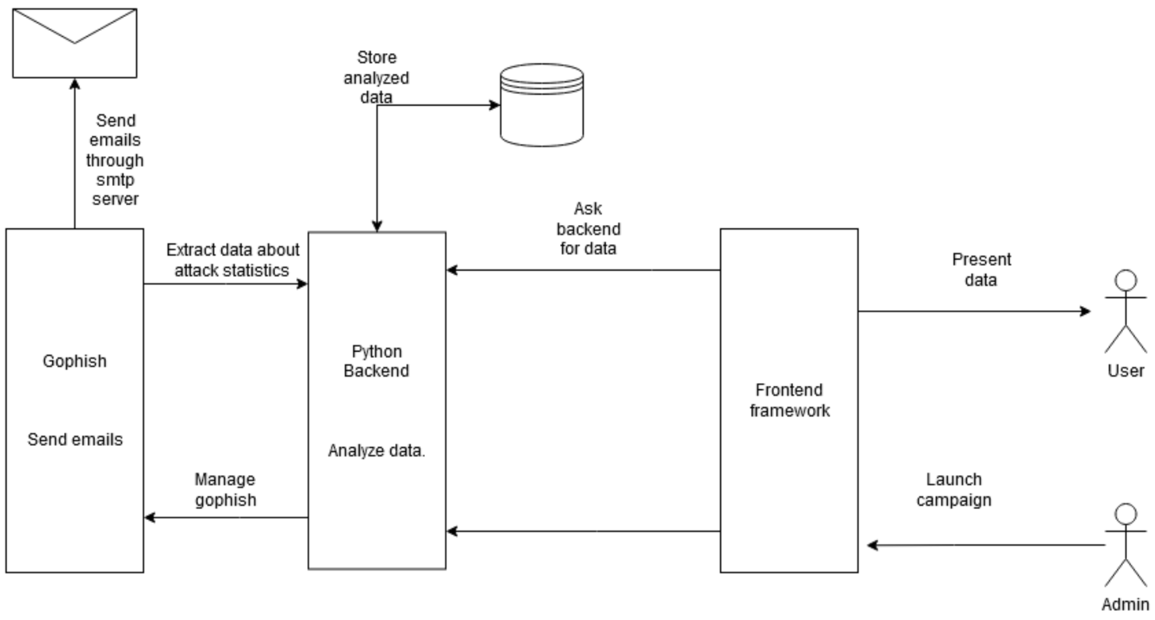


Figure 8. Our initial high level design, created during the pre-project phase.

3.5.3. Week 4& 5 - Research

In week 4 we had the first sprint where we started research on effective phishing methods, attacks and templates. During this time-box we gathered data and an understanding of how to build the templates and landing pages which were going to be implemented in the final program. More about the data and sources analyzed can be found in 3.7 Research of phishing attacks. The following week Frithjof and Martin shifted the research direction into more technical aspects, while others (Kristian, Jørgen, Tayyab) continued the phishing attack research. Kristian researched different features Gophish supported and which of them we could utilize in our product. Frithjof and Martin built a potential solution to handling API requests to Django and forward them to Gophish to utilize the inbuilt API of the phishing agent that was discovered and demonstrated to the other group members.

Forwarding requests from the back-end to Gophish was possible to do in different ways. The Gophish server itself is built to be API first, however in the product we wanted a finished multi-tenant system to handle multiple companies at once. We shifted away from the initial approach of using the Gophish Python API client and decided to control the phishing server using API requests. This way we could forward formated data without having to reformat the JSON in the back-end and therefore save computational cycles.

```
if request.method == 'POST':
   try:
       res = requests.post(host + '/api/pages/',
verify=False,headers={'Authorization': api_key}, data=request.body)

   except:
       return returnMessage("Phishing server error",
status.HTTP_500_INTERNAL_SERVER_ERROR)
   return Response(res.json(), status=res.status_code)
```

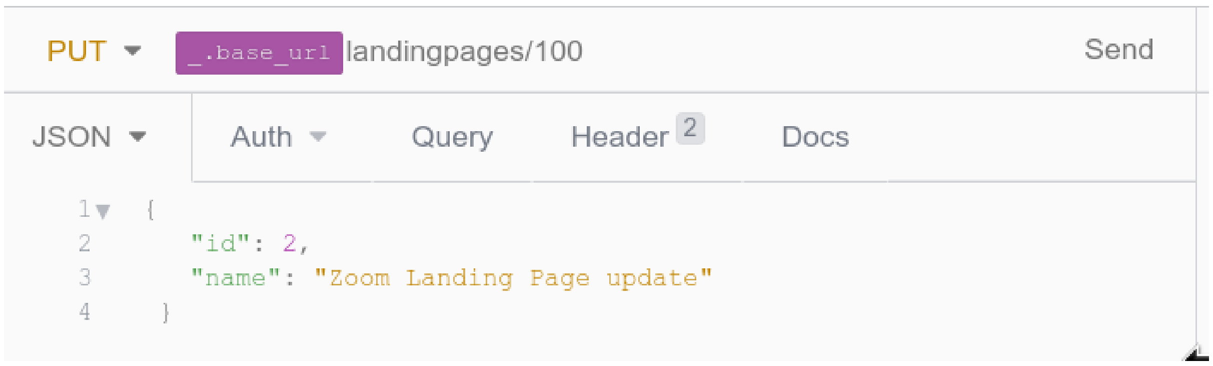
Table 7. Example of how we handle calls on the Gophish API.

3.5.4. Week 6& 7 - Technical implementation basic functionality

The next two week sprint during week 6. and 7. focused on implementing all the functionality of Gophish in the backend project. Endpoints for different functionality were given to each member of the group. This made it so each team member had something to do and could familiarize themselves with Python and Gophish API.

We were aware that some of these endpoints would not be used in the final program but implementing the functionality made sure all the members knew how to work with the code, and built a foundation for developing the more advanced API calls later. All of the API calls were documented with their expected input, output and HTTP methods, this documentation was updated as functionality was added and removed.

During this phase we found an API Design Platform named Insomnia to test the endpoints as they were made. Here we had an informal documentation and test platform for the API calls as we developed them.



```
{
    "id": 2,
    "name": "Zoom Landing Page update"
  }
```

Figure 9. Example of Insomnia call on endpoint.

To solve the multi-tenant 1.6. Framework requirements **R19** we discovered that users in the phishing agent(Gophish) could have different API keys and that way separate data. With this knowledge we started working on a way to specify what Gophish API key to communicate with for each user logged in to django. Tokens were implemented and data separation was achieved.

After this period we learned that the two week sprint was becoming too drawn out and decided to shift more towards a kanban style approach. We still had daily meetings, short reviews and demos regularly.

3.5.5. Week 8 - Campaign creation & Front-end

Week 8 started with a split focus on syncing the Gophish data from campaigns into the backend(Frithjof, Kristian, Tayyab), and to create a functional front-end we could use to demonstrate the program(Martin, Jørgen).

The technical supervisor in CYBR gave us a hint to look into webhooks. This is a functionality where we make Gophish push updates to a webhook, and from there it updates the database in the backend. This removes the need to manually fetch data and then sync changes when data is requested from the front-end. Frithjof looked into implementing the webhook. Kristian started working on the company endpoint in the backend. Figuring out what field we need to store in our own database when creating a new company and at the same time save the response from Gophish in our own models.

Martin and Jørgen developed a simple gui, and started creating login management for the different users. However, after a week we decided that we had more pressing issues and put the front-end development on ice for now. The customer was clear that they were not interested in a front-end. This greatly impacted the choice to scrap the development on that end.

3.5.6. Week 9& 10 Template score & email creation

The program was starting to function as a whole. A lot of the back-end functionality and needed requisites was finished. We took a look at the technical specifications and started implementing the missing requirements in the code.

Frithjof implemented the system which mapped template scores to different templates. This made it possible to increase or decrease the impact made by a successful campaign depending on how malicious the email template was constructed to be. The total risk score calculation was not complete but it was a step in the right direction.

Kristian started working on endpoints showing statistisk for different scenarios. Since the team wanted to extract different information from the campaigns. Total results for all phishing campaigns per company were implemented. Result of one phishing campaign was also developed and others to show statistics in different ways. These endpoints were also implemented so that the admin user could look at statistics for all companies, both total results and per company.

The risk score should also be amplified by the target's positions. To simplify this instead of having individual mappings of each possible position in a company we decided to create risk levels. Jørgen rated these from level 1-5 where each level gets a different amplifier on top of the template_score if they are victims to the campaigns. This saved us and the users a lot of trouble by not having to create roles such as CEO, department leader, CFO etc. These could just be mapped to risk levels such as 5, 3 and 4.

```
def riskMultiplier(self):
    if self.risk_level == 2:
        return 1.25
    elif self.risk_level == 3:
        return 1.5
    elif self.risk_level == 4:
        return 1.75
    elif self.risk_level == 5:
        return 2
    else:
        return 1
```

Table 8. risk multiplier function in Django.

We still wanted to have the ability to send and sort campaigns depending on the department an employee belongs to. Because of this we added a field named 'department' to each employee(target) as well so that we could get statistics on best and worst performing departments. Martin worked on functions for sorting and filtering the employees based on their risk score and department. During this period we also created more test templates for attachment and credential harvesting, to show and test that they were working.

We explained the progress to CYBR, and decided from now on to focus on removing bugs, making the error messages etc user friendly and remove unnecessary code and in general make the code more easily understandable.

3.5.7. Week 11& 12 - Cleanup & risk score calculation

After deciding that the next weeks had two focus points; cleaning up code, and finishing functionality we've started on. We sat down with the kanban board and had a thorough look at what functionality we were going to cut. Things like csv import were scrapped because we already supported JSON imporation. Attachment tracking was laid off since it required remodification of the entire backend api. Bugs and new cards were stored and a deadline was set a week and a half later.

During this week the template scoring system was finished by Jørgen with risk multipliers template scores and email reporting scores. Campaign archivation was implemented by Frithjof which made it possible to stop the campaigns when the user believes the remaining unclicked emails are ignored.

All api calls were tested and rewritten so that they all return correct HTTP responses and descriptive error messages if a problem occurs. If the Gophish server was down an internal phishing server error was also implemented to indicate a problem on the server side, after the initiative of Kristian.

When all the risk score functionality was finished we presented an overview to CYBR of how the risk scores are calculated and how multiple clicks are tracked. See Figure 10. At first additional credentials harvested and links clicked accumulated their points for each click. It was decided that this functionality should be removed and that number of clicks should be stored, but risk score could only be adjusted once per action in each campaign.

It was also mentioned by CYBR that more email templates was a priority and that a front-end could be implemented to show off the basic functionality in a more pleasing way. These points were taken into review and a part of the next sprint.

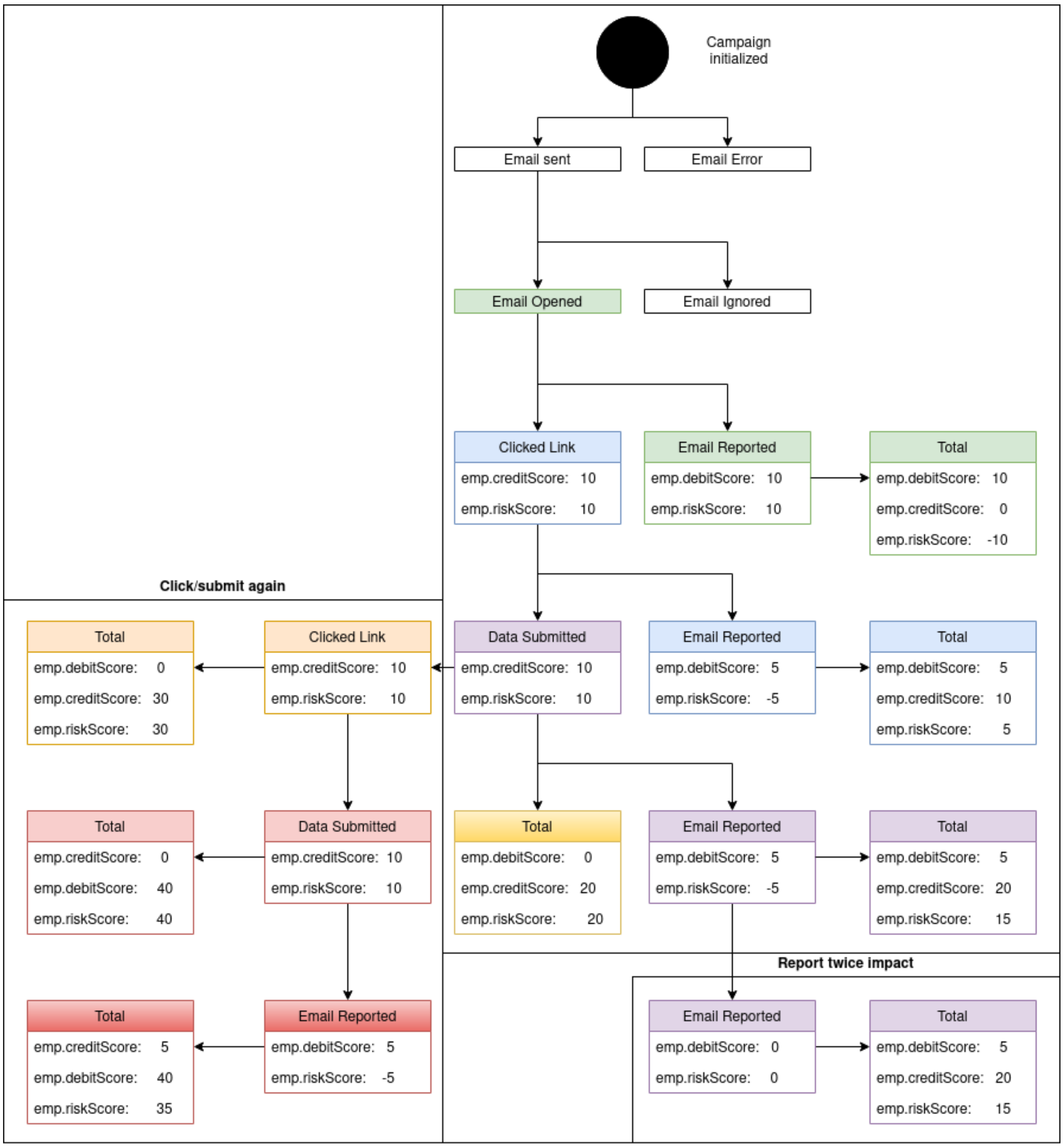


Figure 10. Risk score calculation illustration.

3.5.8. Week 13& 14 - Attack types with different score calculations, templates, deployment & front-end

Cybr wanted more templates. Therefore, we made a plan to create more of those within the attack types. The templates should be mapped to a level which indicates how high the associated risk score incrementation was for that template. Kristian came up with the idea to rebrand email templates and landing pages to phishing attacks. The team also liked the idea and we started to make a few changes to make this.

To achieve this Frithjof added an extra field to the templates, which specified what landing page (if any) was related to the template. We've called the combination of template and landing page AttackTemplate (4.5.1.2.6. Template & AttackTemplate). This way a finished email attack would know it's page if it was present and could calculate what kind of attack type the incoming data(click, submit data, open email) was related to, and then apply the correct score. As an example credential harvesting attacks now give ⅓ of the template score for clicking the link, and ⅔ for submitting sensitive data.

```
harvestCredentialsInCampaign = None

attachmentCampaign = None
if camp.template.page == "None":
   harvestCredentialsInCampaign = False

elif camp.template.page == "Attachment":
   attachmentCampaign = True
else:
   harvestCredentialsInCampaign = True
   attachmentCampaign = False
```

Table 9. Python code of how a campaign is marked as a specific campaign type.

A reset employee risk score endpoint was also made so that their score could be reset after attending a course. This way they stand a chance to get a decent score, even if they did a horrible job before attending the course.

At the end of this week all the reporting and risk score calculations were implemented in the code by Jørgen and statistics and reporting is now successfully updated and tracked for all endpoints.

Kristian started developing the front-end from scratch. Changed the token system to JWT based authentication in the backend and implemented it in the frontend. Learned how to use axios to fetch the api. Then implemented the following features in frontend: login, logout, navbar, and made one simple page showing the result of phishing campaigns using a datagrid by Material UI. Also implemented creating campaigns, creating employees, listing employees, and listing all phishing campaigns.

3.5.9. Week 15: Front-End, web deployment & additional templates

With less than 3 weeks until the internal development deadline, we started focusing on the matters of who were left. Jørgen and Tayyab created more templates. Kristian worked on the GUI for the front-end and made the project deployable with docker and docker-compose(Kristian). Martin handled the deployment on the Google VM, and Frithjof tested the API endpoints, and removed bugs. In addition general bugs and small functionality to support the front-end gui is also added.

CYBR wanted as many templates as possible. This led to us brainstorming more ideas, and adding support for attachment tracking through a loophole in Gophish. Ideally they wanted as many templates as possible. Since the initial goal of the project was to create a prototype which demonstrated the possible effective attacks, we decided to create more templates due to their wish. However, these were within the scope of proof of concept, and not mass production.

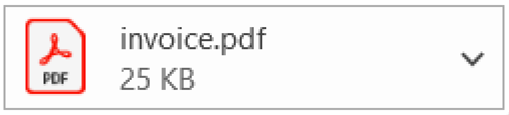


Figure 11. Example of attachment image for tracking attachment downloading.

Dockerization started working out, however the databases are missing crucial information such as webhook setup and predefined password when the system is deployed (the password should change after 1st login). We started looking into pre-injecting the docker databases with the required data to simplify the dockerization process.

In the front-end Kristian realized that *react query* was a good choice for handling the state of the data. So the start of the week was used to refactor parts of the code, to save time and make it easier to implement more pages later. Then implemented features like ending phishing campaigns, listing phishing attacks, editing from field in phishing attacks, and a timeline for employees per phishing campaign.

3.5.10. Week 16: End of development

The last week of development was dedicated to cleaning up bugs, finishing up the templates, and making sure everything was imported successfully on creation of a company. Since the week after was reserved for dockerization, Martin kept working on pre-injecting the docker databases.

Separately the containers worked great. However, we had some issues getting the Gophish container to properly communicate with the backend. After some research this was due to needing to specify the Gophish location through its hostname rather than the address.

```
version: '3.8'

```

```
services:
  django-backend:
    build:
      context: ./poc-phishing-backend
    command: python manage.py runserver 0.0.0.0:8000
    volumes:
      - ./poc-phishing-backend:/poc-phishing-backend
    ports:
      - 8000:8000
    env_file:
      - ./.env.dev
    depends_on:
      - db
      - gophish
  react-frontend:
    build:
      context: ./poc-phishing-frontend
    volumes:
      - ./poc-phishing-frontend:/poc-phishing-frontend/build
    ports:
      - 3000:3000
    command: npm start
  db:
    image: postgres:13.2-alpine
    volumes:
      - postgres_data:/var/lib/postgresql/data/
    environment:
      - POSTGRES_USER=<user>
      - POSTGRES_PASSWORD=<password>
      - POSTGRES_DB=<name_of_db>
  gophish:
    image: gophish/gophish:latest
    ports:
      - 3333:3333
      - 80:80
volumes:
  postgres_data:
```

Table 10. Docker-compose yaml file

The remainder of the templates were verified and tested that worked as intended by Tayyab. The project seemed to wrap up well. We had a meeting with the product owner and scheduled for a demo in late May. That way we had the remaining 3-4 weeks to focus on finishing testing and writing the report.

In the front-end Kristian implemented an admin view. Where admin can create, update and delete companies. Also a simple statistic of phishing campaigns per company was implemented. Rest of the week was used to clean up the code and fix minor bugs. E.g. a loading icon was implemented when waiting for that data to be fetched from the server.

3.6. Product specification

The first version of the requirement specification was made after some initial meetings with CYBR. In these meetings we focused on getting a clear understanding of the product they wanted us to make and how it should work. After these meetings we made a sketch of the requirements and sent it to CYBR to see if we had understood them correctly.

The specification has changed a bit during the work. Changes have been made if problems we hadn't thought about have occurred or if we have gotten new ideas to make the final product better. We've had meetings with CYBR where we have discussed these things and then found a solution that works for both parts.

Some of the requirements from the requirement specification have been made more detailed later in the working process. This isn't surprising, there will always be problems that can't be identified before they actually occur and when you are working on a project you will find new possible solutions that can make the final product even better. One example is the risk score. At first, we just planned to have a risk score, however, after working with it the risk score system is far more detailed, see section 4.5.6 Risk Score for more about the risk score.

The requirement specification describes a product that is pretty similar to the one we are making. The main difference is that this mostly specifies the main features of the program, not the smaller ones. Most of these have been identified later in the working process after the requirement specification was made.

During a development process there will always occur unexpected problems and you will find new and better solutions than what you initially planned. This has happened in the working process as well. The final product has been changed a bit since the first meetings with CYBR. We think these changes have made the product even better and that it will give CYBR a program they can use for the purposes that they want. And these changes are also discussed with CYBR so they have agreed on them as well.

3.6.1. Frontend

Making a frontend was not a priority for both the team and CYBR. The team wanted something to display and showcase core endpoints that the REST API potentially could do.

3 months into the project we felt confident that we would finish all other work we planned with CYBR so we started working on the frontend. We allocated one member of the team to spend around one month creating a simple GUI. The goal what to showcase a few critical functions of the product including:

- Create a campaign.
- Show all active and finished campaigns.
- List all employees and add employees to a company.
- Show results of phishing campaigns.
- List all phishing attacks.

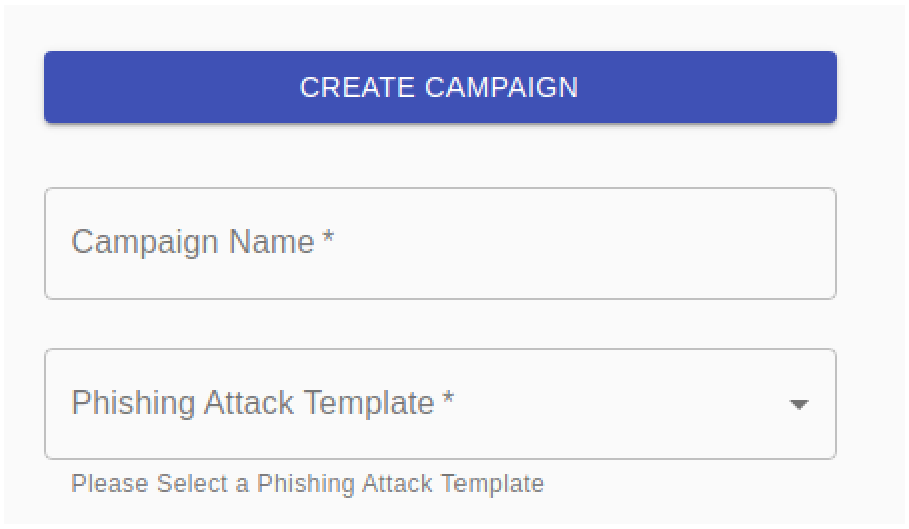

CREATE CAMPAIGN
Campaign Name *
Phishing Attack Template *
Please Select a Phishing Attack Template

Figure 12. Input fields for creating a campaign.

The frontend was kept simple and the only goal was to showcase functinaly. UI and UX was not a priority as it was a proof of concept and never would make it into production. Styling and display of potential error messages is also limited. The core product is the REST API, the frontend is just supplemental.

3.7. Research of phishing attacks

We have done research to help understand what phishing is and how it can affect organizations. This has been very helpful for us when making the program. We have researched how phishing attacks are performed, how many attacks there are yearly and how expensive it can be if you get phished. As well as how organizations can work to prevent themself from getting phished.

3.7.1. Security awareness training

The results of Terranova's 2020 Phishing report (Terranova, 2020) showed that around 20% of all employees are likely to click on a link in a phishing email. And even more worrying, from these 20% are 67,5% likely to give away their details.

Another study, done by proofpoint found that 95% of all organizations participating in the study were saying that they were performing phishing awareness training on their employees. Still phishing is what is most likely to cause a data breach for a company. (Proofpoint, 2020) This is not weird as the biggest attack surface a big company has is their employees. This shows how important phishing awareness training is.

Research from proofpoint (Proofpoint, 2020) showed that 75% of all organizations around the world were subject to a phishing attack in 2020. It also showed that 74% of all the attacks on American businesses were successful. Therefore companies and organizations should always work on getting better to prevent phishing. Two ways to improve a company's ability to detect phishing and other types of cybercrime is:

- **Monitor emails:** Software that monitors all emails sent to employees and scans them for malicious content. If any type of scam is detected, the mail gets blocked, and the receiver never gets the mail. Devices that help doing this are usually good at blocking traditional spam emails and regular phishing. However, they are struggling more with spear phishing attacks as these usually look like normal emails.
- **Security awareness training:** Training the employees to detect phishing emails and other cyber attacks. The training should first lecture the employees about what phishing is and give some examples of phishing mails and tell what they should look out for. E.g., bad language or requests of money transfers. Then emulated phishing attacks should be sent to the employees, this is done to check who of the employees that manages to detect phishing and who not.

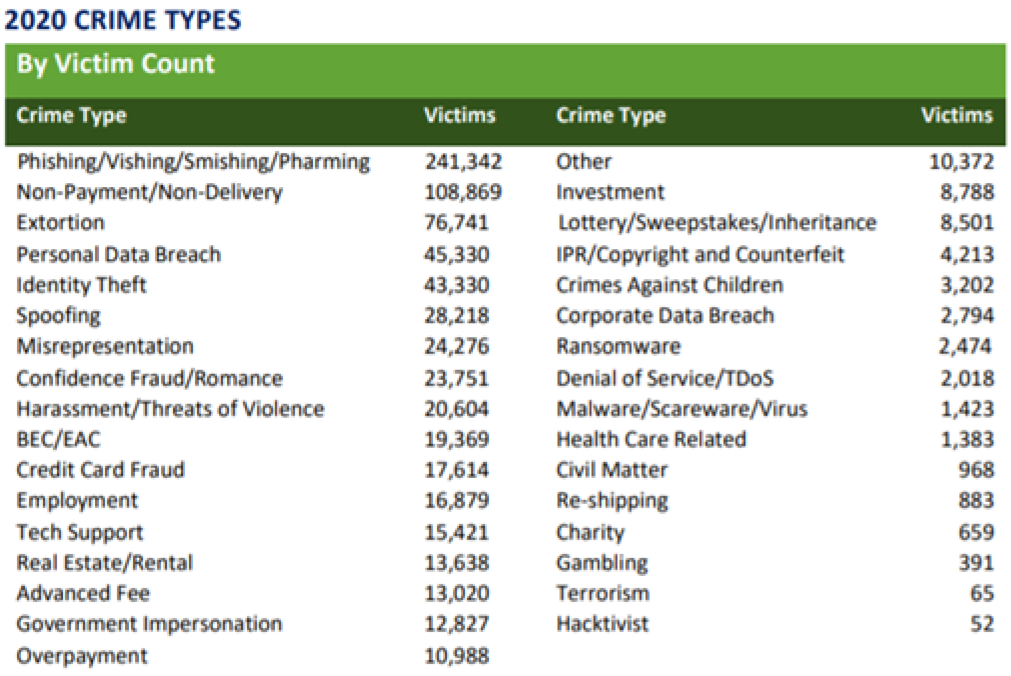
Successful security awareness training should make the employee feel more comfortable dealing with phishing. And after completing cyber security training employees should be better able to detect which emails are scam and then report them to the IT staff. The more emails the employee's reports, the better it is. It doesn't hurt anyone if they report an email that is not a phishing mail, but it could potentially hurt a lot if they don't report a mail that is a phishing mail.

A study from Cofense (Cofense, 2021) found that employees that were trained in security awareness were far more likely to report a phishing attack than those who had not had this training. And if someone reports the attack quickly, the danger of the attack will stop quickly as well. Because then the rest of the employees can be warned before they do anything.

The study from Cofense also showed that 82% of the employees that were trained reported a simulated phishing attack within 1 hour, 52% within 5 minutes and 19% within 30 seconds. A study from KnowBe4 (KnowBe4, n.d.) showed that an organization who completed one year of phishing awareness training would improve 87% when it comes to detecting phishing attacks.

3.7.2. Phishing Methods

The by far most used phishing method is regular phishing. (Bisson, 2020) According to the FBI there were more than twice as many regular phishing attacks in 2020 than any other cybercrime. (FBI, 2021) The attackers mass send emails pretending that they are from a social media or another big organization and try to steal personal information, log in details or money from their victims.



2020 CRIME TYPES

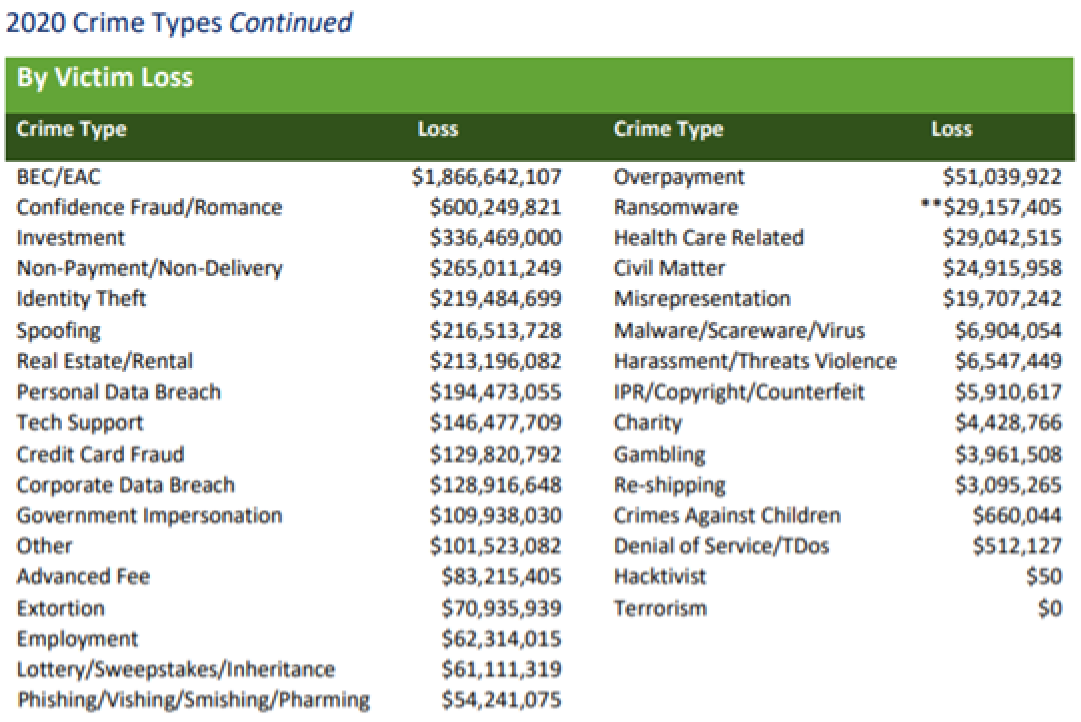
By Victim Count

Crime Type	Victims	Crime Type	Victims
Phishing/Vishing/Smishing/Pharming	241,342	Other	10,372
Non-Payment/Non-Delivery	108,869	Investment	8,788
Extortion	76,741	Lottery/Sweepstakes/Inheritance	8,501
Personal Data Breach	45,330	IPR/Copyright and Counterfeit	4,213
Identity Theft	43,330	Crimes Against Children	3,202
Spoofing	28,218	Corporate Data Breach	2,794
Misrepresentation	24,276	Ransomware	2,474
Confidence Fraud/Romance	23,751	Denial of Service/TDoS	2,018
Harassment/Threats of Violence	20,604	Malware/Scareware/Virus	1,423
BEC/EAC	19,369	Health Care Related	1,383
Credit Card Fraud	17,614	Civil Matter	968
Employment	16,879	Re-shipping	883
Tech Support	15,421	Charity	659
Real Estate/Rental	13,638	Gambling	391
Advanced Fee	13,020	Terrorism	65
Government Impersonation	12,827	Hacktivist	52
Overpayment	10,988		

Figure 13. Crime Types by Victim Count (FBI, 2021).

However, the type of phishing that is most costly for companies is spear phishing. This type of phishing can also be called Business Email Compromise (BEC). A typical BEC scam is invoice fraud. Here the attacker pretends to be someone else that needs payment from the victim company. This could for example be a supplier or a construction company. To make a trustworthy attack like this the attacker needs to do some research first. One successful spear phishing attack does on average result in a loss of 1,6 million dollars for a company. And in 2020 FBIs Internet Crime Report reported a total of more than 1,8 billion dollars lost in BEC attacks.

3.7.3. Costs of a phishing attack



2020 Crime Types *Continued*

By Victim Loss

Crime Type	Loss	Crime Type	Loss
BEC/EAC	$1,866,642,107	Overpayment	$51,039,922
Confidence Fraud/Romance	$600,249,821	Ransomware	**$29,157,405
Investment	$336,469,000	Health Care Related	$29,042,515
Non-Payment/Non-Delivery	$265,011,249	Civil Matter	$24,915,958
Identity Theft	$219,484,699	Misrepresentation	$19,707,242
Spoofing	$216,513,728	Malware/Scareware/Virus	$6,904,054
Real Estate/Rental	$213,196,082	Harassment/Threats Violence	$6,547,449
Personal Data Breach	$194,473,055	IPR/Copyright/Counterfeit	$5,910,617
Tech Support	$146,477,709	Charity	$4,428,766
Credit Card Fraud	$129,820,792	Gambling	$3,961,508
Corporate Data Breach	$128,916,648	Re-shipping	$3,095,265
Government Impersonation	$109,938,030	Crimes Against Children	$660,044
Other	$101,523,082	Denial of Service/TDos	$512,127
Advanced Fee	$83,215,405	Hacktivist	$50
Extortion	$70,935,939	Terrorism	$0
Employment	$62,314,015		
Lottery/Sweepstakes/Inheritance	$61,111,319		
Phishing/Vishing/Smishing/Pharming	$54,241,075		

Figure 14. Crime types by Victim Loss (FBI, 2021).

FBI also estimates that US companies have lost around 12 000 000 000 dollars in total from all types of phishing attacks the last 5 years. (Daly, 2019)
Research from BDO found that around 60% of mid-sized businesses in the UK were hit by fraud in 2020, and that the average cost of this was 245 000 pounds. (BDO, 2021)

3.7.4. How a phishing attack works

An attacker usually exploits trends when he makes his attacks. This is done because it is a theme that many are interested in and will therefore click the link. Most people will probably not click on a link to a service they don't use. Last year many attackers exploited the COVID-19 pandemic when making their attacks. FBIs internet crime report for 2020 reports that they received over 28 500 complaints in 2020 related to COVID-19. (FBI, 2021)

The fact that many people have been working remotely during the pandemic have also been exploited heavily by attackers. A report made by Microsoft says that 80% of security professionals have seen an increase in security threats since shifting to remote work. (Microsoft, 2021) And of these 80%, 66% of them have said that phishing campaigns have increased the most. Typical attacks that have been made because of remote work are attacks where the attacker pretends to be from services like Zoom and Microsoft teams as these applications are used a lot for people working remotely.

3.8. Results from the research

Before starting with the development, we needed some solid research to get a clear understanding of what phishing is, and on what impact it has on organizations today. In the first part of the research we looked at the most successful phishing attacks in recent times. We wanted to put attention on the methods used in these successful attacks.

3.8.1. Phishing technique researching

The technique section is the most interesting part as it highlights the various possibilities for gaining access to a system, which is useful when developing templates and landing pages.
In this part of the research we discovered that one type of phishing attack was more prominent than others, spear phishing. Here the attackers target a few select people within the company, and in doing so the phisher can make more tailored methods of tricking the recipients. We’ve read in section 2.2.2. Spear phishing attack that spear phishing attacks were successfully performed in such a way where attackers have impersonated a leader within the company and sent out emails to their subordinates, giving them instructions with pure malicious intent. These instructions would range from sending payments to various fake accounts, changing passwords on critical systems or making employees change their passwords and in the process give out their login credentials.

3.8.2. Making use of the research

Later it was important to use the data gathered to make AttackTemplates. We wanted attacks that would be oriented around the business setting, therefore research was done on what type of services businesses used on an everyday basis. We found out that most businesses use the same services such as office 365 and Zoom, and that there is not that much variation in what's used. After understanding this we could focus on developing the necessary templates.

3.9. Conclusion and reflections

3.9.1. Reflections from the team

In the start of the project the learning curve was steep. The team had a general programming background, but did not have much experience in some of the chosen tools and framework. The team wanted to work with well established tools that could serve us in future endeavours.

CYBR gave us a lot of freedom on the technical side. So we could choose technologies the team wanted to work with. This worked out great and the team learned a lot. The team is happy to have chosen well known and documented frameworks like Django and React. The product documentations of the chosen frameworks become extremely valuable and it helped the team overcome any obstacles they met during the project phase. The Google Cloud VM instance provided by the client helped the team learn about working with a cloud setup. It was both fun and challenging and the team learned a lot from this experience.

3.9.2. Challenges

Working together was a bit different this year, since the team and CYBR could never meet in person. The team having three core workdays between 09.00 - 15.00 starting with a daily stand-up 09.00 was crucial in this. This gave the team a feeling of working together as a unit. Everyone was available on slack in the given timeframe and that made it easy and convenient to reach other group members.

A different challenge was knowing exactly what the client wanted. The team did not have much experience with the topic of phishing. So in the start of the project the team spent much time learning about phishing. Then the team got a clearer understanding of the goal of the client. The task was also very open, both in a technical sense and features of the product. This was both challenging and at the same time fun figuring out how the product should be designed from scratch.

3.9.3. What we could have done differently

Towards the end of the project the team realised that they could have made up some fake scenarios to test the program. These scenarios would be of fake companies with different departments and employees, the plan was that these companies would act as testing ground for the program. The team would then make custom templates and landing pages to essentially attack these fake companies. This could give a proper look into how the program performed and if it would work in a semi real setting.

3.9.4. Conclusion

The team is happy to have produced a proof of concept REST api that fulfills most of the initial requirements from the client and we've also added some additional features. CYBR have said that they potentially want to use parts of the product in production or take inspiration from it. Whether the product is being used or not, the team has done its best to make it as easy as possible to potentially take over the project for anybody interested.

4. Product Documentation

4.1. Preface

The product documentation explains how the components of the product work. It is a combination of three different applications excluding the database. The main part of the project is the Django backend which handles all data analysis and company management. It is wrapped around the Gophish application which provides functionality for sending phishing emails. A simple frontend uses the backend API to show statistics and demonstrate the potential of the program.

It is recommended to read the 1. Introduction and presentation before reading the product documentation. The documentation assumes that the reader has technical experience with programming languages, and a decent understanding of APIs.

After reading the product documentation the reader should inherit a clear understanding of how the program works, and essential features of the program. Detailed information about all of the endpoints and their functionality is described in 5.3. Attachment 3: API Documentation. Together with the 5.2. Attachment 2: User guide this should give a good foundation to install, maintain and operate the product.

4.2. Features of the product

The product backend is a phishing emulation tool that can be used to improve phishing awareness. It supports two different user types. An admin for CYBR and each company will have a separate “normal” account that can perform phishing attacks and view statistics of phishing campaigns.

4.2.1. As an Admin user

The admin user can add, edit and remove companies to the tool.

The admin user will also be used to maintain phishing email templates and landing pages. If there are any new templates the admin user can add it to all current companies and modify templates if needed. An admin user will be able to extract info and statistics about how all companies performed in total. See Figure 15. Use case of admin functionality.

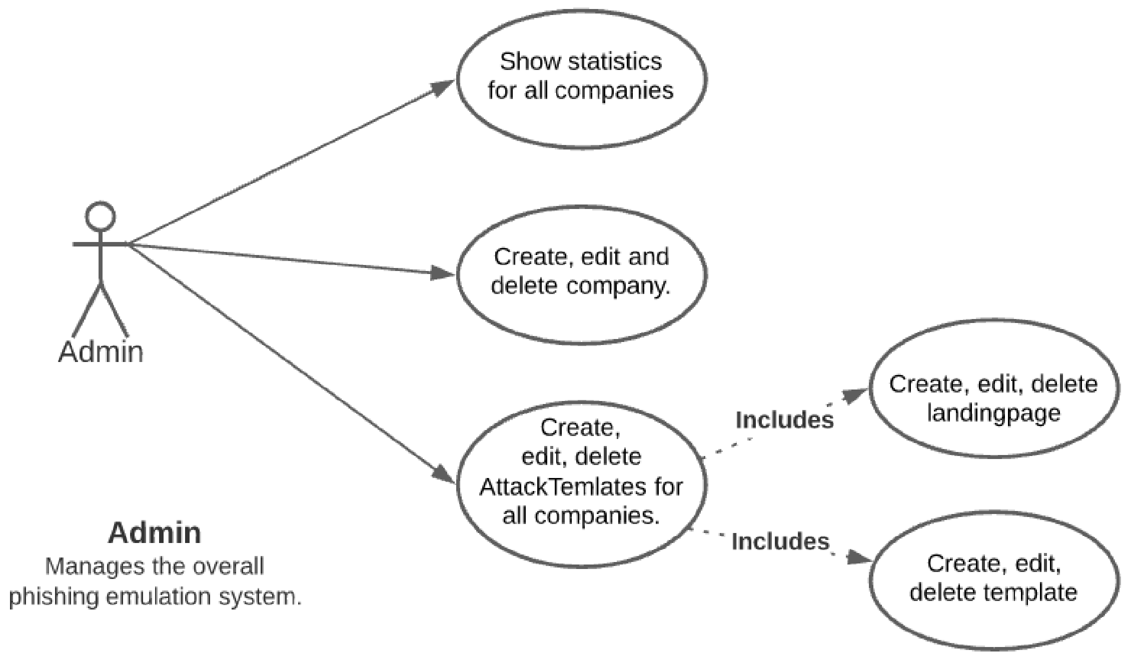

Show statistics for all companies
Create, edit and delete company.
Admin
Create, edit, delete landingpage
Includes
Create, edit, delete AttackTemlates for all companies.
Includes
Create, edit, delete template
Admin
Manages the overall phishing emulation system.

Figure 15. Use case of admin functionality.

4.2.2. As a normal company user

A normal user from a company can start phishing campaigns, show all active campaigns and individual stats from each campaign. The product supports adding new employees, see a total overview of how all employees performed and more detailed statistics for each employee.

Each employee will be given a risk level based on their position and access to sensitive data. The risk level will be used to calculate a risk score.

The tool comes with a pool of different phishing attack templates to choose from. They have been given a risk score based on their difficulty. “A phishing attack” contains an email template and a landing page. A deep dive on how this works can be found at 4.5.3. Webhooks.

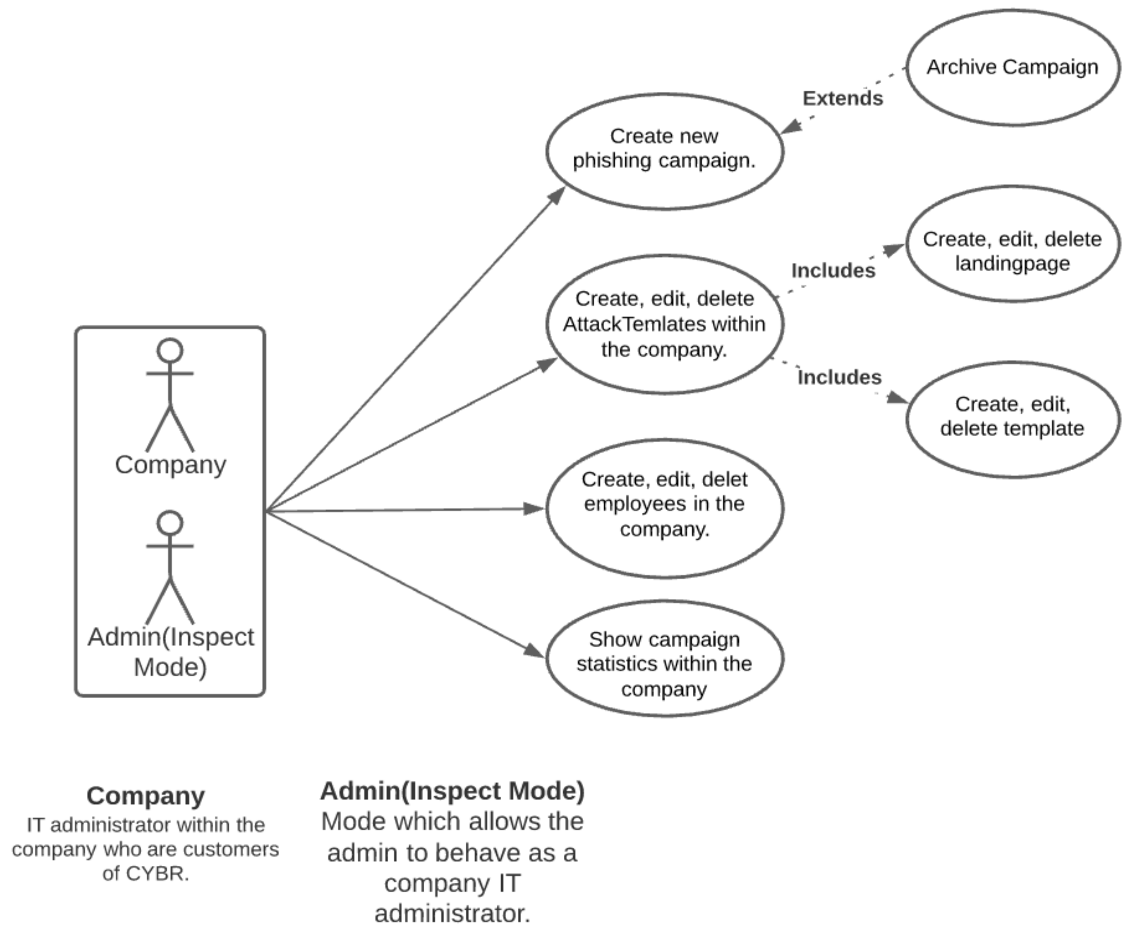
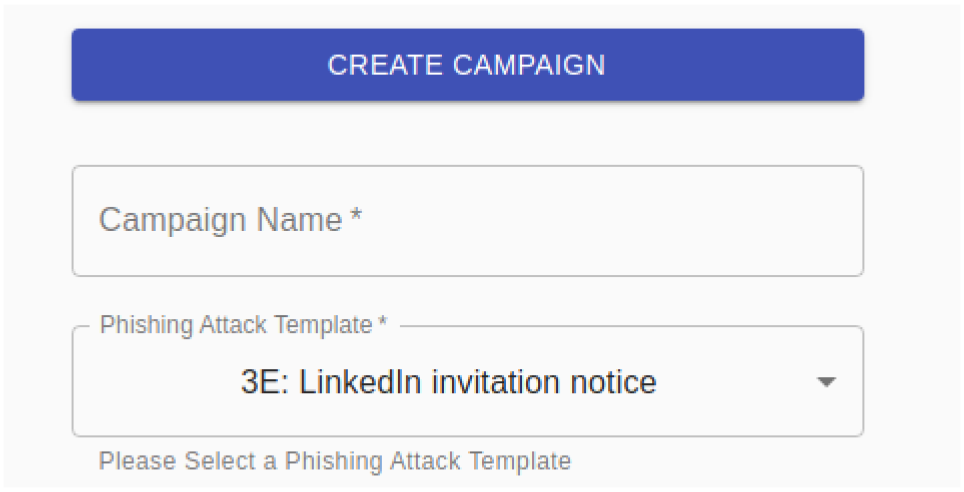

Archive Campaign
Extends
Create new phishing campaign.
Includes
Create, edit, delete landingpage
Create, edit, delete AttackTemlates within the company.
Includes
Create, edit, delete template
Company
Admin(Inspect Mode)
Create, edit, delet employees in the company.
Show campaign statistics within the company
Company
IT administrator within the company who are customers of CYBR.
Admin(Inspect Mode)
Mode which allows the admin to behave as a company IT administrator.

Figure 16. Use case for company and Admin(Inspect Mode)

4.2.3. Creating a phishing campaign

When creating a phishing campaign, the user has to select a descriptive name, the targets who require phishing awareness training, and the desired attack template. One campaign sends one email to all the selected employees in the company.



Phishing Attack Template *: 3E: LinkedIn invitation notice

Please Select a Phishing Attack Template

Figure 17. Illustration of how campaigns are created in the Frontend.

The campaign tracks info about the target's actions. How many:

- Emails received.
- Emails opened.
- Links clicked inside the email.
- Credentials submitted on the landing page.
- Emails the target reported.
- Attachments downloaded.

Based on the target's action they will be placed in different "risk categories". Low, medium, and high risk. This information can be used to inform the company about the recommended action to prevent phishing attacks. If the employee is low risk, more realistic phishing attacks can be selected. If the employee is marked as high risk, it's recommended that the employee takes part in cyber security training. In Figure 18. Use case for employees the different action for an employee is displayed. The actions are tracked in the backend.

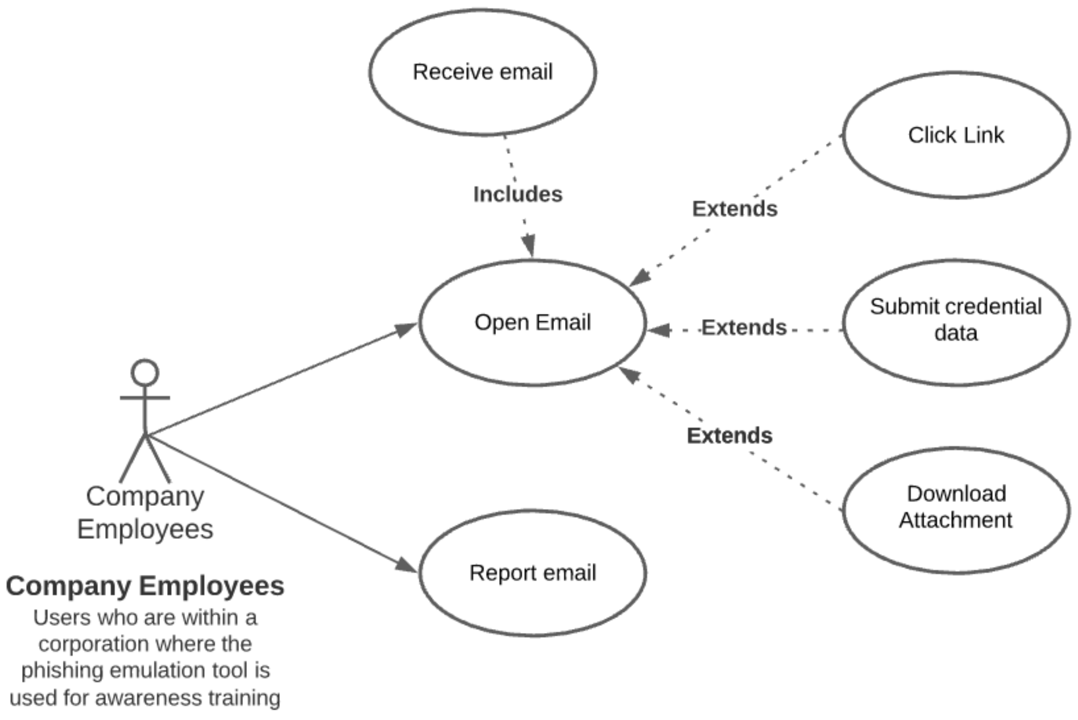


Figure 18. Use case for employees.

Every action is stored in a timeline and the company can get more detailed information about the action performed by the target. More info can be found in 4.5.1.2.7 Timeline.

4.3. Main program and its components

In this chapter the most essential components of the main program will be described. It will be shown which of the applications execute which parts of the code, and where the data is processed and stored. Hopefully, this will clarify how the frontend, backend and Gophish communicate and share data.

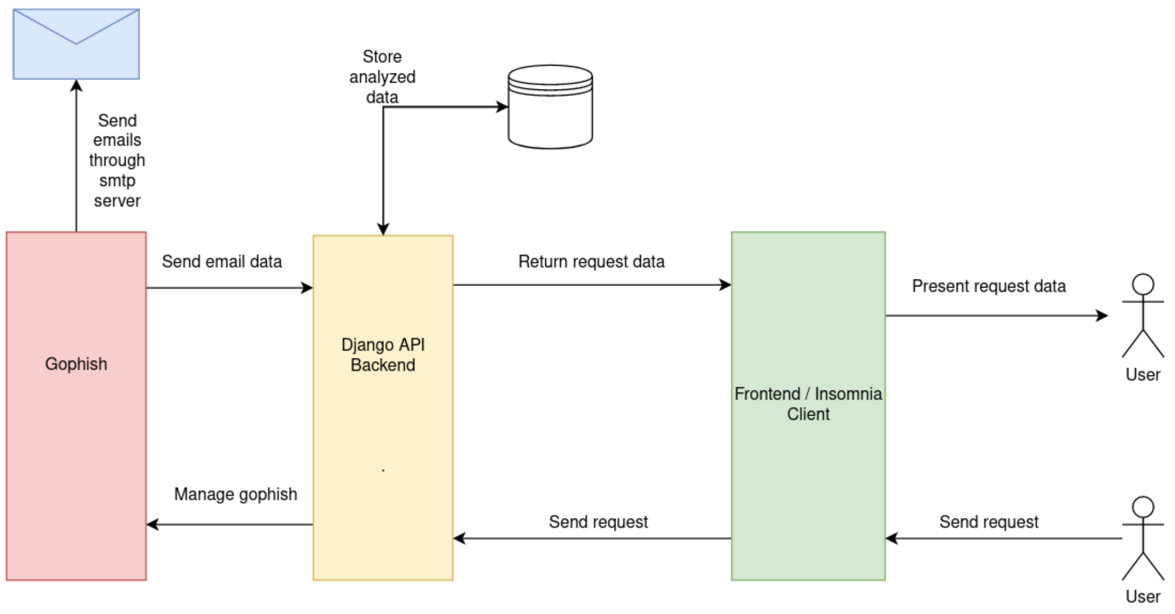


Figure 19. High level overview of how the different servers communicate.

Figure 19 gives a general overview of the flow of the finished product. In reality the communication process is more complex and varies by the requested action sent by the user. Retrieving objects for all the scenarios are always retrieved in reverse of what is shown. E.g. Landingpage requests are forwarded to the Gophish storage, and returned from there. Campaign data / results, employees, and campaign details are retrieved from the backend. Templates are retrieved from Gophish, and configured to AttackTemplates in the backend. See 4.5.1.2.6. Template & AttackTemplate for more details on how they work.

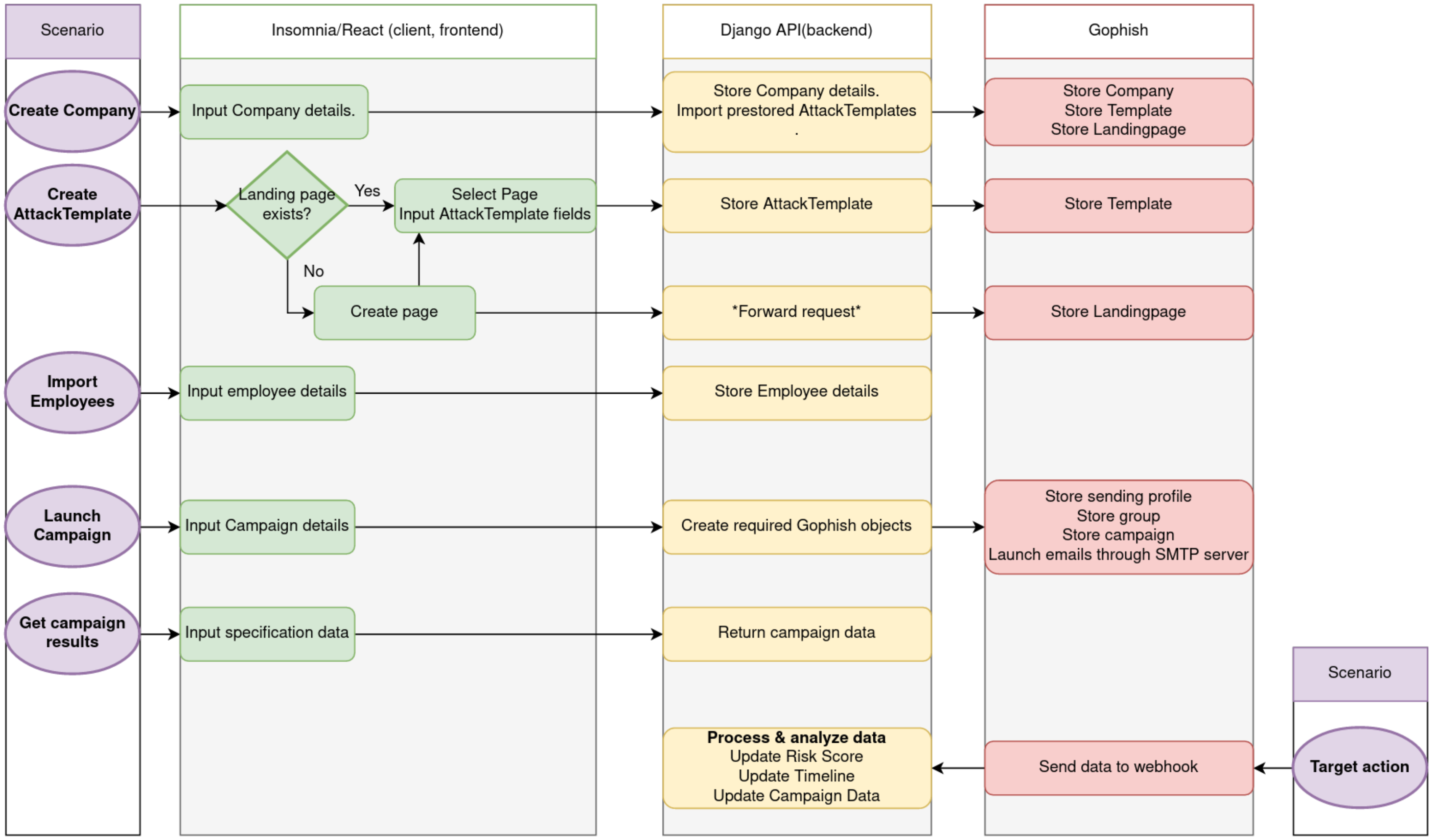


Figure 20. Overview of how the actions are processed by the different applications for different scenarios.

In Figure 20 different scenarios and how they're passed through the applications is shown. A brief introduction to how each of the scenarios work will be described in this section. Specific data for the referenced endpoints can be found in 5.3. Attachment 3:API Documentation.

4.3.1. Create Company

When creating a company the required data for the CO1 endpoint has to be input by the client. This data is then sent to the backend which stores SMTP and company details. The pre-import process described in 4.5.7. Import AttackTemplates on Company creation is also started and AttackTemplate data is stored in the backend while the html and css data of the template and landingpage data is sent to Gophish. A matched company by ID is created in Gophish to map the templates correctly.

4.3.2. Create AttackTemplate

Creating an AttackTemplate with the AT1 endpoint. If the wanted landingpage does not exist this has to be created first using the LP1 endpoint. The AttackTemplate risk calculation data is stored in the backend while the css & html of the template itself is forwarded to Gophish.

4.3.3. Import Employees

Importing employees with the EM1 endpoint stores targets and their details in the backend.

4.3.4. Launch Campaign

Launching a campaign using the CA1 endpoint does not require much input. The process starts a sequence of objects created in Gophish which are deleted when the campaign is archived. More about this process can be found in 4.5.2. Campaign initialization. After the objects are created the campaign is launched.

4.3.5. Get campaign results.

Campaign data and timelines are stored in the backend. There are 14 different endpoints for returning different statistics described in the RE and CA sections in the 5.3. Attachment 3:API Documentation.

4.3.6. Target action

When Gophish registers an action performed by one of the targets (employees) the data is sent to the backend using the webhook. From that data risk score and campaign results are processed. More info about the webhook in 4.5.3. Webhooks and how the risk score is calculated in 4.5.6. Risk score.

4.4. The structure and behaviour of the program

This section will give a short introduction to django and its use in the product. Django uses five main components when providing data to the end user. These five components are views, urls, serializers, models and settings.

4.4.1. Views

Views are python classes or functions which take web requests and return web responses. This is used in the backend for sending modified data to the external Gophish server, and also for fetching and editing data received from it. See Table 11.

```
#Example of a method which will act as a view file. This method filter employes according their department
@api_view(['GET'])
def stats(request, department):
   if request.method == 'GET':
       queryset = Employee.objects.filter(department__contains=department,
department__iexact=department)
       serializer = Serializer(queryset, many=True)
```

```
        return Response(serializer.data, status=status.HTTP_200_OK)
```

Table 11. Example of a view function with @api_view(), which is a part of function based views from rest_framework.

Function based views are used throughout the program. With function based views simple decorators can be wrapped around views. These decorators remove the standard `http request` within django and replaces it with a tag (`@api_view(['GET'])`) on top of the function. This simplifies the work process and also makes the code more readable.

4.4.2. Serializer

Serializers (django-rest-framework.org, n.d.) takes complex data such as querysets or models and turns them into native Python data types as shown in Table 12. This includes deserializers which take the python data types and turn them back into complex data. The serializers interact with the models and give the possibility to create stored data.

```
class EmployeeSerializer(serializers.ModelSerializer):
   class Meta:
       model = Employee
       fields = ('first_name', 'last_name', 'position',
'department','risk_level')
```

Table 12. Example of a typical serializer function

4.4.3. Models

The models are the single definitive source of information and have direct correlation with the database. Models are used for making custom single tables in the database which can easily be modified and accessed in django. The models are strongly related to the architecture of object oriented programming and essentially are class functions. These classes can easily be modified and make database management easier.

```
class Company(models.Model):
   id = models.AutoField(primary_key=True)
```

```
    api_key = models.CharField(max_length=255, null=True, blank=True)
    gophish_id = models.CharField(max_length=255, blank=True, null=True)
    gophish_sending_profile_id= models.IntegerField(default=0)
```

Table 13. Example of a model used in django

4.4.4. Url

Urls are used in django to create paths for accessing the functions in the views. Urls build the foundation of the API.

```
path('templates/', views.templates, name='templates'),
path('import/email', views.importTemplate, name='importTemplate'),
re_path(r'^templates/(?P<template_id>\d+)$',
views.templateId,name='templatesId'),
```

Table 14. Example on how the urls for the api look.

4.4.5. Settings.py

The settings.py is a configuration file which tells what is allowed within the application. It is where frameworks and other dependencies are confirmed and made accessible to be used in the code, see Table 15.

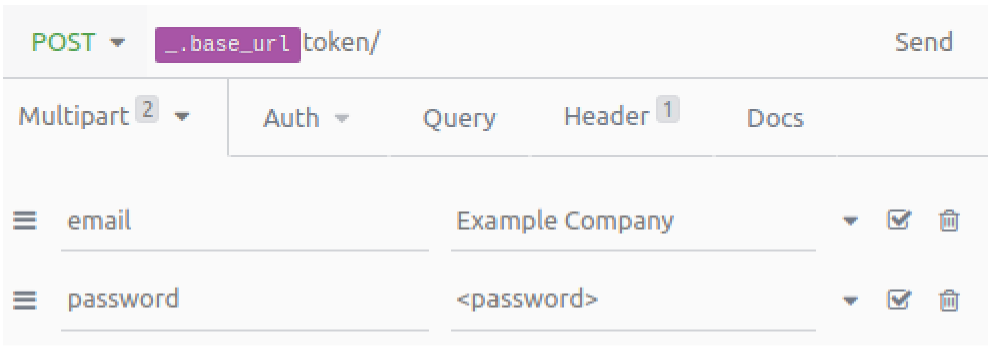
```
INSTALLED_APPS = [
   'rest_framework',
   'rest_framework.authtoken',
   'api.apps.ApiConfig',
   'django.contrib.admin',
   'django.contrib.auth',]
```

Table 15. A configuration setup from settings.py showcasing applications used by the code.

4.4.6. Authentication with JWT

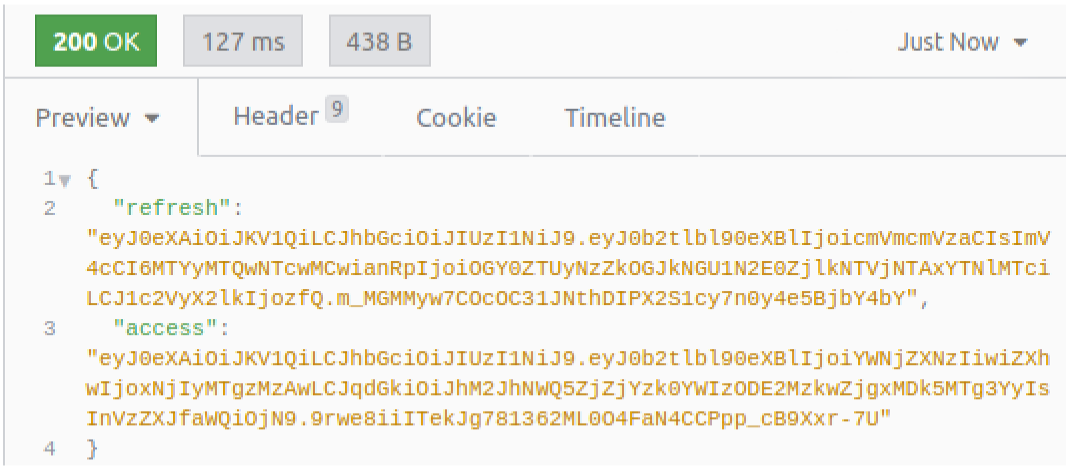
For authentication JSON web tokens are used. Simple JWT (Sanders, 2021) is an authentication plugin for DRF. It comes with endpoints used to create and refresh tokens.

These endpoints are implemented in the Django project and can be used by sending requests via curl, a frontend, or a tool like Insomnia. An example of how this can be done is shown in Figure 21, Figure 22 and Figure 23.



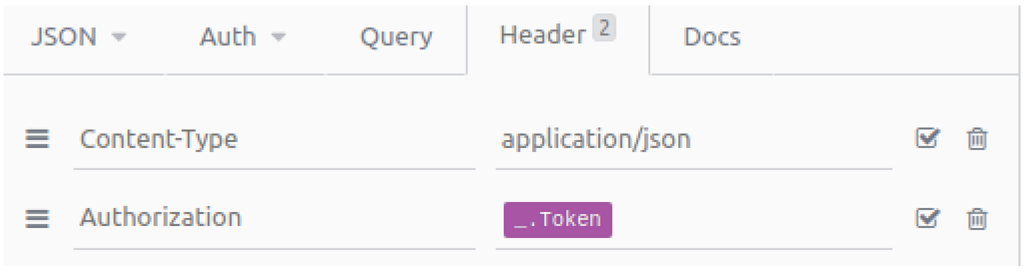
email	Example Company
password	<password>

Figure 21. Input email and password into the request.



```
{
  "refresh":
"eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ0b2tlbl90eXBlIjoicmVmcmVzaCIsImV
4cCI6MTYyMTQwNTcwMCwianRpIjoiOGY0ZTUyNzZkOGJkNGU1N2E0ZjlkNTVjNTAxYTNlMTci
LCJ1c2VyX2lkIjozfQ.m_MGMMyw7COcOC31JNthDIPX2S1cy7n0y4e5BjbY4bY",
  "access":
"eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ0b2tlbl90eXBlIjoiYWNjZXNzIiwiZXh
wIjoxNjIyMTgzMzAwLCJqdGkiOiJhM2JhNWQ5ZjZjYzk0YWIzODE2MzkwZjgxMDk5MTg3YyIs
InVzZXJfaWQiOjN9.9rwe8iiITekJg781362ML0O4FaN4CCPpp_cB9Xxr-7U"
}
```

Figure 22. Response with a refresh and access token



Content-Type	application/json
Authorization	_.Token

Figure 23. Then the access token can be put into the header of the request. The token variable is the access token shown over in the format "JWT <access token>".

The desired settings for the django project are configured in the settings.py. The access token has a lifetime of five minutes. This is to prevent unauthorized authentication in case JWT gets compromised. Tokens also get rotated and blacklisted tokens after use.

Simple JWT comes with an application that provides blacklist functionality. The blacklist app stores outstanding and blacklisted tokens in the models: *OutStandingToken* and *BlacklistedToken* (Sanders, 2020). Then when using the endpoint, the outstanding and blacklisted tokens are stored in these objects.

The JWT is bound to the Django user and is used to separate the data from the different companies and the admin user.

4.4.7. Company vs Admin authentication

In the *settings.py* file the global default permission is set to *IsAuthenticated* (See Table 16). That means anyone using the endpoint has to have a valid access token. This will make it so that the API is only accessible to registered users. All companies are normal users and get access through their django user.

```
REST_FRAMEWORK = {
    'DEFAULT_AUTHENTICATION_CLASSES': (
      'rest_framework_simplejwt.authentication.JWTAuthentication',
    ),
    'DEFAULT_PERMISSION_CLASSES': (
        'rest_framework.permissions.IsAuthenticated', )
}
```

Table 16. Global authentication settings

For the admin endpoint the authentication is set on a per-view basis. By using a decorator as shown in Table 17. *IsAdminUser* sets the permissions class so that only users that are marked as a staff user have access to the endpoint

```
# example of a view only accessible for the administrator
@permission_classes([IsAdminUser])
def company(request):
  #function
```

Table 17. Example of "IsAdminUser" decorator.

4.4.8. Login for authentication

The login endpoint can be reached without any form of authentication to allow users to authenticate themself to the system with a username and password. An example of how this works can be seen in Table 18. The decorator *@permissions_classes((AllowAny,))* is used to set the endpoint open for everyone. This will check if the user trying to connect is registered in the system. In a potential frontend the access and refresh token needs to be stored and managed. Please refer to the Simple JWT for a full explanation.

```
@api_view(['POST'])
@permission_classes((AllowAny,))
def loginAPI(request):
# Get username and password from request.
username = request.data.get("username")
password = request.data.get("password")
# Returns an error message if username or password is missing.
if username is None or password is None:
   return Response({'error': 'Please enter username & password!'},
status=status.HTTP_400_BAD_REQUEST)

# Tries to authenticate the given username and password.
user = authenticate(email=username, password=password)
if not user:
   return Response({'error': 'Please enter Valid Credentials!'},
status=status.HTTP_404_NOT_FOUND)

return Response({ "msg": "Logged in"}, status=status.HTTP_200_OK)
```

Table 18. Code of login endpoint.

4.4.9. Log files

The backend logs keeps a log file for debug purposes if issues occur.

This can be found in the root folder of the project and is named *logs.txt*.

```
2021-05-05_14-59-36: Error:400 - Employee not existing or not
participating in campaign
2021-05-05_15-00-16: Error:404 - Campaign does not exist
2021-05-05_15-02-48: Error:404 - Employee with: 10 ID does not exist!
2021-05-05_15-03-58: Error:403 - You're not authorized to access this
object.
2021-05-05_15-04-11: Error:404 - Campaign matching query does not
exist.
2021-05-05_15-09-52: Error:500 - Phishing server error
```

Table 19. A few log file examples.

4.5. Central data structures

In this section the core parts of the API are demonstrated. Code snippets with description are used to simplify the more advanced sections. It is written for developers who intend to take over the development and expand the program.

4.5.1. Objects & Models

4.5.1.1. High Level Architecture

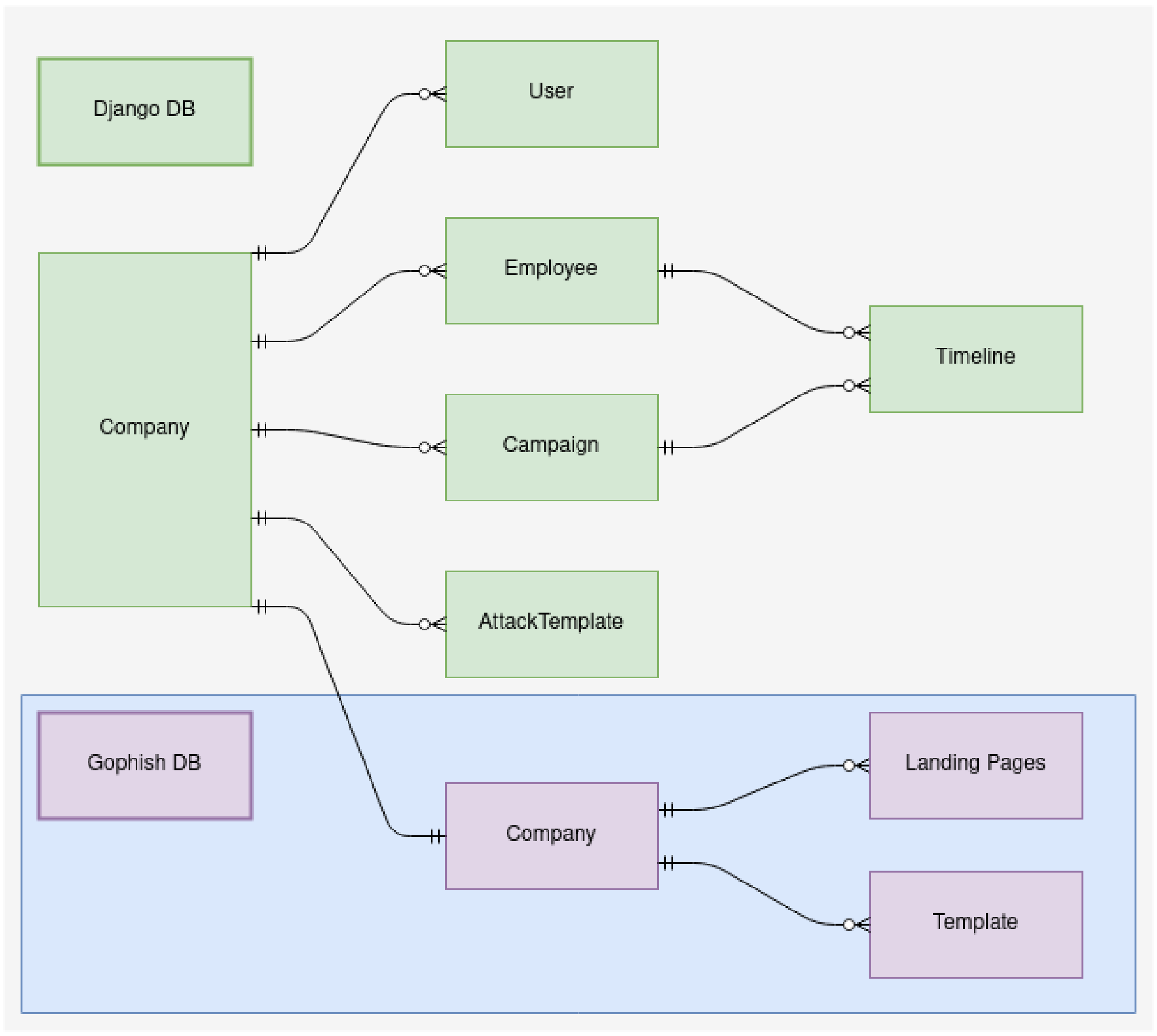


Figure 24. High level architecture design.

4.5.1.2. Model explanation

4.5.1.2.1. User

Objects for the customers who want to use the Gophish API. They authenticate to the API, retrieve a JSON Web Token and are able to use the API calls to create attacks. The data stored in the API for the User object can be seen in Table 20.

```
email       # email/username for authenticating
```

```
password    # password for authenticating
company     # company relationship.
api_key     # Gophish api key. only used by Admin user. Non-administrators
retrieve Gophish api_key from their company relationship.
```

Table 20. Field explanation for the User object.

4.5.1.2.2. Employee

Employees within the company. These have various fields to map their risk score, risk level, department and data such as email, first name, and last name as shown in Table 21. This data is used to create targeted attacks by levels and departments. The risk scores are updated after each campaign. More info about risk score calculation 4.5.6. Risk score.

```
id            # employee id (auto).
company             # relationship to Company.
risk_level          # how valuable this target is. Score modifier.
department          # what department does the employee belong to.
email               # email.
first_name    # first name.
last_name           # last name.
position            # position.
# managed automatically below.
risk_category       # low, medium, high category. based on risk_score.
risk_score            # risk score total (debit - credit).
risk_score_credit   # total sum of negative actions in phishing attacks.
risk_score_debit    # total sum of positive actions in phishing attacks.
email_sent                  # stat for total campaigns participated in.
email_opened                # stat for total emails opened.
email_clicked_link          # stat for total links clicked.
email_data_submitted        # stat for credential data submitted.
email_reported              # stat for emails reported.
email_attachment_downloaded # stat for attachments downloaded.
```

Table 21. Field explanation for the Employee object.

This Employee object has a *getRiskMultiplier()* function which is used to retrieve the risk_level of the employee object, and then multiplied with the accumulated points retrieved through actions. More about how this works in 4.5.3 Webhooks.

4.5.1.2.3. Campaign

This object is created when a user schedules or launches a phishing attack.
For an overview over the attack template, employees who are targeted, created date, campaign start time and a reference to the attack template used in the phishing attack, see Table 22.

```
id             # campaign id (auto)
# Required fields.
name           # campaign name
template       # email template used in the attack.
page           # landing page used in attack. Retrieved from template.
employees      # Many to Many relationship to targets.
company        # relationship to Company.
# managed automatically below.
status         # if the campaign is in progress or archived.
created_date   # when the campaign was created.
completed_date # when the campaign was archived.
# Optional Fields - or default value = now.
launch_date    # specify to schedule an attack in the future.
send_by_date   # specify to spread the send time of the emails.
```

Table 22. Field explanation for the Campaign object.

4.5.1.2.4. Company

The multi-tenant system is oriented around the company object. All other objects have a relationship to the company object. This way it is possible to separate campaigns, attack templates, and statistics for each company.

```
id                 # company id.
api_key            # company api key for Gophish.
gophish_id         # company id in Gophish.
```

```
name                # company name.
smtp_host           # smtp host for creating a sending profile.
smtp_username       # smtp username for authenticating to mail server.
smtp_pw             # smtp password for authentication to mail server.
url                 # url for listener.
created_at          # when the company object was created.
updated_at          # when the company object was updated.
```

Table 23. Field explanation for the Company object.

The company object has a one to one relationship with the company object stored in the Gophish database. This is needed to map the template and landing page relationship correctly with the django database.

4.5.1.2.5. Landing Page

The landingpage is the object that contains the html & css where the user is directed to after clicking the link in the email. These are only stored in the Gophish database and are therefore just forwarded and returned to and from the Gophish server.

```
name                  # landing page name
html                  # html for the page.
capture_credentials    # should be false.
capture_passwords      # should be false
redirect_url          # when data is submitted where should the page redirect you
to.
```

Table 24. Field explanation for the Landing Page object.

4.5.1.2.6. Template & AttackTemplate

The attack template is an extension of the template object from Gophish. It has additional data that is mapped to the attack template object. They share IDs which makes it possible to keep track of what extra data belongs to the template.

When the user is requesting a template Django asks Gophish for the template data, finds the corresponding AttackTemplate ID and then appends the extra data stored in AttackTemplate, such as template score. This allows us to map malicious attacks based on research with different scores, and these scores are later used to calculate risk scores.

```
# AttackTemplate
name            # template name
template_score  # score used to calculate employee risk score impact.
landing_page    # what landing page mail link redirects to.
description     # a description of the template & recommended actions.
from_field      # whom the email is from - as shown in inbox.
# Template (Gophish)
subject         # subject - shown in mail inbox.
text            # can be text.
html            # or html content in the email.
modified_date   # when the template was modified.
attachments     # currently not supported by Gophish.
```

Table 25. Field explanation for the Attack Template object.

4.5.1.2.7. Timeline

Timeline objects are created whenever Gophish posts and updates the backend using the webhook. These are objects who keep a record of all actions made by the users within each campaign. It also keeps track of how the employee score was affected for that campaign in this event. The event is also mapped to companies through the employee and campaign relationship.

```
id                     # timeline id.
employee               # employee relationship.
campaign               # campaign relationship.
time                   # event timestamp.
message                # action performed in the event.
details                # info about where the action was performed.
risk_score_increment   # risk score incrementation.
risk_debit_increment   # debit incrementation.
risk_credit_increment  # credit incrementation.
```

Table 26. Field explanation for the Timeline object.

More info about how Timelines are created and how their score is calculated can be found in [4.5.3 Webhooks](#).

4.5.2. Campaign initialization

The team wanted to make it easier for the end customer to create campaigns. A lot of the functionality Gophish requires was therefore implemented into the backend systems. An example of the required actions for each system is shown in Figure 25.

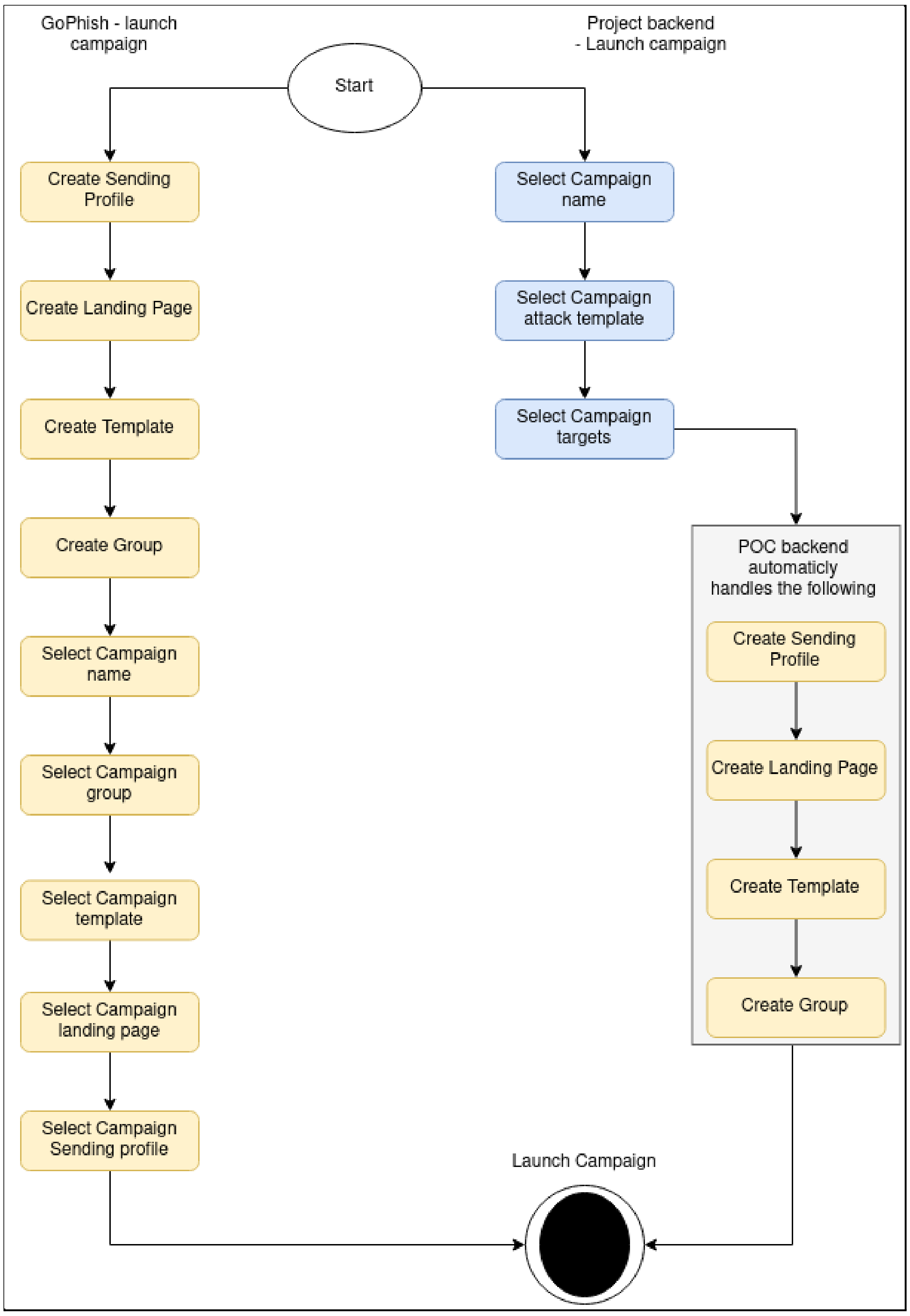


Figure 25. Flow chart - initialize campaign. Gophish vs POC backend.

This is due to required data being stored separately and merged into the required objects by Gophish. They're then sent to Gophish, validated, then the campaign is started with those objects. Figure 26 shows an overview of the fields which are inputted in blue. Grey fields are already stored in the database, the yellow are the resulting objects which are needed to launch the green finalized campaign. The company object is retrieved through the authenticated user.

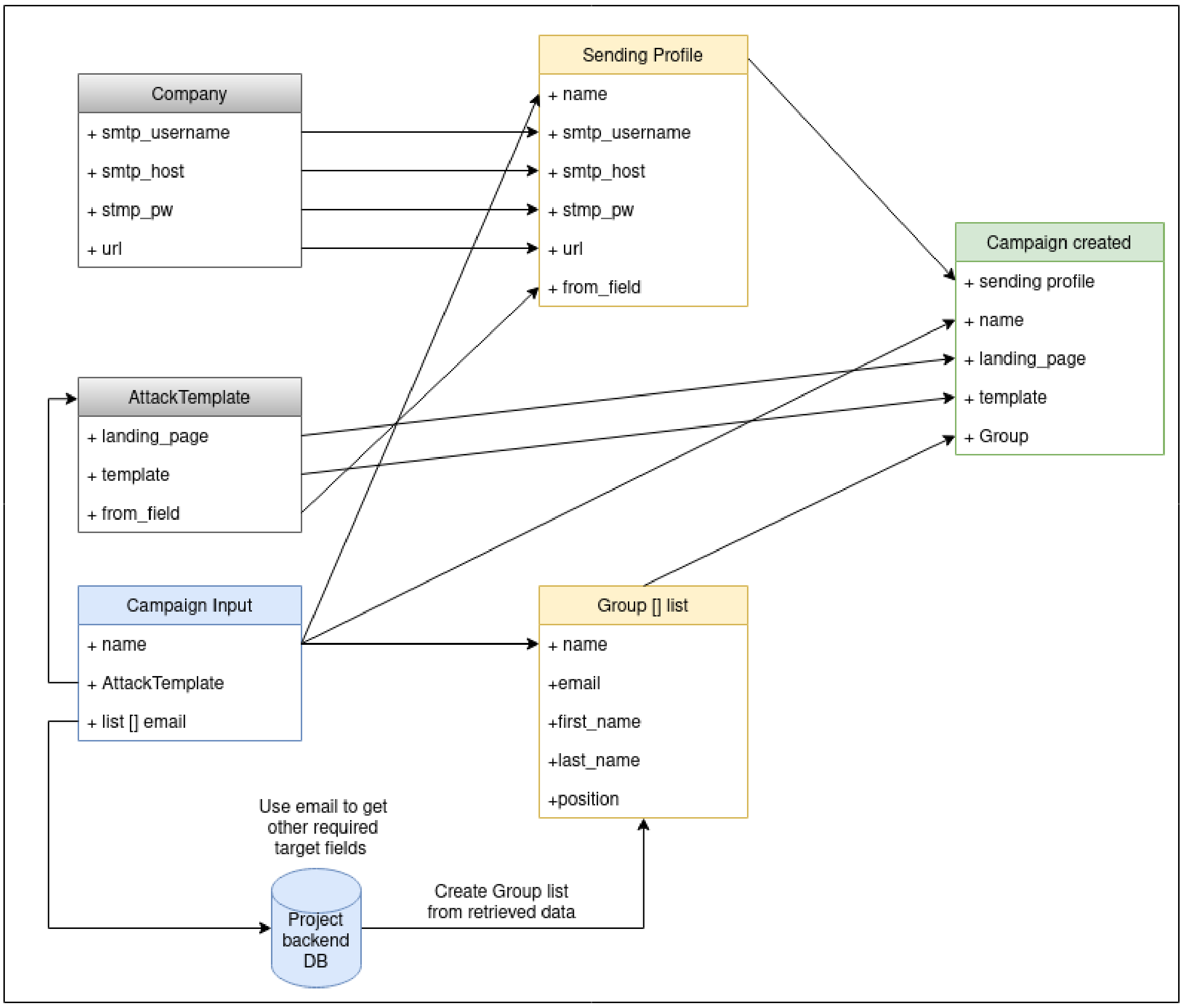


Figure 26. Overview of how various Gophish objects are created when starting a campaign.

Throughout the process any errors are catched and created objects are deleted if anything goes wrong.

4.5.3. Webhooks

4.5.3.1. Supported tracking

In the file */api/views/webhook_timeline.py* the data retrieved from Gophish is processed. Whenever Gophish retrieves an update from the target it is configured to push the update to the api through the endpoint *{URL}/api/webhook* (Gophish - Jordan Wright, 2020).

The payload of webhooks has the structure seen in Table 27.

```
{
    "campaign_id": 10,
    "email": "john.doe@outlook.com",
    "time": "2021-02-17T12:22:41.123674904Z",
    "message": "Clicked Link",
    "details":
"{\"payload\":{\"rid\":[\"B9GUySO\"]},\"browser\":{\"address\":\"127.0.0.1\",\"u
ser-agent\":\"Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:85.0) Gecko/20100101
Firefox/85.0\"}}"
}
```

Table 27. JSON Webhook payload structure.

Each of the json values are extracted and a calculation process is started for each of the potential messages. In Table 28 is a simplified explanation where the content inside each if statement is replaced with comments for a general overview.

```
if message == 'Email Sent':
   # add campaign participated to user statistics.

elif message == 'Email Opened':
   # add email opened to user statistics.

elif message == 'Clicked Link':
   # add email clicked or attachment downloaded to user statistics.
```

```
    # add risk score debit if this was their ->
    # first click for this campaign.

elif message == 'Submitted Data':
    # add submitted data/credentials user statistics.
    # add risk score debit if this was their ->
    # first submit for this campaign.

elif message == 'Email Reported':
    # add email reported to use statistics.

    # if email was clicked or data was submitted before the report ->
    # then the debit gained from 'Email reported' is halved.

    # if email was reported by this user before, do not add debit.

# finally.
# save the user action to the campaign and user to keep a record of   history
```

Table 28. Simplified webhook calculation pseudocode.

In the project *Clicked Link* section mentions that it is either adding email clicked or attachment downloaded. Because of a limitation in Gophish it is not possible to track if a target downloads an attachment. To be able to send invoice scams and track if the targets tried to click the attachments in the email, embedded links were added. This way targets would be redirected to a link when they click the attachment picture. In Figure 27 is how an attachment picture would look like in an inbox.

Figure 27. Image of how attachments look in a phishing email.

When clicking the image in Figure 27, Gophish sends a *Clicked Link* payload to the webhook. From the *campaign_id* in the payload it is checked which *template* and *landing_page* was used in the campaign, and if this *landing_page* equals the *Attachment* page this means that the campaign was an attachment scam. Table 29 demonstrates how this works.

```
# Get campaign_id from json
try:
    campaignId = json_data['campaign_id']
except KeyError:
    return returnMessage("Could not extract data from webhook",
status.HTTP_400_BAD_REQUEST)

# Get the campaign from the database using the retrieved id
try:
    camp = Campaign.objects.get(id=campaignId)
except Campaign.DoesNotExist as c:
    return returnMessage(str(c), status.HTTP_404_NOT_FOUND)

# Check if campaign landing page indicated an attachment attack
elif camp.page == "Attachment":
    attachmentCampaign = True
else:
    attachmentCampaign = False

elif message == 'Clicked Link':
    # add attachment instead of clicked link.
    if attachmentCampaign:
        emp.add_email_attachment_downloaded()
    else:
        emp.add_email_clicked_link()
emp.addCredit(credit)
```

Table 29. Demonstration of how phishing attacks are mapped to correct statistics.

4.5.3.1.1. Timeline Objects

Each of the retrieved payloads are serialized, mapped in a relationship with their employee(target) and the campaign. This gives a returnable json object from the API which can indicate the history of events for an individual user or an entire attack. One of these objects can be seen in Table 30.

```
{
	"campaign_id": "10",
	"employee_email": "john.doe@outlook.com",
	"time": "2021-02-17T12:22:41.123674904Z",
	"message": "Attachment downloaded",
	"details":
"{\"payload\":{\"rid\":[\"B9GUySO\"]},\"browser\":{\"address\":\"127.0.0.1\",\"u
ser-agent\":\"Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:85.0) Gecko/20100101
Firefox/85.0\"}}",
	"risk_score_increment": 12,
	"risk_debit_increment": 0,
	"risk_credit_increment": 12
 }
```

Table 30. Example Timeline JSON object.

When these timeline objects are combined they can give a detailed overview of how a user performed in a given campaign, as shown in Table 31.

```
[
      {
	"campaign_id": "31",
	"employee_email": "john.doe@outlook.com",
	"time": "2021-02-17T12:22:41.123674904Z",
	"message": "Email Opened",
	"details": "<cut for readability>",
	"risk_score_increment": 0,
	"risk_debit_increment": 0,
	"risk_credit_increment": 0
	},
      {
	"campaign_id": "31",
	"employee_email": "john.doe@outlook.com",
	"time": "2021-02-17T12:22:41.123674904Z",
        "message": "Clicked Link",
        "details": "<cut for readability>",
        "risk_score_increment": 10,
        "risk_debit_increment": 0,
        "risk_credit_increment": 10
        },
        {
        "campaign_id": "31",
        "employee_email": “john.doe@outlook.com",
        "time": "2021-02-17T12:22:41.123674904Z",
        "message": "Submitted Data",
        "details": "<cut for readability>",
        "risk_score_increment": 20,
        "risk_debit_increment": 20,
        "risk_credit_increment": 0
        },
        {
        "campaign_id": "31",
        "employee_email": "john.doe@outlook.com",
        "time": "2021-02-17T12:22:41.123674904Z",
        "message": "Email Reported",
        "details": "<cut for readability>",
        "risk_score_increment": -5,
        "risk_debit_increment": 0,
        "risk_credit_increment": 5
        }
]
```

Table 31. Example of how multiple Timeline objects gives a detailed timeline of actions.

Read more about the possible statistics from Timeline objects in the RE section of 5.3. Attachment 3:API Documentation.

4.5.3.1.2. Risk Score calculation.

The webhook process does also compute risk score calculations based on a few parameters.

1. The template used in the campaign. Retrieved from the campaign relationship.

2. Employee risk level. Retrieved from the email.
3. Action performed by the employee.
    a. Actions which increase risk score.
    b. Actions which decrease risk score (report email).

The action performed by the customer can have a different impact on how the template score is represented in the final calculation.

4.5.3.1.3. Email Opened

This is the first step in any campaign and the click is tracked and recorded, but the action is not malicious and does not increase risk score. This is not malicious because newer email clients do not automatically download content and malicious data (Hoffman, 2017).

4.5.3.1.4. Click link & attachment download

Both of these use the same calculation where the template score is used in full. These attack types require one action from the target; clicking the link or attachment. Clicking the link or attachment therefore gives the entire score provided in the template, as seen in Table 32.

```
employee.riskMultiplier() # equals 1.25 multiplier
template_score = 30
action = 'Clicked Link' # 1/1 of the score
# multiply the parameters:
total = template_score * 1/1 * employee.riskMultiplier()
# score for action = 37,5
```

Table 32. Attachment & clicked link calculation.

4.5.3.1.5. Credential Harvesting & BankId harvesting

Here the *template_score* is split depending on the action of the user. These attack types require two different actions from the user and therefore the score is split for the different actions. Clicking the provided email link gives ⅓ of the score, as seen in Table 33.

```
employee.riskMultiplier() # equals 1.25 multiplier
template_score = 30
action = 'Clicked Link' # ⅓ of the score
```

```
# multiply the parameters:
total = template_score * 1/3 * employee.riskMultiplier()
# score for action = 12,5
```

Table 33. Click link in credential harvesting attack calculation.

Submitting data to the landing_page input form gives ⅔ of the data. Example in Table 34.

```
employee.riskMultiplier() # equals 1.25 multiplier
template_score = 30
action = 'Submitted Data' # ⅔ of the score
# multiply the parameters:
total = template_score * 2/3 * employee.riskMultiplier()
# score for action = 25,0
```

Table 34. Submit data in credential harvesting campaign calculation.

By comparing the total scores from the Credential Harvesting campaign and the Click Link campaigns, the total sum of the score is the same when the *template_score* is the same.

4.5.3.1.6. Reporting

If the employee reports the email before triggering a malicious action his risk score will be decreased by 10 points. However, if he already clicked a link, submitted data, or downloaded an attachment his score will only be decreased by 5 points.

4.5.4. CRUD operations on companies as an admin

This section discusses the company endpoints, with pseudocode and code examples. These operations are only available for the backend administrator.

4.5.4.1. CO1: POST company

When creating a new company there are several steps happening. A new company is created in the Django database and a new User in Gophish is created. All email templates and landing pages will also be imported into the new Gophish user.

/api/company/ will receive a post request in the format as shown in Table 35.

```
{
       "name": "Example Company",
      "password": "password",
       //Gophish settings
      "smtp_host": "smtp.eu.example.org:465",
       "smtp_username": "example@example.com",
       "smtp_pw": "password",
       "url": "http://127.0.0.1"
}
```

Table 35. JSON used to create a new Company.

Then the endpoint will handle the request as in Table 36. Pseudo code of creating a company.

```
# POST request
def company(request):
# serialize the data to a company django model
serializer = CompanySerializer(request.data)
if serializer.is_valid():
# saves company to database
new_company = serializer.save()
# put information from the newly created company into a json
json_to_gophish =
{
"id": new_company.id,
"username": new_company.name,
"role": "user",
"password": "<strong random generated password>",
"password_change_required": False
}

# send data to Gophish api to make a new Gophish "User"
postCompanyToGophish(json_to_gophish , request)

if "Creating a new User in Gophish fails":
# Delete the newly created company and return an error message in the response.
```

```
else:
# Save the api key from Gophish in the new company
# Create a new User in Django and set the user company relationship
# Import email templates and landing pages

return Response and status code 201 Created
```

Table 36. Pseudo code of creating a company

Response if company was successfully created as shown in Table 37.

```
{
  "msg": "Company created successfully",
  "id": 9,
  "name": "Example Company",
  "template_import_status": "20/20 templates imported.",
  "landingpage_import_status": "16/16 landingpages imported.",
  "template_import_details": [
       {
         "template_name": "3E_helsenorge_invoice.json",
         "success": true,
         "msg": "Template imported successfully."
       },
# etc however many templates is imported
       {
         "template_name": "3E_Paypal_LoginNotice.json",
         "success": true,
         "msg": "Template imported successfully."
    },
    ],

    "landingpage_import_details": [
       {
         "page_name": "3N_IDportenOnMobileLP.json",
         "success": true,
         "msg": "Landing page successfully imported"
       },
# etc however many landing pages is imported
       {
         "page_name": "2N_Link_Surveymonkey.json",
         "success": true,
         "msg": "Landing page successfully imported"
    },
    ]
}
```

Table 37. JSON response when creating a company

4.5.4.2. CO1: GET all companies

Here the full power of django is utilized. Provide a queryset and put it into the serializer. The response will then be returned as a json body as in Table 38 with all the data needed.

```
@api_view(['POST', 'GET'])
@permission_classes([IsAdminUser])
# /api/company
def company(request):
    if request.method == 'GET':
        queryset = Company.objects.all().order_by('id')
        serializer_class = CompanyAllSerializer(queryset, many=True)
        return Response(serializer_class.data, status=status.HTTP_200_OK)
```

Table 38. Code of GET all companies.

4.5.4.3. CO2: GET, DELETE and PUT company by Id

The rest api also supports getting, updating and deleting companies. When getting and updating a company, the recommended way from the DRF documentation is utilized. For deleting a company it's a bit different. The Gophish users are also deleted when deleting a company.

In Table 39 the full code of the function is shown with some clarifying comments.

```
#/api/companies/<id>
@api_view(['GET', 'PUT', 'DELETE'])
@permission_classes([IsAdminUser])
def companyByID(request, company_id):
    try:
        comp = Company.objects.get(id = company_id)
        gophish_id = comp.gophish_id
        api_key = request.user.API_KEY
    except Company.DoesNotExist:
        return returnMessage("Company with: " + company_id + " ID does not
exist!", status.HTTP_404_NOT_FOUND)

    # Get company
    if request.method == 'GET':
        serializer = CompanyAllSerializer(comp)
        return Response(serializer.data, status=status.HTTP_200_OK)
    # Update company
    elif request.method == 'PUT':
        serializer = CompanyAllSerializer(comp, data=request.data)
        if serializer.is_valid():
            serializer.save()
            return Response(serializer.data, status=status.HTTP_200_OK)
        return Response(serializer.errors, status=status.HTTP_404_NOT_FOUND)

    elif request.method == 'DELETE':
        try:
            # Delete Gophish user
            res = requests.delete(host + '/api/users/' + str(gophish_id),
verify=False, headers={'Authorization' : api_key})
        except:
            return returnMessage("Gophish server down",
status.HTTP_500_INTERNAL_SERVER_ERROR)
# “falseCheck” checks if something went wrong when deleting in Gophish. If so,
return an error message from Gophish.
        if falseCheck(res):
            return Response(res.json(), status=res.status_code)
        comp.delete()
        return Response({'msg': 'Company deleted'}, status=status.HTTP_200_OK)
```

Table 39. Code of the function “companyByID”

4.5.5. Results of campaigns endpoints

The results endpoints show different statistics from the phishing campaigns. All of these are made to display different information.

A company can easily see how employees performed in the phishing campaigns and get a clear picture of what needs improving.

4.5.5.1. As a company:

4.5.5.1.1. RE3 GET company statistics

This will return statistics from all the phishing campaigns performed by a specific company.

First make a queryset with all the employees of the company. Then add all the relevant stats together and return a JSON with all the needed information. The *totalResultPerEmp* function, Table 40, takes in all employees. Then loops through all employees and then adds their different stats together.

```
@api_view(['GET'])
def resultsPerCompany(request):
    if request.user.company == None:
        return returnMessage("You're not assigned to a company.",
status.HTTP_405_METHOD_NOT_ALLOWED)
    # Get all employees from a company
    emps = Employee.objects.filter(company__id=request.user.company.id)
    # Loops through all employees and adds their stats together.
    results = totalResultPerEmp(emps)
    results['Company'] = request.user.company.name
    return Response(results, status=status.HTTP_200_OK)
```

Table 40. Code of function "resultsPerCompany"

```
# Pseudocode
def totalResultPerEmp(emps):
# This is done for all that stats wanted
for emp in emps:
total_risk += emp.risk_score
# Then add all the field in a JSON
results = { "total_risk_score": total_risk}
```

```
return results
```

Table 41. Pseudocode of "totalResultsPerEmp"

Then the response shown in Table 42 will be returned.

```
# JSON response of Total result of phishing campaigns
{
  "total_risk_score": 37,
  "total_campaings": 22,
  "total_email_opened": 40,
  "total_links_clicked": 23,
  "total_email_attachment_downloaded": 5,
  "total_data_submitted": 16,
  "total_reported": 10,
  "Company": "Example"
}
```

Table 42. JSON response of total result of phishing campaigns.

4.5.5.1.2. RE5 GET - Get total statistic by campaign id

This will return the total statistic for one campaign. Here the desired data was stored in the timeline object of the campaigns. So loop through all the timeline objects of the campaign and add all the fields wanted to a total score. In Table 43. Pseudocode of GET results from specific campaigns is shown.

```
# Result of campaigns per campaign
# /results/campaign/<id>
@api_view(['GET'])
def resultsPerCampaignPerCompany(request, campaignId):
# First get the company object
camp = Campaign.objects.get(id=campaignId)
    # Then get all timeline object from one campaign
    for i in Timeline.objects.filter(campaign_id=camp.id):
        previousWebhooks.append(i)
    # Look at every single timeline object and extract the fields need
    for j in previousWebhooks:
        risk_score += j.risk_score_increment
        message = j.getMessage()
        if message == 'Email Sent':
            email_sent += 1
        elif message == 'Email Opened':
            email_opened += 1
        elif message == 'Clicked Link':
            link_clicked += 1
        elif message == 'Submitted Data':
            data_submitted += 1
        elif message == 'Email Reported':
            email_reported += 1
# Add data to a JSON.
# Then return the data.
return Response(results, status=status.HTTP_200_OK)
```

Table 43. Pseudocode of GET results from specific campaign

4.5.5.1.3. RE1/RE2 GET - results all employees and employee by id

These endpoints are a bit more straightforward since the fields wanted already are stored in the employee model. Getting the field is done by making a custom serializer that only will show the wanted fields.

```
# Pseudocode
@api_view(['GET'])
def resultsEmployee(request):
# First do some checks to make sure the company is correct.
# Get a queryset set with all employees. To only show one employee, make a query
with only one employee.
queryset = Employee.objects.filter(company__id=request.user.company.id)
# Then use a serializer to get information from the database.
serializer_class = ResultSerializer(queryset, many=True)
return Response(serializer_class.data, status=status.HTTP_200_OK)
```

Table 44. Pseudocode of results Employee.

This will return a JSON shown in Table 45.

```
{
    "id": 35,
    "email": "perfisker@yandex.ru",
    "company": {
      "id": 1,
      "name": "Example Company"
    },
    "first_name": "Per",
    "last_name": "Fisker",
    "position": "Cost Accountant",
    "risk_score": 27,
    "risk_score_credit": 37,
    "risk_score_debit": 10,
    "risk_level": 2,
    "risk_category": "High",
    "email_sent": 2,
    "email_opened": 3,
    "email_clicked_link": 3,
    "email_data_submitted": 5,
    "email_reported": 2,
    "email_attachment_downloaded": 2
  }
```

Table 45. JSON payload from response.

4.5.5.1.4. RE5 - GET employee statistics per campaign

The response here is pretty similar to the endpoints described in 4.5.5.1.3., but the technical solutions were a bit different. And it only shows the employees that took part in one particular campaign.

The data wanted is stored in the timeline object. Information is extracted by looking through all the timeline objects.

```
def campaignEmployeeStats(request, campaign_id):
# Make sure request sender is assigned to a company
# Get correct campaign from campaign id input
# First loop through all employees in the campaign
# And inside that loop. Loop through the timeline object to that employee.
# Then append results to an array that stores all the data.
# Return all data
```

Table 46. Pseudocode of statistics per campaign.

4.5.5.2. GET statistics As an admin (RE7-10)

An admin user have the possibility to get result of campaigns by:

- All employees of every company
- Total results of phishing campaign for every company
- Statistic per company
- Statistic for all employees by ID

All these endpoints are pretty similar so they won't be discussed in detail here. They get the desired queryset and use a serializer to display the data in a JSON response. More detailed info about this in 5.6. Attachment 6: Backend Source Code and 5.3. Attachment 3: API Documentation.

4.5.6. Risk score

```
class Employee(models.Model):
   id = models.AutoField(primary_key=True)
   company = models.ForeignKey('Company', on_delete=models.CASCADE, blank=True,
null=True)
   # risk score fields
   risk_score = models.IntegerField(default=0) # total score
   risk_score_credit = models.IntegerField(default=0) # credit(bad actions)
   risk_score_debit = models.IntegerField(default=0) # debit (good actions)
   risk_level = models.IntegerField(default=1, validators=[MinValueValidator(1),
MaxValueValidator(5)]) # risk level, between 1 and 5. Higher risk level, more
```

```
dangerous if the employee is phished.
   department = models.CharField(max_length=255, unique=False)
   risk_category = models.CharField(max_length=255, unique=False) #either low,
medium or high
```

Table 47. Part of the employee class with risk score fields.

The product provides a risk score system. This system gives each employee a risk score, which changes after how the target handles every single emulated phishing attack. With this, the company gets a good way to track how their employees are doing, so they know who needs to practice on being better to detect these things and who not. Without the risk score it would be harder for the company to find the vulnerabilities within their employees and train them to do better.

4.5.6.1. Variables

4.5.6.1.1. risk_level

An employee is given a risk level when added to the system. This level is an integer between one and five which indicates how important it is that the employee doesn't get phished. The higher the risk level the more dangerous it is for the company if the employee gets scammed. For example, will a CEO usually have a higher risk level than just a normal employee. This risk level is used in a function which calculates a risk multiplier.

The risk multiplier is used when points are added to an employee's risk score. The higher multiplier, the more points. For example, risk_level = 1 gives a risk multiplier = 1, whilst a risk_level = 3 gives a risk multiplier = 1,5.

4.5.6.1.2. risk_score_credit

This is one of the two factors that is used to calculate the total risk score. The risk score credit is being increased when the employee is doing something bad on a phishing mail. For example, that the employee clicks a link. This means that the lower the risk_score_credit is, the better is the employee when it comes to handling phishing attacks.

4.5.6.1.3. risk_score_debit

The other of the two factors used to calculate the total risk score. The debit is increased when the employee reports an email. So the higher the risk score debit is, the better it is as this means that the employee is good at recognizing phishing attacks. The amount of points that is added to the debit can vary depending on the actions the employee has done before. For example, less points will be added if the employee already has clicked the link before reporting the email compared to reporting it straight away.

4.5.6.1.4. risk_score

$risk\_score = risk\_score\_credit - risk\_score\_debit$

The total risk score. Every time the risk_score_credit is increased, the risk_score is increased with the same number, whilst every time the risk_score_debit is increased, the risk_score is decreased with the same number. So the lower the risk_score is, the better it is.

4.5.6.1.5. risk_category

A text field describing which category the employee has. The category is based on the risk score and is either high, medium or low. The higher risk score, the worse it is and then the employee will get a high risk_category. For an employee that is yet to participate in any campaigns the risk_category is set to “default”. This field is not used in any calculations, it is just made to make it easier to understand how the employee is doing.

template_score: An integer. Each template is given a template score. The more dangerous/better made template, the higher is the template_score. The template_score is used together with the risk_multiplier when calculating the risk_score_credit.

4.5.6.2. How the calculation works

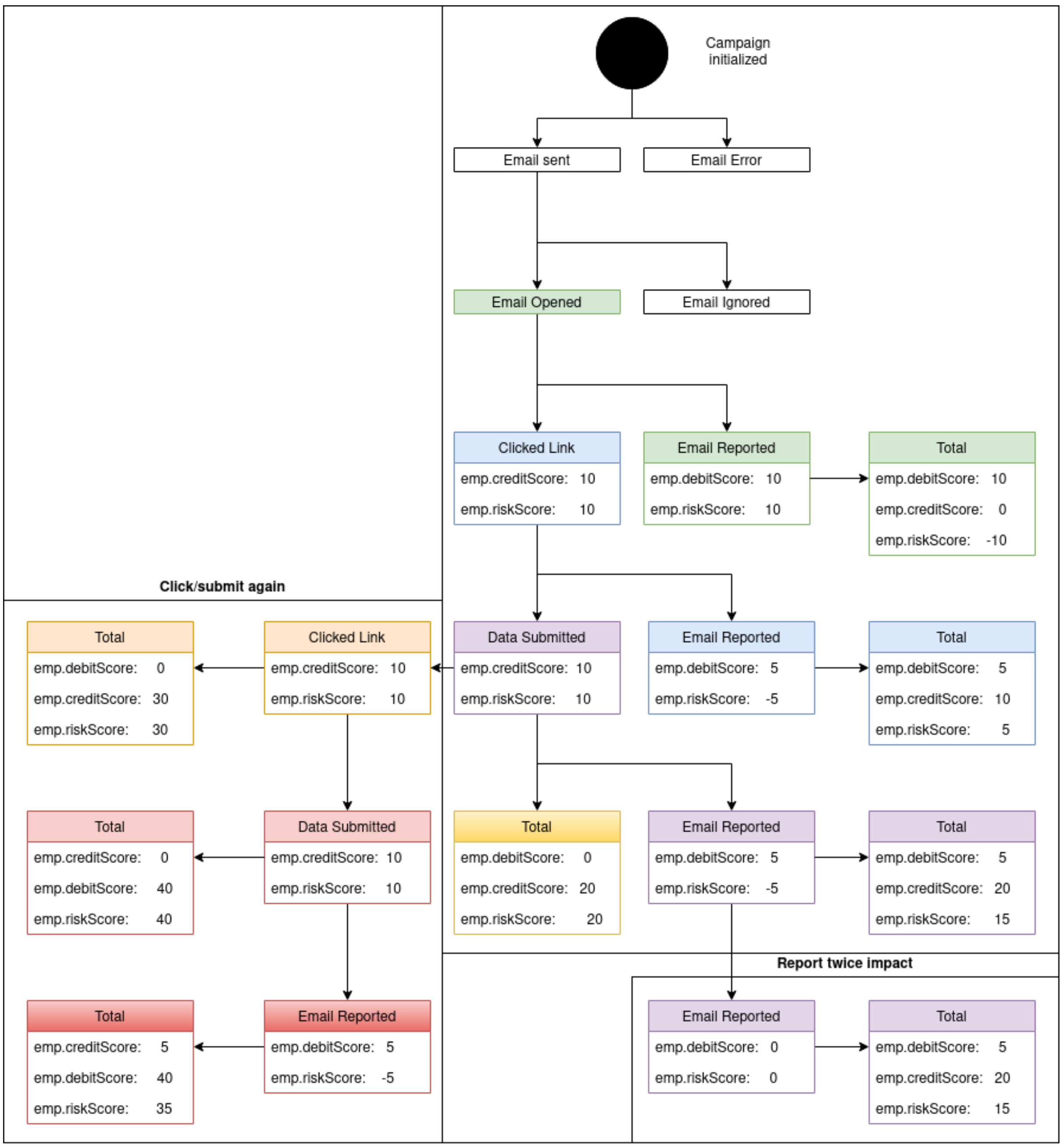


Figure 28. Risk score calculation.

Figure 28. Risk score calculation shows how the risk_score is calculated. It shows calculations of an employee with a risk multiplier of 1. It also uses a phishing attack with an email containing a link to a page where the target is asked to put in some data.

Some of the important things that can be seen on this diagram is how the points the employee gets for reporting the email changes depending on when it is reported. The employee gets 10 points added if the email is reported right away, and just 5 points added if the email is reported after some other actions have been done.

Another important take is that the employee only gets points the first time the email is reported. This is done to prevent the employees from exploiting the reporting system by reporting the same email repeatedly to improve their total risk score.

More about how the different actions are registered and how the points are added can be found in 4.5.3 Webhooks.

4.5.7. Import AttackTemplates on Company creation

When creating a new company default AttackTemplates are created and imported for that company. These default templates are general templates which might require a few edits before they are ready to be used for that particular company. They are there to make it easy to use the software after setting up a new customer.

In the backend the templates and landing pages who make up these AttackTemplates are stored in the *phishing-resources* folder.

```
phishing-resources/
├── landing_pages
│   ├── 1N_IDportenOnMobileLP.json
│   ├── 2E_Link_Surveymonkey.json
│   ├── 2E_NIgerian_PrinceForm.json
│   ├── 2N_IDportenOnMobileLP.json
│   ├── 2N_Link_Surveymonkey.json
│   ├── 3E_Link_Google_Meet.json
│   ├── 3N_IDportenOnMobileLP.json
│   ├── 3N_Link_Google_Meet.json
│   ├── 3N_MicrosoftLP.json
│   ├── Attachment.json
│   ├── GoogleLP.json
│   ├── InstagramLP.json
│   ├── LInkedInLP.json
│   ├── None.json
│   ├── PaypalLP.json
│   └── ZoomResetLP.json
└── templates
    ├── 1E_attachment_download.json
    ├── 1E_GoogleDriveTP.json
    ├── 1E_Nigerian_PrinceNotice.json
    ├── 1N_attachment_download.json
    ├── 2E_GoogleDriveTP.json
    ├── 2E_Link_SurveyMonkey_TP.json
    ├── 2E_paycheck.json
    ├── 2N_Link_SurveyMonkey_TP.json
    ├── 2N_paycheck.json
    ├── 3E_GoogleDriveTP.json
    ├── 3E_helsenorge_invoice.json
    ├── 3E_Instagram_ResetNotice.json
    ├── 3E_LInkedIn_InvitationNotice.json
    ├── 3E_Link_Google_Meet_TP.json
    ├── 3E_MicrosoftTP.json
    ├── 3E_Paypal_LoginNotice.json
    ├── 3E_Zoom_MeetingNotice.json
    ├── 3N_helsenorge_invoice.json
    ├── 3N_Link_Google_Meet_TP.json
    └── 3N_SkatteetatenTP.json
```

Table 48. Folder architecture in the phishing-resources folder.

If a new AttackTemplate is to be imported when companies are created it can be added to the folder phishing-resources (Table 48), and imported for all existing companies using other endpoints 4.5.7.1. AttackTemplates for all companies. When a company is created with the CO1 endpoint found in the 5.3. Attachment 3:API Documentation, the returned JSON data indicates if all the AttackTemplate components were correctly imported.

```
{
    "msg": "Company created successfully",
    "id": 1,
    "name": "Your Company",
    "template_import_status": "20/20 templates imported.",
    "landingpage_import_status": "16/16 landingpages imported.",
    "template_import_details": [
        #list with details for each imported template
    ],
    "landingpage_import_details": [
        #list with details for each imported landing page
    ]
}
```

Table 49. JSON response when creating a company

If the AttackTemplate requires editing before being ready the “description” field usually notifies the user that some fields need to be changed. A typical field can be the ‘from_field’ which is who the email appears to be sent from. These changes can be done with the AT2 endpoint found in the 5.3. Attachment 3:API Documentation.

4.5.7.1. AttackTemplates for all companies

If all companies require a new template or an existing one has to be deleted or updated, there are endpoints for this too. Take a look at the AT3 and AT4 requests in the 5.3. Attachment 3: API Documentation.

4.6. Cloud Setup & Deployment

This section discusses the cloud setup on Google Vm and deployment of the PoC.

4.6.1. The cloud setup

4.6.1.1. Google cloud platform

CYBR provided a private cloud instance, which was used for testing and hosting the backend and the frontend. Performing changes on the platform proved difficult as CYBR had only given minimal permission and access. This was understandable as receiving admin access would be unsafe considering the assets they have on the platform. Therefore, every firewall setting change on the cloud platform had to go through them. On the platform CYBR created a virtual machine running Ubuntu 20.04 LTS, with the possibility for external ssh connection. To gain access to the vm, the team provided individually a public key which would be placed in the cloud. Then with the usage of an ssh command and the private key related to that public key, access to the vm through a local machine was possible. After cloning the code from github to the vm, the last step was opening the right ingress ports on the platform so that the code could be accessed on the web.

4.6.1.2. Required ports

Service	Port
Phishing frontend	3000
Phishing backend	8000
Gophish	3333
PostgreSQL	5432

Table 50. Ports used in the setup.

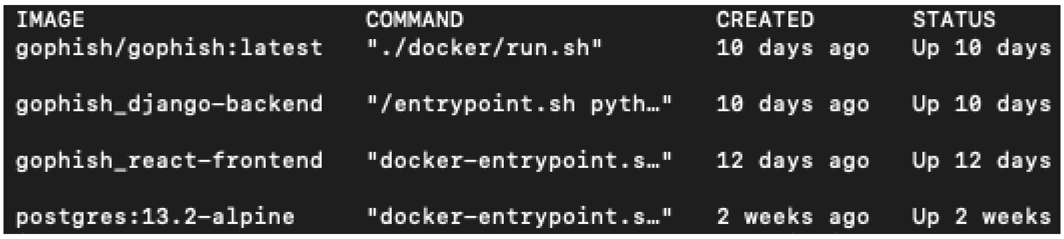
4.6.1.3. Virtual machine

CYBR created an vm-instance running Ubuntu 20.04 LTS. As a fresh install, Ubuntu had none of the required tools or frameworks to optimally run the code. Before cloning the code a seperate folder was made to store all the project files. Then Docker and Docker-Compose were downloaded as the last step of vm setup.

4.6.2. The docker setup

4.6.2.1. The docker containers

Docker utilises containers which can hold compressed versions of software (called Docker image) to run specific tasks. In this project there are 4 containers running simultaneously. These 4 containers are Gophish, the frontend, the backend and a database instance running postgreSQL.



```
IMAGE                    COMMAND                  CREATED       STATUS
gophish/gophish:latest   "./docker/run.sh"        10 days ago   Up 10 days

gophish_django-backend   "/entrypoint.sh pyth…"   10 days ago   Up 10 days

gophish_react-frontend   "docker-entrypoint.s…"   12 days ago   Up 12 days

postgres:13.2-alpine     "docker-entrypoint.s…"   2 weeks ago   Up 2 weeks
```

Figure 29. Image from the vm of the running containers

Before being able to run the containers some modifications had to be done. If the frontend or the backend was going to be runnable containers they had to be turned into docker images. To make the code into a runnable docker image, both the frontend and backend needed customised Dockerfiles. They are essentially text files that contain commands for setting up and building a docker image.

```
#pulling official image
FROM python:3.9.4-alpine

#set env vars
ENV PYTHONUNBUFFERED 1
ENV PYTHONDONTWRITEBYTECODE 1

#working dirs
```

```
WORKDIR /poc-phishing-backend

# install psycopg2 dependencies
RUN apk update \
    && apk add postgresql-dev gcc python3-dev musl-dev

#install dependencies
RUN pip install --upgrade pip
COPY requirements-pip.txt requirements-pip.txt
RUN pip install -r requirements-pip.txt

COPY ./entrypoint.sh /

#copy whole project
COPY . .

ENTRYPOINT ["/entrypoint.sh]
```

Table 51. The dockerfile for the backend

Table 51 is the Dockerfile of the backend. The *FROM* command starts a new build stage and makes the base image ready for taking subsequent instructions. In this case the base image is *python:3.9.4-alpine*, this is because the backend code is entirely written in python and the container won't run if it cannot register if python is installed. The *WORKDIR* command chooses the work directory for where the dockerfile is going to perform its activation instructions. Then the rest of the dockerfile instructions are mostly for downloading and installing requirements needed by the code to run accurately(`RUN pip install -r requirements-pip.txt`). This also includes making the code suitable for the postgreSQL database and adapting the database service with the code. Lastly the *ENTRYPOINT* command configures how the container will run, and here it runs a script called entrypoint.sh. This script checks if the postgreSQL database has started and migrates the database if true.

```
FROM node:15.14-alpine
WORKDIR /poc-phishing-frontend
```

```
ENV PATH="./node_modules/.bin:$PATH"
COPY . .
RUN npm run build
```

Table 52. The dockerfile for the frontend

Table 52 shows the Dockerfile from the frontend. The frontend was developed in React and therefore the base image here is *node:15.14-alpine*. React is built upon the node framework and the container can only run if it recognises that node is installed. Mostly the dockerfile has the same setup as the one from the backend, but there is one difference. The last command uses a different syntax, which is *RUN npm run build*. The *RUN* command in docker is a way of starting a desired action and npm run build is a command derived from node and builds the frontend service.

There was no need to make any custom Dockerfiles for both the Gophish and postgreSQL images as prebuilt images were used for this setup.

4.6.2.2. The docker-compose setup

The concept of Docker Compose is binding and running multiple Docker containers effortlessly. Port delegation and container networking tasks are more efficient and made easier to handle.

```
version: '3.8'

services:
  django-backend:
    build:
      context: ./poc-phishing-backend
    command: python manage.py runserver 0.0.0.0:8000
    volumes:
      - ./poc-phishing-backend:/poc-phishing-backend
    ports:
      - 8000:8000
    env_file:
      - ./.env.dev
    depends_on:
      - db
      - Gophish
  react-frontend:
    build:
      context: ./poc-phishing-frontend
    volumes:
      - ./poc-phishing-frontend:/poc-phishing-frontend/build
    ports:
      - 3000:3000
    command: npm start
  db:
    image: postgres:13.2-alpine
    volumes:
      - postgres_data:/var/lib/postgresql/data/
    environment:
      - POSTGRES_USER=<username>
      - POSTGRES_PASSWORD=<password>
      - POSTGRES_DB=<name-of-db>
  Gophish:
    image: gophish/gophish:latest
    ports:
      - 3333:3333
      - 80:80

volumes:
  postgres_data:
```

Table 53. The docker-compose.yml file used for the setup

Table 53 is the *docker-compose.yml* file which contains all the instructions for automating the container setup. In this file all the containers which make up the project are present, and by going down the list the instructions are short and concise. Seeing as backend and frontend already has a Dockerfile containing most of the necessary setup, only defining the location of the Dockerfile is required here. But with the postgreSQL and the Gophish images being a default package, a bit more instructions has to be given. The last step for making the container run properly is giving the right ports and writing the command for starting the application.

```
DEBUG=0
SECRET_KEY=<secret key>
DJANGO_ALLOWED_HOSTS=localhost 127.0.0.1 [::1]
SQL_ENGINE=<sql>
SQL_DATABASE=TEXT
SQL_USER=<db username>
SQL_PASSWORD=<password of user>
SQL_HOST=TEXT
SQL_PORT=<port to run on>
DATABASE=TEXT
```

Table 54. The structure of the env.dev file using empty data

To make sure that the backend can properly communicate with the frontend, Gophish and postgreSQL, the usage of an *env.dev* file is required. This file is used for defining environment parameters and variables. In the *DJANGO_ALLOWED_HOSTS* variable the backend says that these IP addresses are allowed to communicate with the backend and if an IP is not on the list it won't be reachable. The Gophish and the frontend IPs are written here. The last part of this file makes sure that the backend is connected to the postgreSQL DB and that it automates the creation of an user on that service.

Then to build the entire application type in the command in Table 55.

```
docker-compose up -d --build
```

Table 55. The docker-compose command for building the setup

4.7. Frontend

Frontend was made as a super simple GUI to display core functionality of the rest api. This section will give a quick overview of how the website is made and how it interacts with the REST api. Create react app handles everything "under the hood" and is used to configure a single page application.

4.7.1. Material UI

To speed up the development process material ui components have been used when it was possible. The styling is kept to a minimum on the components used. The components come with an api and minor changes have been made to fit the needs. Many components are inspired by examples given by material ui and use the same styling. E.g, the login page is inspired by a login component example made by material UI.

4.7.1.1. Example of how components are used.

Here the *DataGrid* and *GridToolBar* are imported from Material UI. The datagrid component comes with an api so it can be customised to anyone's needs. In the example under it is added a *checkboxSelection* that will show a checkbox on every row. There is also added a *onRowDoubleClick* that will direct to a single employee. Table 56 shows an example of a DataGrid component.

```
import { DataGrid, GridToolbar } from '@material-ui/data-grid'
<DataGrid
 columns={cols}
 // data fetched from endpoint.
 rows={data}
 checkboxSelection
 // When double clicking a row. Go to that employee.
 onRowDoubleClick={(rowClick) => {
   history.push('/employee/' + rowClick.id)}}
 components={{
  Toolbar: GridToolbar,}}
/>
```

Table 56. Example of how the DataGrid component is used.

```
const cols = [
  { field: 'id', hide: true, width: 65 },
  { field: 'risk_score', headerName: 'Risk Score', width: 130 },
  { field: 'email_sent', headerName: 'Campaigns', width: 130 },
<part cuts for readability>
  {
    field: 'risk_category',
    headerName: 'Risk Cat',
    width: 120,
    type: 'string',
    cellClassName: (params) =>
      clsx('super-app', {
        default: params.value === 'Default',
        low: params.value === 'Low',
        medium: params.value === 'Medium',
        high: params.value === 'High',
      }),
  },
  { field: 'first_name', headerName: 'First', width: 120 },
  { field: 'last_name', headerName: 'Last', width: 120 },
]
```

Table 57. Example of how columns look like. Columns is the name of the field to display.

4.7.2. Interacting with the server

4.7.2.1. React Query

React query is a data-fetching library for React. It is used to fetch and update data from the django REST API and manage state. Below are some examples of how fetching and update functions are used in the frontend. The methods explained are used in most of the pages on the website and won't go in detail on every single page.

4.7.2.1.1. Fetching data from the server

```
// make a query by using "useQuery"
// 'companies' is the unique-key of the query
// getAdminCompanies: function that fetches data from API
const { isLoading, data, error } = useQuery('companies', getAdminCompanies)

// Show loading circle when data is loading
if (isLoading) return <CircularProgress />

// Display any errors
if (error) return 'An error has occurred: ' + error.message

// data is JSON that is used to display the desired data
return (do something with data)
```

Table 58. Example of how "react query" is used to fetch data in a react component.

4.7.2.1.2. Modifying data on the server

When creating or updating data on the server mutations are used with the *useMutation* hook from react query. Table 59 shows how updating or creating data is done.

```
// make a mutation
// "postAdminComanies" is a function that posts data to the server
const mutationCreateCompany = useMutation((newCompany) =>
    postAdminCompanies(newCompany)
  )
// Use the mutation
// "createCompanyData()" is data needed to create a new company.
mutationCreateCompany.mutate(createCompanyData())
```

Table 59. Example of how modifying data is done.

4.7.2.2. API calls

Functions that call on the api stored in are stored in the file *restapicalls.js*. They are exported so they can be used in multiple components. Removing the need for rewriting code. These functions are used by react query to make queries from the server.

```
import axiosInstance from './axios'

export const getAdminCompanies = async () => {

// “.get” can be replaced with post, delete, and put depending on the endpoint.
  const response = await axiosInstance.get('companies/')
  if (!response.statusText === 'OK') {
    throw new Error(response.statusText)
  }
  return response.data
}
```

Table 60. Example of request to the server.

In *axiosInstance* the base url for the server is stored. So when a call is made using axiosInstance it will look like: *<baseURL>/companies/.*

4.7.3. Create Campaign

There are several things happening on this page. There is a list of all targets, a selection option of all phishing attacks and input field for naming the campaign.

4.7.3.1. Input field

Figure 30. Input field for “Campaign Name”

This is done by utilizing a TextField component wrapper with different options selected.

```
 <TextField
variant='outlined'
margin='normal'
required
fullWidth
id='name'
label='Campaign Name'
name='name'
autoComplete='name'
autoFocus
onChange={handleChange} />
```

Table 61. Example of TextField component.

Input field to give the campaign a fitting name. A *TextField* component is used and will capture the input data. The *handleChange* function will trigger on event changes. This function updates the state of the formData that stores the inputted data. This is done by using the useState hook from react. So everytime the input field changes the state is stored and ready to be used.

```
import React, { useState } from 'react'
// Initial state is just an empty object.
const [formData, updateFormData] = useState(initialFormData)

const handleChange = (event) => {
    updateFormData({
      [event.target.name]: event.target.value,
    })
  }
```

Table 62. Code example of handleChange function.

4.7.3.2. Selecting phishing attack template

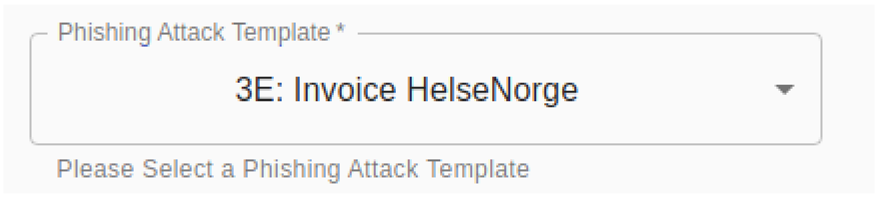

Phishing Attack Template *
3E: Invoice HelseNorge
Please Select a Phishing Attack Template

Figure 31. Select with dropdown when creating a campaign.

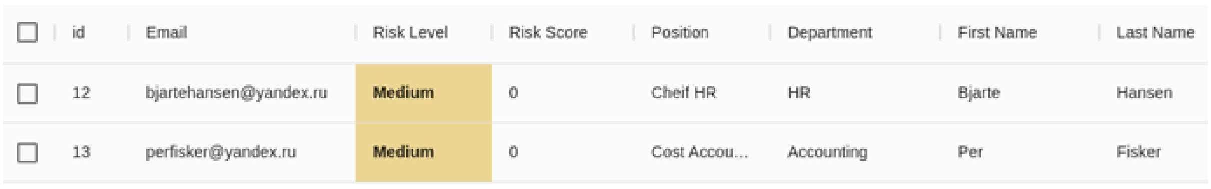
TextField also comes with a select option. Here all the templates are stored in *emailTemplate*. Loop through all templates and put them in a MenuItem componente. *onChange* is also used here and works as described in 4.7.3.1. Input field.

```
<TextField
id='emailTemplate'
select
margin='normal'
fullWidth
variant='outlined'
required
label='Phishing Attack Template'
value={selectedPhishingAttack}
onChange={handleTemplateChange}
helperText='Please Select a Phishing Attack Template'>
// Going through all templates and display them in a <MenuItem/>
{emailTemplate.map((template) => (
 <MenuItem key={template.name} value={template.name}>
                  {template.name}
</MenuItem>
              ))}
</TextField>
```

Table 63. TextField used with the “select” option.

4.7.3.3. Targets

For displaying all targets a *DataGrid* is used as described in [4.7.1.1. Example of how components are used](). With a checkbox for selecting targets to participate in the phishing campaign.



☐	id	Email	Risk Level	Risk Score	Position	Department	First Name	Last Name
☐	12	bjartehansen@yandex.ru	Medium	0	Cheif HR	HR	Bjarte	Hansen
☐	13	perfisker@yandex.ru	Medium	0	Cost Accou...	Accounting	Per	Fisker

Figure 32. Screenshot of datagrid listing employees.

Every checkbox selected is stored by using *onSelectionModelChange* for the *DataGrid* component.

```
onSelectionModelChange={(newSelection) => {
              setSelectionModel(newSelection.selectionModel)
            }}
selectionModel={selectionModel}
```

Table 64. "onSelectionModelChange" function.

4.7.3.4. Putting it all together

Figure 33. Button for creating a campaign.

The button has an *onClick* that will use a function to handle the submit. Now all the data needed is stored in different variables. Then put it all together and make a POST request to the server.

```
const handleSubmit = (event) => {
event.preventDefault()

// Use the selected targets and get the emails. This is done to get the request
on the desired format for the REST API.
```

```
// "selectionModel" contains all the ids of the targets.
const employeeEmails = getEmpEmail(data, selectionModel)

const formatedData =  {
// Input form formData
name: formData.name,
// Selected phishing attack
template: selectedPhishingAttack,
// List of emails of targets.
employees: employeeEmails,}

// Then post this data to the server.
```

Table 65. Pseudo code of "handleSubmit"

4.8. Compliance between requirements and the product

In the tables found in 4.8.1. Research Requirements and 4.8.2. Technical Requirements the initial software and research requirements are listed. For each one there is an overview if it was:

1. Implemented
2. Partly Implemented
3. Not Implemented

And a comment where additional information on what was done or not done for each requirement.

4.8.1. Research Requirements.

Requirement	Status	Comment
R1: Identify the most	Implemented	Have identified that spear phishing is the most

efficient phishing methods.		efficient phishing method against companies.
R1.1: Use the results of the research about effective phishing methods when making attack templates.	Partly implemented	Found that the most efficient phishing methods are tailored to the target through social engineering. This is difficult without detailed information about the potential targets.Therefore are the phishing AttackTemplates focused towards the general public, rather than some specific companies.
R2: Research the possible cost of breach of a phishing attack	Partly implemented	The research contains some numbers of possible costs of breach. The problem is that the amounts vary a lot from attack to attack and that a lot of businesses wont give away all the facts when they are attacked. Therefore it is hard to give an accurate number of the possible cost of breach.
R3: Research different tools and methods to make emulated phishing attacks.	Implemented	After researching different options, Gophish was the clear choice.
R4: Research how phishing awareness training of employees can improve the cyber security of a company	Implemented	The research shows that phishing awareness training improves the employee's ability to identify a phishing attack by a lot. However, no matter how much training that is done, it will never be 100% perfect because of new phishing methods and attacks occurring.
R5: Find out what an attacker thinks about	Implemented	Found that the attacker often exploits topics that there is a lot of fuzz about or pretends to be

when making a phishing attack.		someone else. See section 3.7.4. How a phishing attack works.

Table 66. Overview of how research requirements have been solved.

4.8.2. Technical Requirements.

Requirement	Status	Comment
R6: Phishing servers whitelisted in firewalls.	Implemented	Whitelisted in the deployment environment.
R7: Open source phishing framework should be the backbone of the project.	Implemented	Gophish to track emails. Mailgun has been used as the SMTP server.
R8: Frontend should be a minimalistic GUI for data presentation.	Implemented	React and Material UI have been used to give a simple, visually pleasing GUI to present statistics for the phishing attacks. All endpoints are not implemented in the frontend, but there are enough of them to get a general feeling of the program flow.
R9: It should be easy to import emails and other types of information.	Implemented	All endpoints use the JSON format, and emails are imported with employees with the EM1 endpoint.

R10: A showcase of statistics for the different campaigns that were used.	Implemented	Statistics for the campaigns are presented in the RE and CA sections of the 5.3. Attachment 3: API Documentation.
R11: The possibility to create custom user groups.	Implemented	When creating a campaign through the CO1 endpoint from 5.3. Attachment 3: API Documentation custom groups of any employees within the company can be selected. These groups can also be implemented by first searching for departments or risk categories and therefore also fulfills **R11.1.** All groups are for each company by default, **R11.2**.
R12: Premade email templates are stored in the database or system.	Implemented	Premade "AttackTemplates" are stored in the backend. These are imported into each company when a new one is created 4.5.7 Import AttackTemplates on Company creation.
R13: Store metric data for each company and campaign.	Implemented	Employees get scores related to a campaign, and that relationship is mapped to a company. Therefore all metric data are stored with a relation to a company and a campaign.
R13.1-7: Store different metrics and data.	Implemented	The product is tracking email sent, email opened, clicked links, attachments downloaded, and emails reported.
R14: An option that allows for scheduling before a campaign is	Implemented	The CO1 - create campaign endpoint allows for a JSON field (launch_date) which allows for a ISO8601 formatted time to schedule the

launched.		campaign sometime in the future.

Table 67. Overview of how technical requirements have been solved.

4.8.2.1. Additional technical requirements(if time)

Requirement	Status	Comment
R15: Long term engagements with high priority targets.	Partly Implemented	The backend does support searching and filtering for high priority targets through endpoints such as EM6 or EM7. This way it is possible to create long term attacks by extracting data from who opened a prior email, and send a new email to high priority targets. However, this is not implemented as an automatic pipeline.
R16: A grading system (based on how many clicked the link, which can be helpful later when they are going to calculate the costs. e.g., Risk level.)	Implemented	For each campaign action user risk scores are updated. There is more detailed information about this process for the risk score in 4.5.6 Risk score
R17: Email import: import from other management tools, such as azure active directory.	Not Implemented	It is supported for importing employees with their email through the EM1 endpoint in the 5.3. Attachment 3: API Documentation. However, the functionality to get email and details through azure active directory or other similar tools has not been implemented.

	🟥	The customer also mentioned during the development process that they had their own systems for this, and did not want additional options.

Table 68. Overview of how additional technical requirements have been implemented.

4.8.3. Framework Requirements

4.8.3.1. Security

Requirement	Status	Comment
R18: In the test cases no sensitive data will be stored, and all information are test cases and imaginary numbers. Security of the databases is therefore not a priority.	Implemented	All data in the test cases has been generated with fake information, and no real persons or personal email accounts. Usernames, passwords, and other sensitive information is never stored when trying to capture them.
R19: Authentication and verification of users from different companies will be implemented to demonstrate data separation in a multi-tenant system.	Implemented	All endpoints require a valid JWT token. These do then map the user to their company and create a multi-tenant separated system. More about JWT in 4.4.6 Authentication with JWT.

R20: Only the django server will be exposed to the internet. Gophish servers and api keys will be hidden behind the exposed REST API.	Partly Implemented	In the final deployment the backend API server and the Gophish admin server are exposed to the internet. This makes it possible to access the admin GUI in Gophish to set up webhooks and report settings. However all traffic and API keys are travelling locally from the backend to Gophish and are not exposed to the internet.

Table 69. Overview of how Framework Requirement: Security has been implemented.

4.8.3.2. Scalability

Requirement	Status	Comment
R21: The system should be containerized in separate containers for the backend and frontend.	Implemented	The deployment file contains four different docker containers for the backend(Django REST API), Gophish, PostgreSQL and the frontend.
R22: Each of these containers can be load balanced.	Partly Implemented	The containers do support sharing databases if the configuration file is configured for it. This makes it possible to have multiple docker containers sharing synced databases. However, this is not implemented in the deployment file.
R23: Gophish servers can have multiple companies.	Implemented	Gophish separates companies and uses an API key to send campaigns for each one of them.

Table 70. Overview of how Framework Requirement: Scalability has been implemented.

4.8.3.3. Usage

Requirement	Status	Comment
R24: For reusability provide API documentation which shows available endpoints, their functionality, expected input and output.	Implemented	All endpoints have been documented with expected input, output, required data and status codes. See 5.3. Attachment 3:API Documentation and 5.1. Attachment 1: API Endpoint Test Data.

Table 71. Overview of how Framework Requirement: Usage has been implemented.

5. Attachments

5.1. Attachment 1: Test API report

See the separate pdf: *Attachment-1-Test-API-Report.pdf*

5.2. Attachment 2: User guide

See the separate pdf: *Attachment-2-User-Guide.pdf*

5.3. Attachment 3: API Documentation

See the separate pdf: *Attachment-3-API-Documentation.pdf*

5.4. Attachment 4: Phishing AttackTemplates

See the separate pdf: *Attachment-4-PhishingAttack-Templates.pdf*

5.5. Attachment 5: Frontend Source Code

See the separate zip archive: *poc-frontend-source-code.zip*

5.6. Attachment 6: Backend Source Code

See the separate zip archive: *poc-backend-source-code.zip*

5.7. Attachment 7: Insomnia configuration file

See the separate JSON file: *POC_emulation-insomnia.json*

5.8. Bibliography

Abnormal Security. (2020, Q2). *Abnormal Quarterly BEC Report Q2 2020*. Retrieved May 10, 2021, from https://info.abnormalsecurity.com/rs/231-IDP-139/images/AS_Qtrly_BEC_Report_Q2_2020.pdf

Abnormal Security. (2020, April 21). *Zoom Phishing*. Retrieved May 9, 2021, from https://abnormalsecurity.com/blog/abnormal-attack-stories-zoom-phishing/

BDO. (2021). *Are businesses more vulnerable to Corporate fraud?* Retrieved May 7, 2021, from https://www.bdo.co.uk/en-gb/rethink/business-issues/strategy-operations/has-covid-19-made-your-business-more-vulnerable-to-corporate-fraud

Bisson, D. (2020, October 20). *6 Common Phishing Attacks and How to Protect Against Them*. Retrieved May 7, 2021, from https://www.tripwire.com/state-of-security/security-awareness/6-common-phishing-attacks-and-how-to-protect-against-them/

Cofense. (2021, January). Annual report. Retrieved May 7, 2021, from https://cofense.com/annualreport/

Daly, A. (2019). *Phishing Scams Cost U.S. Companies Billions*. Retrieved May 7, 2021, from https://www.inky.com/blog/phishing-scams-cost-companies-billions

Dam, R. F., & Siang, T. Y. (2021, January). *Personas – A Simple Introduction*. Personas – A Simple Introduction. Retrieved May 20, 2021, from https://www.interaction-design.org/literature/article/personas-why-and-how-you-should-use-them

django-rest-framework.org. (n.d.). *Serializers*. Serializers. https://www.django-rest-framework.org/api-guide/serializers/

DutchNews.nl. (2020, November 12). *Internet con men ripped off Pathe NL for €19m in sophisticated fraud*. Retrieved May 9, 2021, from https://www.dutchnews.nl/news/2018/11/internet-con-men-ripped-off-pathe-nl-for-e19m-in-sophisticated-fraud/

Eclee. (2019). *Agile Method SCRUM*. Agile Method SCRUM. Retrieved May 15, 2021, from https://www.eclee.com/training/scrum/

Embla.net. (2020). *Phishing picture*. Retrieved May 20, 2021, from https://embla.net/index.php/133-blog/nyheter/325-phishing-test

FBI. (2021, April 9). *Internet Crime Report 2020*. Retrieved May 6, 2021, from https://www.ic3.gov/Media/PDF/AnnualReport/2020_IC3Report.pdf

Gophish - Jordan Wright. (2020, July). *Webhooks*. Retrieved May 3, 2021, from https://docs.getgophish.com/user-guide/documentation/webhooks

Gophish - Jordan Wright. (2020, October). *Gophish - Building Your First Campaign*. Gophish - Building Your First Campaign. Retrieved May 3, 2021, from https://docs.getgophish.com/user-guide/building-your-first-campaign

Hoffman, C. (2017, July 12). *Why you can't get infected by just opening an email*. Why you can't get infected by just opening an email. Retrieved May 23, 2021, from https://www.howtogeek.com/135546/htg-explains-why-you-cant-get-infected-just-by-opening-an-email-and-when-you-can/

Insomnia. (2021, April 28). *Download - Insomnia*. Download - Insomnia. Retrieved May 11, 2021, from https://insomnia.rest

KnowBe4. (n.d.). *Phishing*. Retrieved May 7, 2021, from https://www.knowbe4.com/phishing

Lutkevich, B. (2021, January). Retrieved May 21, 2021, from https://searchsecurity.techtarget.com/definition/whaling

Microsoft. (2021). *The New Future of Work*. Retrieved May 7, 2021, from https://www.microsoft.com/en-us/research/uploads/prod/2021/01/NewFutureOfWorkReport.pdf

Nohe, P. (2017, September 1). *MacEwan University Phishing Scam: A Cautionary Tale*. Retrieved May 9, 2021, from https://www.thesslstore.com/blog/macewan-university-phishing-scam-cautionary-tale/

Proofpoint. (2020, December). *2020 State of the phish*. Retrieved May 09, 2021, from https://www.proofpoint.com/sites/default/files/gtd-pfpt-uk-tr-state-of-the-phish-2020-a4_final.pdf

Robertson, K. (2018, April 4). *MacEwan University recovers millions lost in phishing scam*. Retrieved May 9, 2021, from https://globalnews.ca/news/4122937/macewan-university-recovers-millions-lost-in-phishing-scam/

Sanders, D. (2020, March 19). *Blacklist app*. Retrieved May 5, 2021, from https://django-rest-framework-simplejwt.readthedocs.io/en/latest/blacklist_app.html

Sanders, D. (2021, May 18). *Simple JWT*. Simple JWT. Retrieved May 18, 2021, from https://django-rest-framework-simplejwt.readthedocs.io/en/latest/index.html

Schwartz, M. J. (2018, November 13). *French Cinema Chain Fires Dutch Executives Over 'CEO Fraud'*. Retrieved May 9, 2021, from https://www.bankinfosecurity.com/blogs/french-cinema-chain-fires-dutch-executives-over-ceo-fraud-p-2681

Sorsa, D. (2017). *PAIR PROGRAMMING & SCRUM*. PAIR PROGRAMMING & SCRUM. Retrieved May 15, 2021, from https://solvingadhoc.com/wp-content/uploads/2017/09/pair-programming-e1507114531919.jpg

Terranova. (2020). *Terranova's 2020 Phishing report*. Retrieved May 8, 2021, from https://terranovasecurity.com/2020-gpt-report/?utm_campaign=En_GPTReport2020&utm_medium=Google&utm_source=Ads&utm_content=NewAd3&gclid=CjwKCAjw6fCCBhBNEiwAem5SO8oIgjFVtVzMA5pg-uSkRAho6S356pspA4bY3FBFk9FXCKW0Ksq-ExoCsHEQAvD_BwE

Wagner, J. (n.d.). *API-First*. Understanding the API-First Approach to Building Products. Retrieved May 21, 2021, from https://swagger.io/resources/articles/adopting-an-api-first-approach/