Functional Design

Christopher Sulistiyo (4850025)

Christopher.sulistiyo@student.nhlstenden.com

ICT & IC Information Technology Department Emmen

Client

Quality ICT B.V.

Version 1.1 – 21/02/2024

VERSION CONTROL **2**
REMARKS **2**
INTRODUCTION **3**
BACKGROUND AND CONTEXT 3
PROBLEM STATEMENT 3
OBJECTIVES 3
CHAPTER 2 – DESCRIPTION OF THE INFORMATION SYSTEM **4**
SYSTEM OVERVIEW 4
USER STORIES 4
USES CASES 5
ACCEPTANCE CRITERIA 9
MOSCOW ANALYSIS 10
CHAPTER 3 DATA MODEL **12**
DESIGN PATTERNS USED. 15
USER INTERFACE AND SYSTEM DESIGN 21
CHAPTER 4 – OUTPUT **28**
CHAPTER 5 REQUIRED INPUT **34**
CHAPTER 6 MENU STRUCTURE AND AUTHORIZATION **38**
CHAPTER 7 ORGANIZATIONAL CONSEQUENCES .... ERROR! BOOKMARK NOT DEFINED.
CHAPTER 8 TECHNICAL CONSEQUENCES **40**

Version control

Version	Activities	Date
Initial version 1.0	Draft version	06/02/2024

Remarks

Any changes and new developments that have a significant impact on the project proceedings will be noted here.

Introduction

Background and Context

This document is meant to provide an overview of the systems and data flow of the new features regarding SentinelOne EDR API integration to the QaaS app project. The project is carried out as a graduation project to Christopher Sulistiyo, a 4th year ICT student in NHL Stenden as a necessary requirement to achieve his diploma.

This project serves as the first proof of concept, making it the first attempt in the iteration.

Problem Statement

The company currently just purchased SentinelOne subscription alongside its Vigilance package and want to implement it to the QaaS app. Q-ICT currently already has 5 APIs to manage their customer's device health, finance, and Microsoft subscription, which namely are: Bodyguard, N-Central, SnelStart, PerfectView, and Pax8. The goal of integrating SentinelOne is to show more transparency to their customers, so that they can also know the information about their servers, devices, and computers.

Objectives

The goal of this project is to create a new functionality on top of an existing web application that can display all the necessary data coming from SentinelOne APIs. The data is then be represented in graphs, charts, or table. The graphs and charts themselves are customizable, whereas the table can sort, filter, and search the data based on user input. The application will also make use of stored session based on user's preferences on each user and store the session on the database (Firestore) securely with scalability in mind.

The integration to the QaaS app should also synchronize with the other 5 API data that in the system, whether they are stored in the company's Firestore database or not.

Additionally, the company also wishes to express the SentinelOne data to also collaborate with N-Central API data, in which it is an RMM (Remote Monitoring Management) application related to cybersecurity too, to manage clients' IT infrastructure, including servers, workstations, mobile devices, and network devices.

The project description is explained better in the Thesis of the author's graduation work placement project. For better information, please refer to the Chapter 1 of the Thesis.

Chapter 2 – Description of the Information System

System Overview

The product in question, the QaaS app, is an ERP-like web application used by Q-ICT and its clients. An ERP (Enterprise Resource Planning) application is a software platform that integrates the core of business processes and functions into a single system to streamline operations and improve efficiency. It facilitates cross-functional collaboration by centralizing data and automating workflows across different departments. The QaaS app enables Q-ICT to make data-driven decisions, improve productivity, reduce costs, and enhance customer satisfaction.

User Stories

High priority

	For clients	**For Helpdesk**	**For IT admin**	**Fo**
	As a customer, I want to be able to see the overview status of all my devices in my network so that I know which endpoints are infected with malware and which devices are healthy. As a customer, I want to be able to see the audit trail and the timeline of every threat detection and mitigation. As a customer, I want to be able to see detections and mitigations within a specific period (week/ month) per device. As a customer, I want to be able to customize the widgets responsible for visualizing the data shown from SentinelOne (which types of graphs or charts I would like to select, with adding and deleting the visualization widgets), so that I can have more control over what I see on the screen to help me understand the context more with my representation preferences.	As a helpdesk employee, I want the system to display more detailed comprehensive technical information about our clients' devices, so that I can facilitate effective and efficient monitoring to their health status and provide timely support. I should be able to see the health status of all the devices of the customers. As a helpdesk employee, I want the app to be able to access comprehensive threat response guidance in real-time from	As an IT admin, I want to have full control of my QaaS web application, by having the same read permission as the Helpdesk and being able to do POST, PUT, and PATCH requests.	As wa Vig off tim ser inc me ava eff ass ris

	As a user I want to be able to have filtering, sorting, and searching functionalities for the SentinelOne data shown in tables so that I can choose which data to be shown, so that I can understand the data more.	SentinelOne whenever a security threat is detected, so that I can promptly and effectively assist our clients. As a helpdesk employee, I want a system to automatically send an e-mail to me in case of a high severity unsolved cyber threat happens to one of our clients, so that I can be notified.		

Uses Cases

- **Device Health Monitoring**
    - Actor: clients
    - Use case: the QaaS app displays real-time health information about each device (e.g., laptops, workstations, servers) monitored by SentinelOne agents installed in those devices. This includes metrics such as installed applications, CPU core count, CPU version, RAM memory space, status of the health, is active or not, last active date, etc,
    - Precondition:
        - Client is logged into the QaaS app.
        - The client has devices installed and monitored by SentinelOne Agents EDR.
    - Main flow:
        - The client navigates to the SentinelOne dashboard page section of the QaaS app

        - The system retrieves the latest device health and status information from the SentinelOne EDR.
        - The system displaye device health and status in a user-friendly format.
    - Implementation: retrieve device health data from SentinelOne API and display it in a user-friendly format within the QaaS app dashboard or device management section.
    - Postcondition:
        - The client can view the health and status of their devices.
    - Alternative flow:
        - If the system cannot retrieve the device health and status information, if the data is null, if an error occurred on the server or on the client, it should handle the error and display the error message appropriately to the client (in a user-friendly way, without displaying too much technicality).
- **Threat Detection and Alerting**
    - Actor: the client, the IT administrators.
    - Preconditions:
        - The client is logged in into the QaaS app
        - The client has devices installed with and monitored by SentinelOne Agents EDR
    - Main flow:
        - The system monitors the devices for any changes in health or status.
        - If a significant change is detected (e.g., a device's health status changes from "Healthy" to "Unhealthy" or "Infected", the system sends an alert to the IT admin and the infected client)
    - Use case: SenntinelOne EDR platform detects a security threat (e.g., malware, ransomware, viruses, worms, spyware, etc.,) on a client's device.
    - Implementation: The threat will immediately be removed from the endpoint by SentinelOne and be listed and displayed in the dashboard as the detected threats that have happened for the Helpdesk and Security Experts of Q-ICT to analyse and document later. Optionally, a notification in the form of e-mail or from the QaaS should be delivered to the concerned user.

    - Postcondition:
        - The client is notified of any significant changes in their device's health or status.
    - Alternative flows:
        - The client can choose to silent or opt-out from receiving the alert notification.
- **Audit Trail and Incident Response**
    - Actor: the client
    - Precondition:
        - The client is logged in into the QaaS app
        - The client has devices monitored by SentinelOne Agents EDR
    - Main flow:
        - The client navigates to the Threats section of the QaaS app
        - The system retrieves the audit trail of threats from SentinelOne API
        - The system displays the audit trail in chronological order, with details of each threat.
    - Use case: a security incident occurs on a client's device, such as a successful malware infection or unauthorized access attempt.
    - Implementation: Retrieve audit trail data from SentinelOne via the API to track the sequence of events leading up to the incident. Present this information in a chronological order within the QaaS app, allowing administrators to investigate and respond to security effectively.
    - Postconditions:
        - The client can view the audit trail of threats that have occurred on their devices
    - Alternative flows:
        - If the system cannot retrieve the audit trail, it displays an error message in a user-friendly way.
- **Policy Management and Compliance**
    - Actor: the company
    - Preconditions:

        - The company has subscription to the QaaS app and has its own environment registered along with its clients.
        - The company has their SentinelOne agents installed to the desired endpoints.
        - The company has a certified cybersecurity expert to monitor those devices in case of a threat happens.
    - Use case: Q-ICT and its clients need to enforce security policies and ensure compliance with international industry regulations and standards regarding security (ISO 27001).
    - Implementation: With integrating real-time compliance-related data from SentineOne, such as security posture assessments and compliance reports, it will help Q-ICT and its clients demonstrate adherence to regulatory requirement.
    - Postcondition:
        - Q-ICT and its clients now have more protection on their IT systems from an EDR and XDR, which are an upgrade from traditional AV.
    - Alternative flows:
        - The clients of Q-ICT can choose to unsubscribe from SentinelOne through Q-ICT if they want to take a different EDR platform.
- **User-friendly Reporting**
    - Actor: the client, IT admin
    - Preconditions:
        - The client is registered by Q-ICT with SentinelOne subscription.
        - The client does not have to be logged in to the QaaS app.
    - Use case: clients want to view comprehensive reports on their device security posture and threat landscape.
    - Implementation: generate customizable reports within the QaaS app using data obtained from SentinelOne. These reports can include metrics such as threat detection rates, incident response times, and overall devices security scores. Ensure that the reports are easy to understand for clients with varying levels of IT and cybersecurity knowledge.
    - Postconditions:

    - The clients can now download reports in form of PDF or HTML.
  - Alternative flows:
    - Should display an error if failed to download reports

Acceptance Criteria

- **Real-time Threat Detection:** The QaaS app must integrate with SentinelOne EDR API to receive real-time alerts and notifications about detected security threats, including but not limited to:
  - Number of detected threats
  - Threat severity levels
  - Endpoint health status
  - Compliance posture
- **Threat Severity Identification:** The app should categorize each threat based on its severity level, incident status, and security verdict to prioritize response efforts.
- **User-friendly Interface:** For the clients, the UI should be intuitive and user-friendly, ensuring that the customers can easily navigate through page.
- **Customization and Flexibility:** The app should allow for customization of data visualization and response guidance based on the user preference, organization's policies, procedures, and specific client requirements.
- **Technical Data Display:** The system should display detailed technical information about each device, including but not limited to:
  - Endpoint name and category type (server, computer, laptop, workstation)
  - Operating system type and version
  - Hardware specifications (e.g., CPU, RAM, local storage type and details)
  - Network connectivity status (last activated date, IP address, is it connected to the internal network, last username who used it)
  - Installed applications and their versions (along with their dependencies and potential risks)
- **Health Status Indicators:** Each device entry should include health status indicators (e.g., color-coded icons) to quickly identify devices that require attention based on predefined criteria such as:
  - Connection status (offline/online)

    - Performance metrics (e.g., CPU usage, memory usage)
    - Security compliance (e.g., up-to-date AV definitions)
- **Filtering and Sorting:** The system should allow the users and personnel to filter and sort devices based on various criteria (e.g., customer name, device type, health status) to streamline monitoring efforts.
- **Real-Time Updates:** The system should update device information in real-time to ensure that helpdesk personnel and clients have access to the most current data.

MoSCoW Analysis

Must have	Should have	Could have	Would have
Showing SentinelOne data in tables	Search functionality	User session to store user's preference over the visualization they picked from the last session	Integration with N-Central API data
Visualization of data in the form of graphs and charts	Pagination functionality in tables to improve speed performance	Responsiveness to other screen resolution like on mobile with iPhone 11, Galaxy Note 20, etc,	Utilizing SentinelOne Vigilance package
Scheduled functions in Firebase to renew SentinelOne API key once it's about to expire	Sorting functionality	Unit test all the widgets of the SentinelOne pages	Dark mode, light mode
Securely store the API keys and other important information in Google Secret Manager	Filter column functionality	Utilizing Redis DB to improve querying speed	Different languages option (German, English, etc,)
Visualizing timeline of threats from SentinelOne data			Drag and drop functionality to widgets
Ensure proper permission for different User Tags			Resize the widgets' size (width and height)
Ensure that users from another company can only			

see the data specific of their own company			

Legend:

- Search functionality: utilizing Algolia AI powered search functionality (storing the data in and retrieving data from Firestore DB)
- Pagination: will make use of limit and skip functionality of SentinelOne API. In the page, there should be an array that stores the initial first n data retrieved and when the user clicks to show the next page, another API call should be called to for the next n data, and the array should store those as well. In the end when a user clicks to the end of the page, all the data has been retrieved and store nicely. When the user is at the end of the page all the data has been fetched and stored on the array, so when the user navigates back, there is no need to fetch data from the API anymore. This will be done in the back end from the Firebase cloud functions.
- Sorting: by column names and ascending or descending functionality of the table, making use of SentinelOne API onSort and sortOrder functionality

Chapter 3 Data Model

This chapter describes what data will be used in the project. It also shows how the data will flow and what are the relationships between each component in the system.

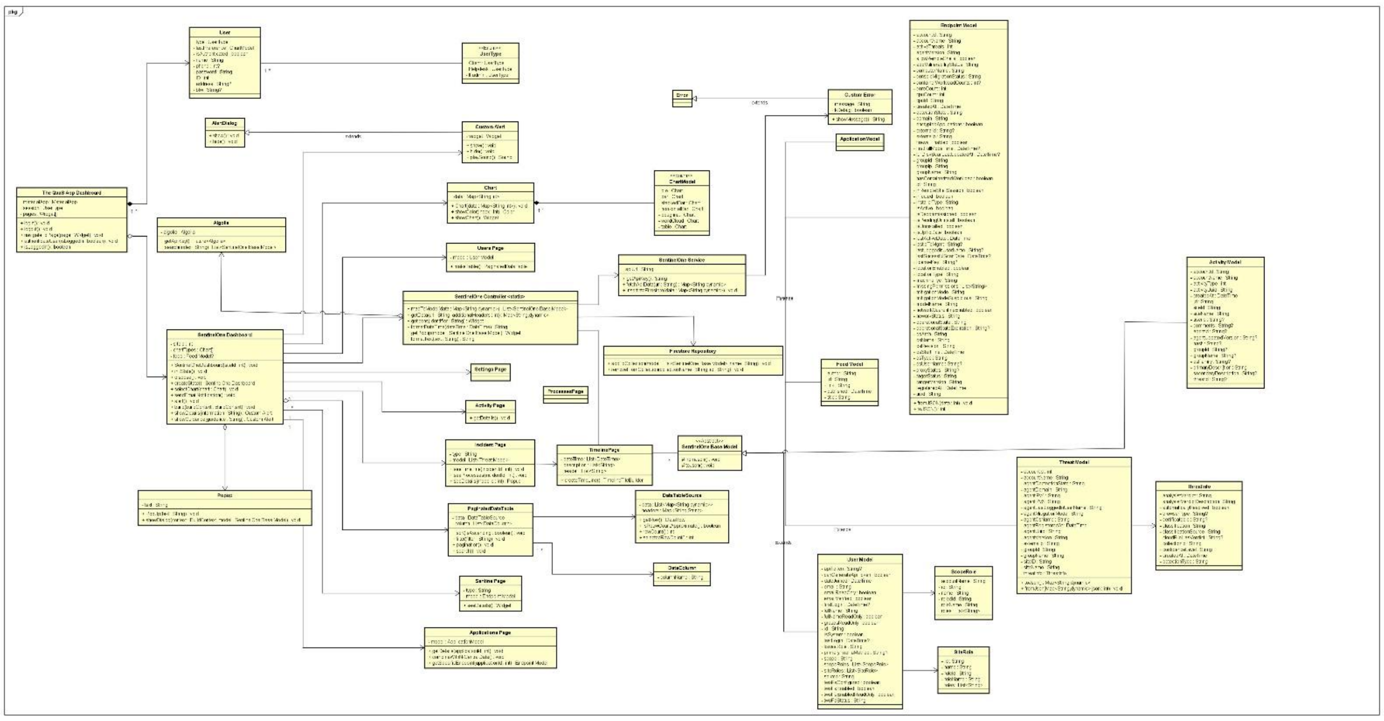
This project will utilize Firebase Firestore as its primary database. Firestore itself is a NoSQL document-oriented database. Because the nature of NoSQL databases that are designed to store data that do not have a fixed structure that is specified prior to developing the physical model, the focus is shifted on the physical data model. The developers who use NoSQL typically developing applications for massive, horizontally distributed environments. This puts emphasis more on figuring out how the scalability and performance of the system will work. But they also still need to think about the data model they will use to organize the data.

NoSQL DBs do not have a schema in the same rigid way that relational databases have a schema. There are 4 types of NoSQL database: document-based databases, key-value stores, column-oriented databases, and graph-based databases. NoSQL being the document-oriented database, typically store data in JSON, BSON, and XML format. Because the nature of SentinelOne API calls, only JSON and XML file format are the focus of the development. In a document-oriented database, documents (or items) can be nested, and elements can be indexed for faster querying.

The tables in relational databases are called collections in NoSQL database. They are the containers for documents that share a common structure or purpose. Unlike the traditional RDBMS, collections do not enforce a schema, allowing documents withing the same collection to have different fields or structures or types. Documents are the basic unit of data storage in NoSQL, and each is a JSON-like object that contain key-value pairs. Documents are stored within collections and represent individual records or entities. They can contain nested objects and arrays, providing flexibility for storing complex data structures. Each document would then consist of key-value pairs, where the key is a field name, and the value is the corresponding data that wanted to be stored. Key-value pairs are like rows in relational database, but with more flexibility in terms of data structure. The value can be various types, including strings, numbers, Boolean values, arrays, nested objects, and even binary data.

The most widely adopted NoSQL document-databases are usually implemented with a scale-out architecture. Providing a clear path to scalability of both data volumes and traffic.

Instead of ERD, this document will present Class Diagram to represent the structures and relationships of each entity and classes in this software development project, as well as DFD to visualizes the data flow between different types of users and the data sources/ sinks.



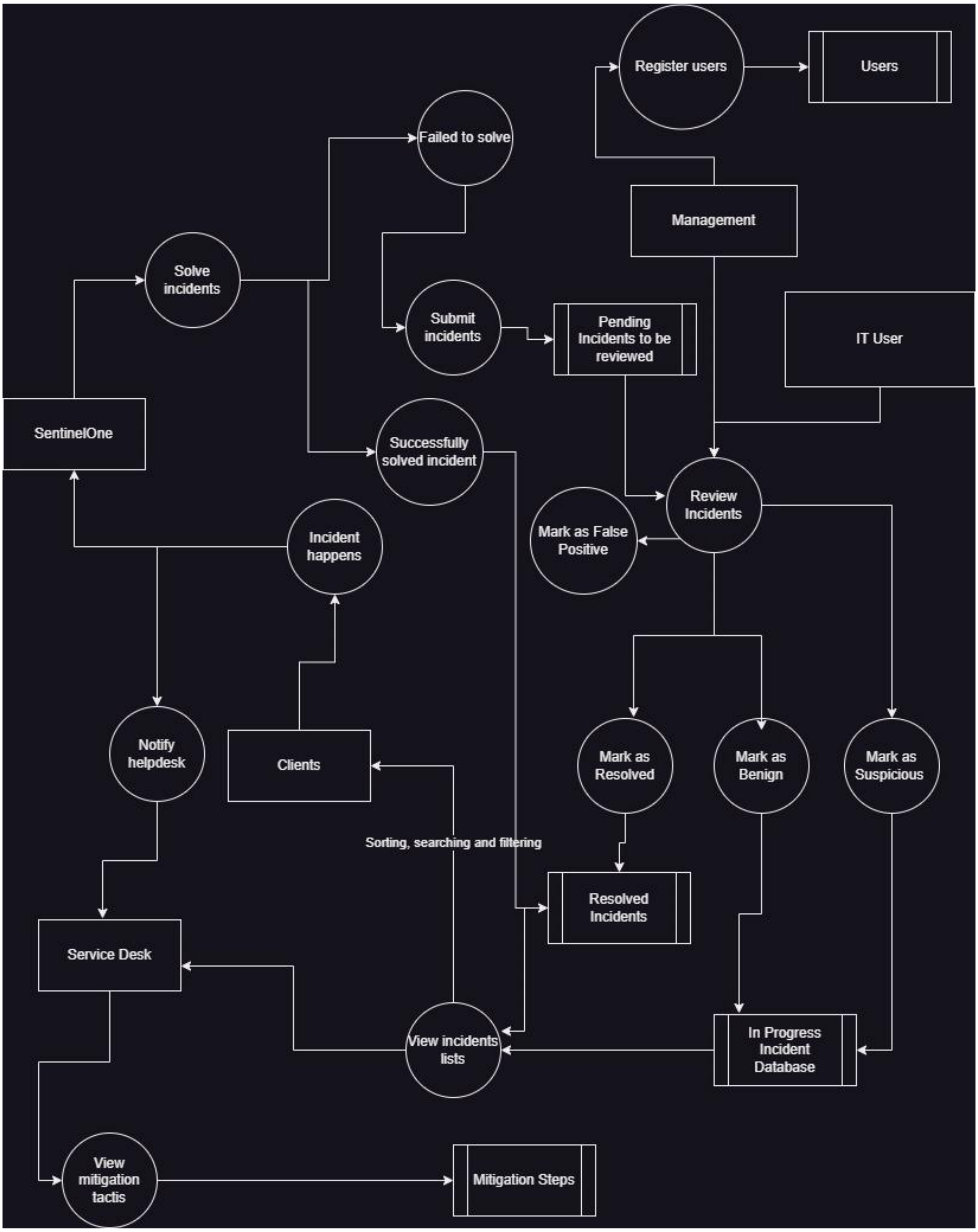


Figure 1 The DFD

Design Patterns used.

Flutter front-end

MVC (Model-View-Controller)

It is a software architectural pattern used usually for designing UIs and organizing code in applications. It promotes a clean separation of concerns, making applications easier to read, develop, scale, and maintain, while also promoting efficient code organization. It also supports modularity and reusability since each component can be developed independently of the others. MVC is widely used in web development (ASP.NET, Ruby on Rails), desktop applications, and mobile app development. The components that will be used in the graduation project are the following:

- Model: it represents the data and the business logic of the application. It encapsulates the data and behaviour of the application domain and responds to requests for information about its state (usually from the view) and instructions to change state (usually from the controller). It interacts with the database or any other data source to fetch or store data. It promotes relational modelling such as ORM for SQL, and ODM for NoSQL. It also is widely used in Repository pattern, where.
- View: represents the presentation layer of the application. It is responsible for displaying the data provided by the model in a user-friendly format. Views can be anything from a simple text output to a complex graphical UI web page. Views can receive data from the model and present it to the user, but they do not directly interact with the Model, only with the Controllers.
- Controller: it acts as an intermediary between the Model and the View. It receives user input from the View, processes it (potentially interacting with the Model to retrieve or modify data), and updates the View accordingly. The Controller interprets user actions (i.e., mouse clicks, keyboard inputs) and translates them into commands for the Model or View. Controllers control the flow of the application, orchestrating interactions between the Model and the View.
- Service: this directory is for the files that will be making calls to Firebase Cloud Functions on various types of triggers, such as HTTP Triggers (Get request), Firestore Triggers (insert, create, update Firestore documents), Callable Functions (On Call

function), Authentication Triggers, Cloud Storage Triggers, Pub/Sub Triggers, Analytics Triggers, etc.,

- Middleware: this directory is made for custom errors and loggings that the author might have when starting this project. The purpose of this customized error is for easier readability during testing and debugging.
- Utility: it provides functions or modules that are used commonly across the application. These utilities are often generic, reusable pieces of code that perform specific tasks or offer helper function to simplify development.
- Shared Widgets: this directory functions as a mix between the Utility and the View. It provides commonly used UI Widgets that are exported to classes or methods to simplify or shorten the code file in the View directory.
- Test: it contains the unit test files that are a crucial aspect of software development that ensures applications behave as expected, catches bugs in early development process, and maintain code quality over time. The test will include widget test to test the UI components, functional test that test the methods in Controller and Service, and Integration test that verify that different parts of the application work together correctly.

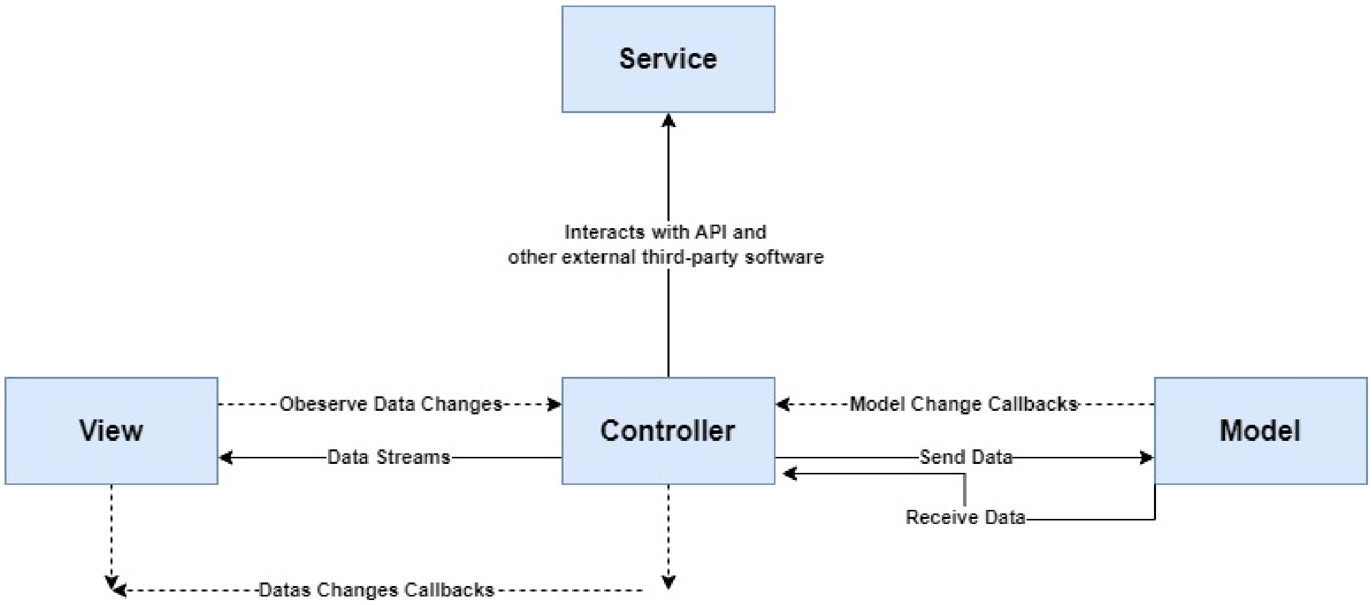
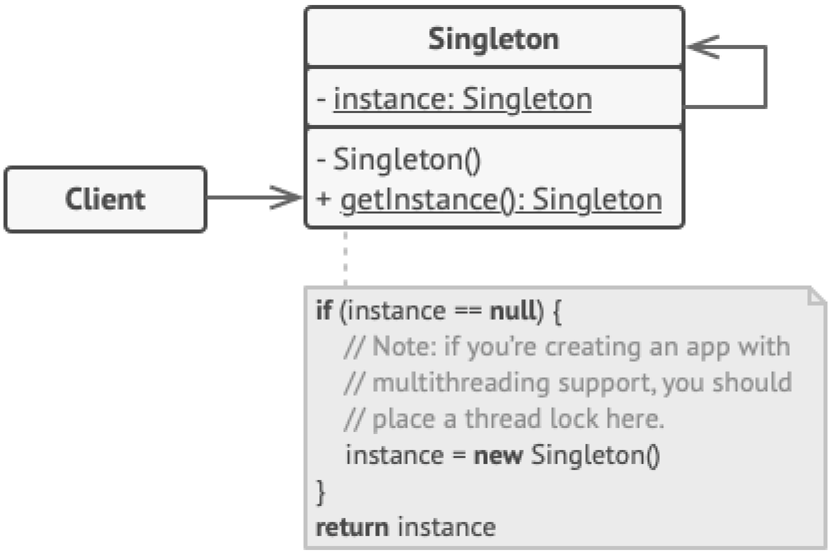


Figure 2 The MVC pattern in the Flutter front-end framework of the QaaS app

Singleton Pattern

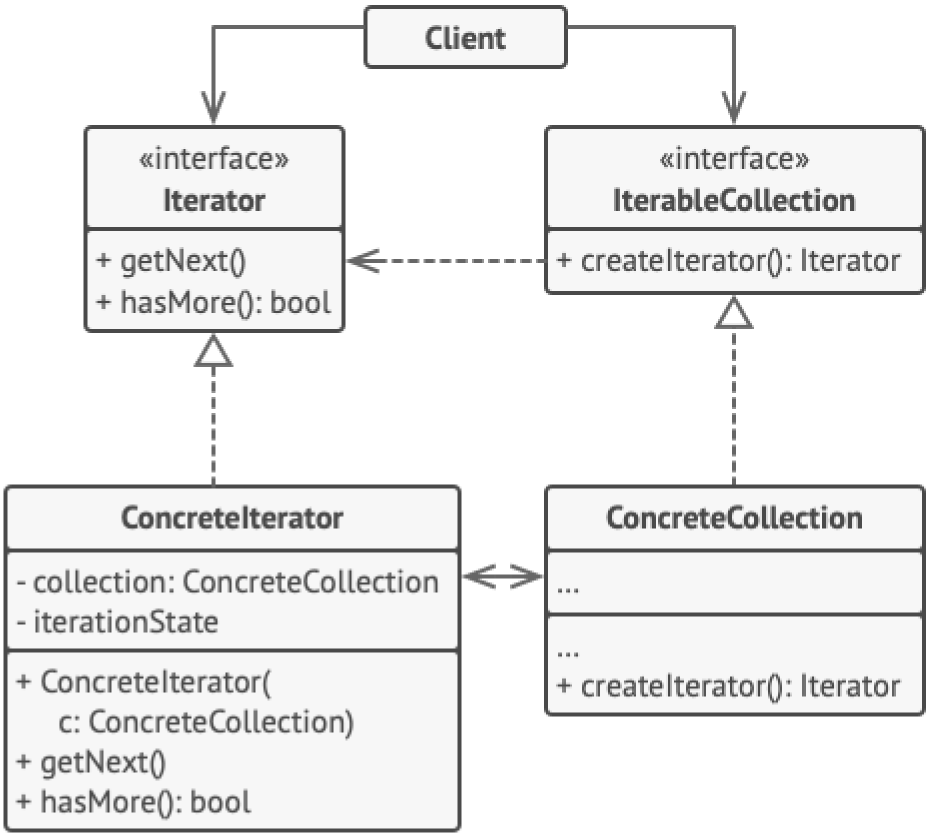
The Singleton Pattern will be used for the Service, Controller, and View files to make sure that they are only called once in the web application and will have to be terminated (clear out of memory) before initiating them again. This would be helpful especially when dealing with View files that contain multiple Widgets, as they need to be disposed first on close after the user leave the page to make sure that there are no memory leaks within the application and to maintain high performance optimization.



```
if (instance == null) {
   // Note: if you're creating an app with
   // multithreading support, you should
   // place a thread lock here.
   instance = new Singleton()
}
return instance
```

Iterator

It is a behavioural pattern that provides a way to access the elements of a collection sequentially without exposing its underlying representation. It mainly allows the developers to traverse through the elements of a collection one by one without needing to know the internal structure of the collection. This pattern will be used as SentinelOne data models will be represented in one abstract base model, and SentinelOne pages of the QaaS app will have many lists or collection of them, sometimes even more than one.



Node.js back-end

Application Element:

Controllers

Controller folders are responsible for storing files that contain the logic of the server. This does not necessarily mean to be files related to API calls, as they are already stored in the Service folder, but

Helpers

This folder is used to store functions that provide common functionality or assists in various tasks throughout the application. It may contain variety of use cases such as constants and enumerations, custom validators, helper classes, and utility functions such as API data validation, date formatting, encryption, String manipulation, etc.,

This folder is responsible for error handling utilities, including error formatting functions, error logging utilities, or custom error classes.

Routers

Routers are used for the files that contain the definitions of routes for handling incoming HTTP request. These routes determine how the server responds to different types of requests (e.g., GET, POST, PATCH, PUT, DELETE) on specific endpoints (URL paths). Because the nature of Firebase Cloud Functions to have allowing independent separation of functions, this directory will not be used a lot in this project, but it is good to have in the future use in case Q-ICT wants to have a Node.js Express server inside one of their Firebase Cloud Functions. The server, because it lives inside a cloud function, will have cold start latency, shorter limited uptime and resource limits compared to the traditional hosted Node.js Express server, but it may also have several upsides that are not yet explored.

Models

The Models contains the data models or schema definitions for interacting with database or other data storage mechanisms. These modes represent the structure and behaviour of the data within the application and are often implemented using ORM or ODM library or a database query builder. It will be used mostly for providing the abstraction later between the application logic and external services (SentinelOne and NoSQL Firestore database), making it easier to manage changes to the external services (such as changing the API version from the current version 2.0 of SentinelOne API, in case the JSON data changes), unit testing, and maintenance.

Middleware

The Middleware is a supplementary directory, serving a very close functionality to Helper folder. It will contain functions that access the request (`req`) and response (`res`) objects for authentication (verifying the authenticity of incoming requests by checking Firestore ID tokens, web sessions, or other authentication mechanisms), and authorization (whether a user has a permission to perform a certain action based on roles (clients, helpdesk, IT administrator)). It is a good to have (Could have in MoSCoW analysis for the application's request-response cycle.

Services

The Service folder contain files that are responsible for making API calls to SentinelOne. The purpose of this folder is made so that there is separation of error messages between the

Controller and the Service, to make it easier for the developer to debug the application when an API call went awry.

Views

This directory will be responsible for storing the files that contain HTML elements to show the JSON response to the user in the server. The files of this directory will not be a lot.

Config

This directory will contain the configuration files that are used throughout the back-end application. The purpose of this directory is very similar to the one in the Flutter front-end application, only this configuration files concern more information to the back end, like the Q-ICT SentinelOne domain name, and the secret names of all the Google Secret Manager secrets that contain the SentinelOne API keys.

Public

This directory will contain images in the form of SVG, JPG, PNG, or a custom-made HTML 404 (not found) or 401 (unauthorized) pages in the Node.js server to show to the user. In case of a malicious attacker trying to access the web server, it should always display the custom-made HTML pages.

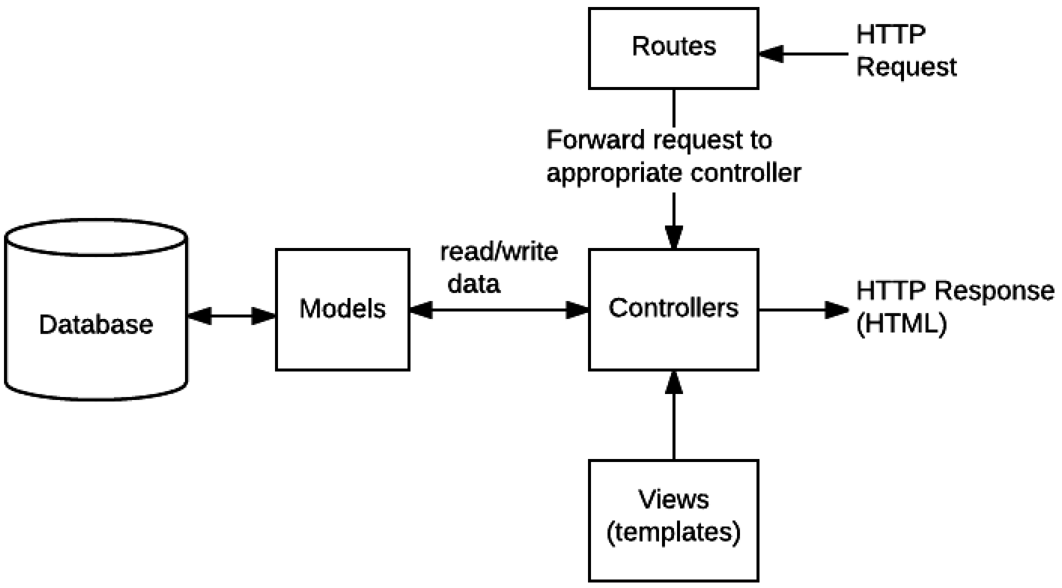
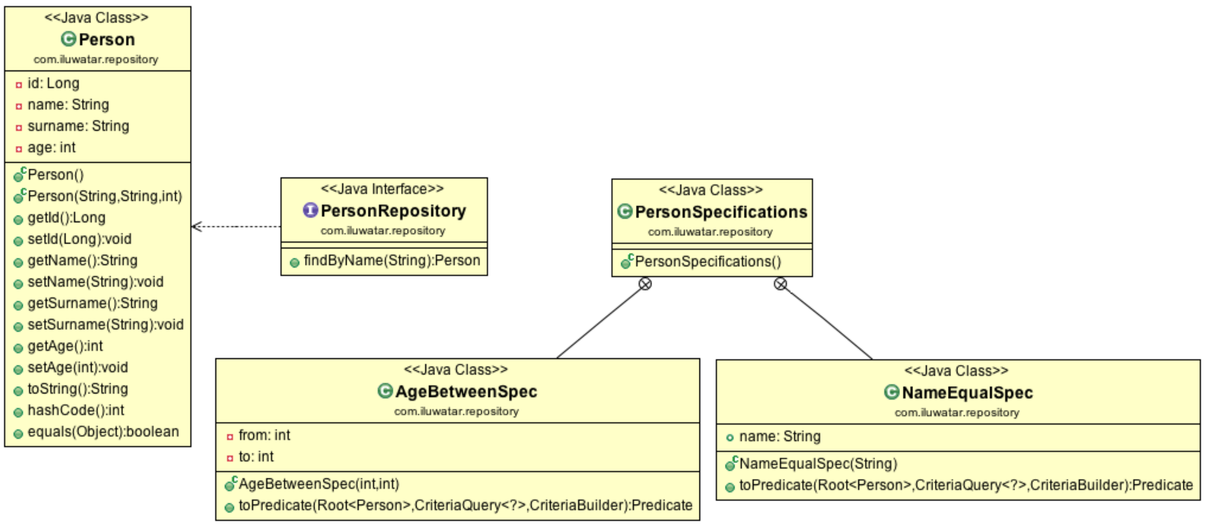


Figure 3 The back-end pattern in Firebase Cloud Functions of the QaaS app

Repository Pattern

This pattern provides abstraction layer between the data source and the business logic, making it easier to switch between different data sources (Firestore, SentinelOne APIs) without affecting the rest of the application. It provides a collection-like interface for accessing domain objects. This pattern is commonly used for managing the data flow when having to deal with large database and REST APIs. This pattern will be very powerful if combine with object mapping concepts like ORM and ODM.



ODM (Object Document Mapping)

It is a concept derived from ORM (Object Relational Mapping). It is a technique used in software development to bridge the gap between object-oriented programming and document-oriented databases. It is a way to map objects in the code to the documents stored in NoSQL database like Firestore. Flutter provides a package for this ODM to Firestore by using *cloud_firestore* and *algolia* for Algolia's mirror NoSQL database for the search functionality. In the back end, Node.js provides *firebase-admin* package that will be used for logging in and adding/deleting data to the correct collection.

User Interface and System Design

Wireframes

Sentinel One Dashboard

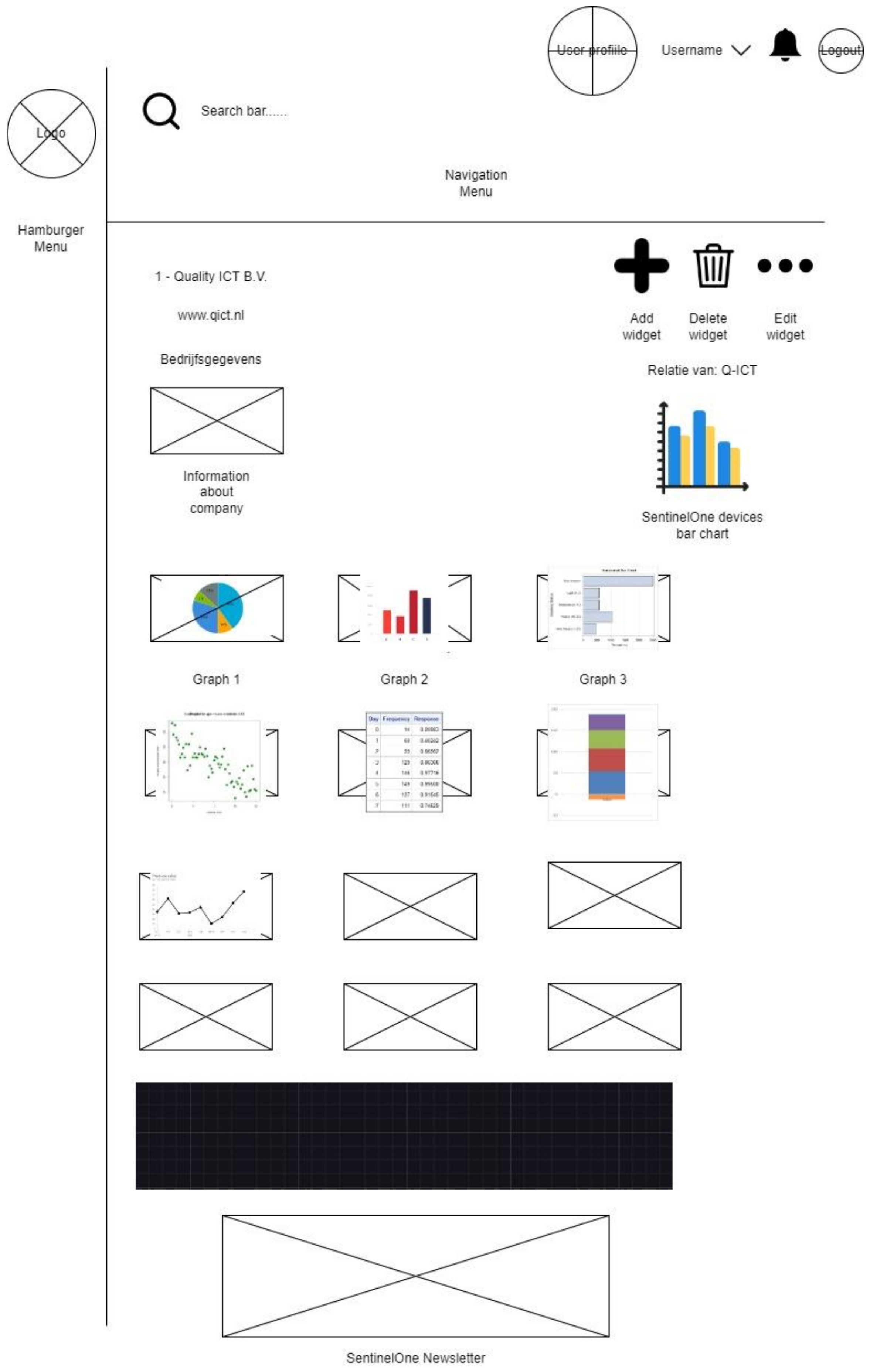

User profiile
Username
Logout
Logo
Search bar......
Navigation Menu
Hamburger Menu
1 - Quality ICT B.V.
www.qict.nl
Bedrijfsgegevens
Information about company
Add widget
Delete widget
Edit widget
Relatie van: Q-ICT
SentinelOne devices bar chart
Graph 1
Graph 2
Graph 3
SentinelOne Newsletter

Figure 4 Main SentinelOne Dashboard of the QaaS app with the customizable visualization widgets

Used SentinelOne API(s):

- Get Agents (filter: Site ID)
- Get Threats (filter: Site ID)
- Get Applications (filter: Site ID)
- Get Ranger Table
- Get Rogues Table
- Get S1 RSS Feed (general/ no filter)

Widgets:

- Pie chart
- Vertical bar graph
- Vertical stack bar graph
- Horizontal bar graph
- Line chart
- Scatter chart
- Word cloud
- Table

SentinelOne data specific page:

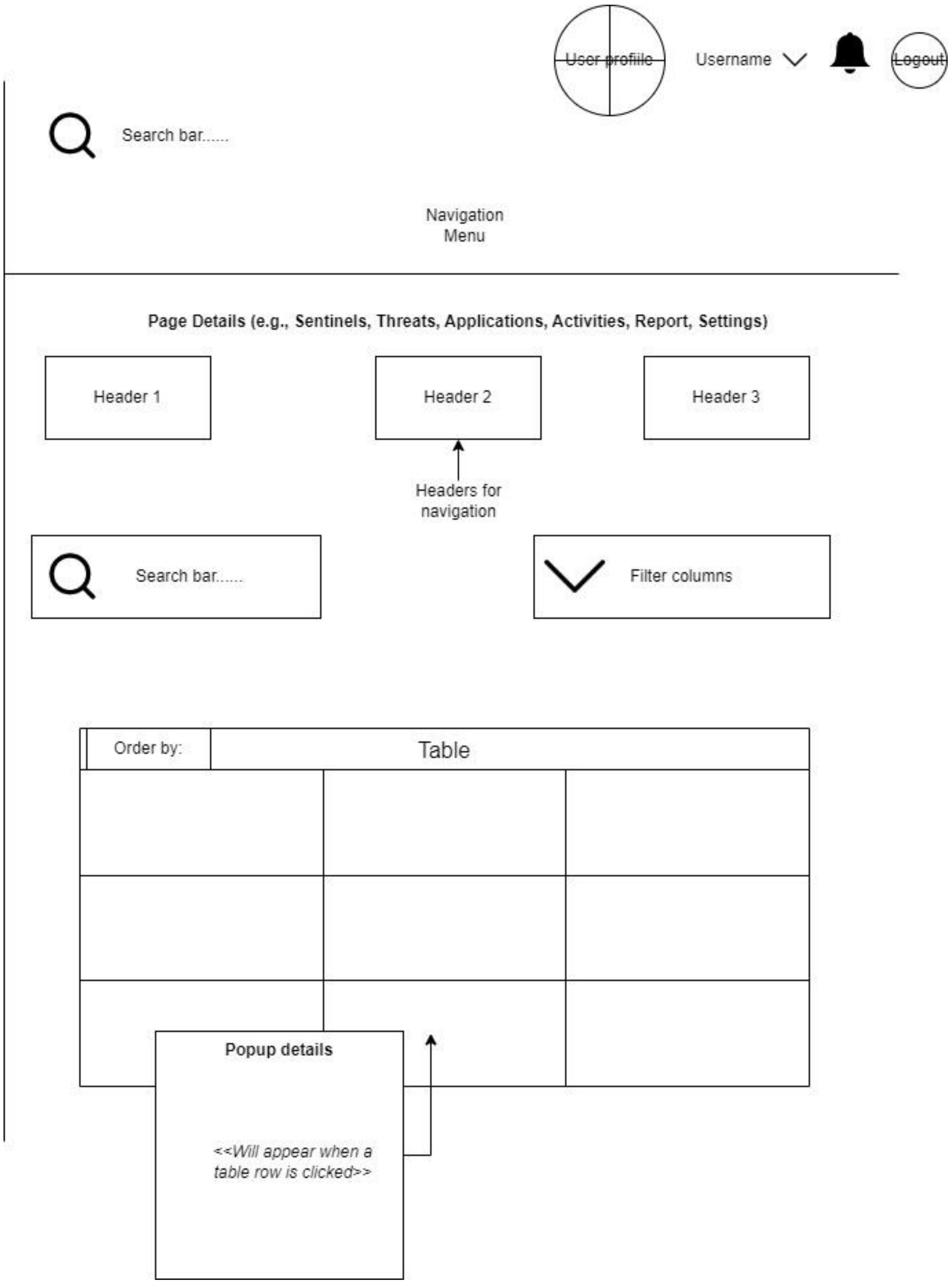


Figure 5 SentinelOne data specific page, will show popup with more details if a row of the table is clicked. WIll contain information about endpoints, threats, applicaitons, and activities.

Used SentinelOne API(s)

- Endpoints:
    - Get agents
    - Get endpoint applications (filter: agent id)
    - Get application process (filter: agent id)
- Threat
    - Get threats
    - Get alerts
    - Get threat notes (filter: threat id)
    - Get agents (filter: agent id)
- Application:
    - Get application with risks
    - Get aggregated applications with risks
    - Get application inventory endpoint
    - Get application inventory
    - Get application CVE
- Activity
    - Get activities
- Miscellaneous
    - Get latest SentinelOne news feed
    - Get customizable notes

Threat Processes Page

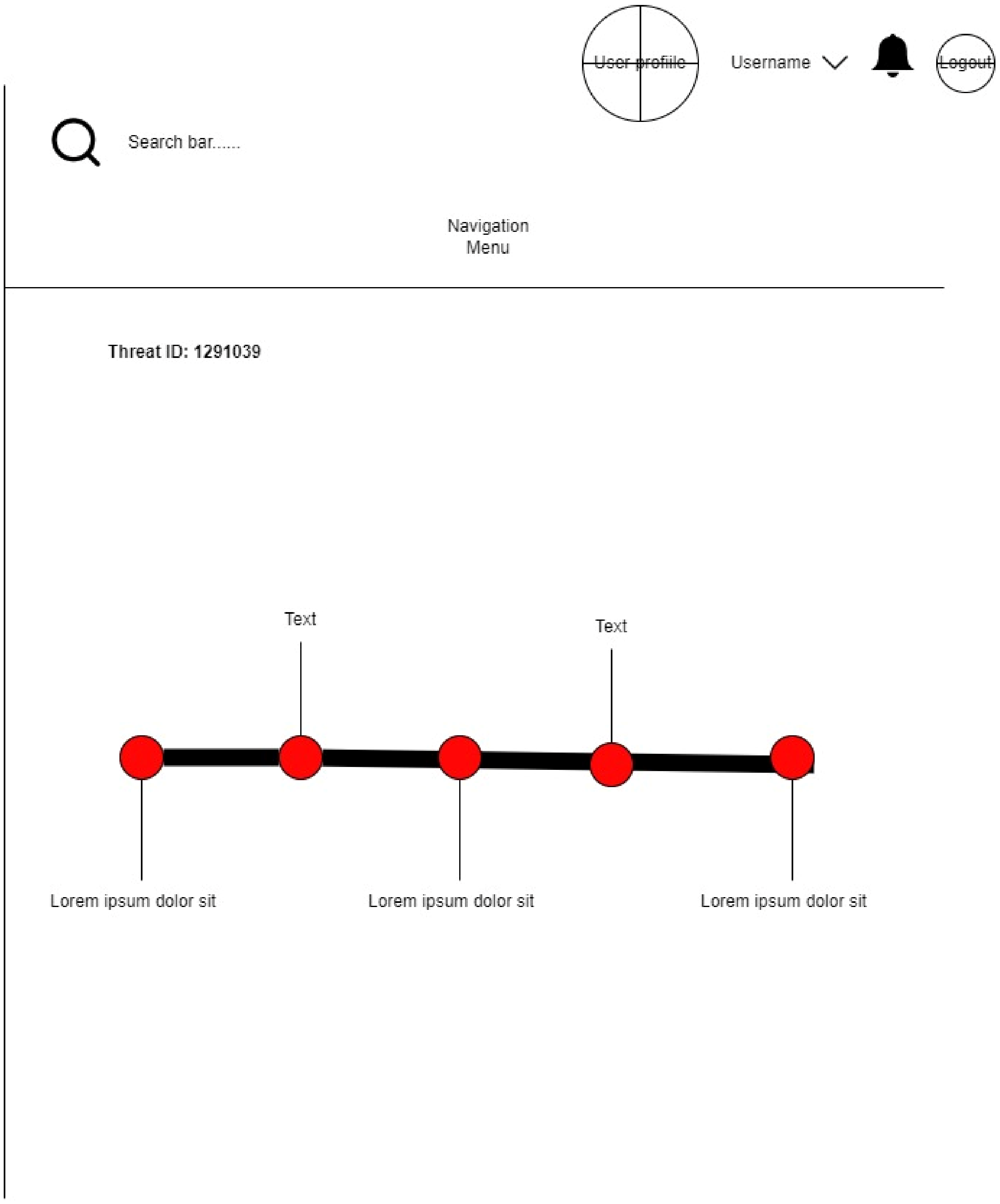
Used SentinelOne API(s):

- Get threat process (filter: threat id)

Widgets:

- Threat process widgets

Threat timeline page



Used SentinelOne API(s):

- Get Threat Timeline (filter: threat id)
- Get activity types

Widgets:

- Timeline widget

Settings Page

Used SentinelOne API(s):

- Get Global Policy
- Get Site Policy (filter by: Site ID)
- Get Group Policy (filter by: Group ID)
- List users

Mockups

Chapter 4 – Output

Any system is designed to produce an output, which is based on the input. In the tables below this output is detailed according to each entity and the input provided. It also goes into detail about what information users can expect to be able to get out of the system.

Code Output Product	**O-01**
Name	Overview of device status and health information
User	Clients, Helpdesk, IT admins
Objective	To provide information and export information about endpoint (on user request)
Frequency	Always present on the website and should be updated automatically once the page is refreshed
Data to be exported	Widgets (graphs, charts, table)

Code Output Product	**O-02**
Name	Information regarding a specific endpoint
User	Clients, Helpdesk, IT admins
Objective	To show more detailed information about an endpoint
Frequency	Should make an API call once every day to keep the system up to date (through cron jobs)
Data to be exported	JSON XML

Code Output Product	**O-03**

Name	General information about mitigated threats happened
User	Clients, Helpdesk, IT admins
Objective	To show all the threats that have happened on a specific Site (client's network)
Frequency	Every time the page loads
Data to be exported	Table

Code Output Product	**O-04**
Name	Specific information about a threat
User	Clients, Helpdesk, IT admins
Objective	To more detailed information about a threat that have happened in a specific Site
Frequency	Every time the page loads
Data to be exported	JSON XML

Code Output Product	**O-05**
Name	Threat timeline
User	Clients, Helpdesk, IT admins
Objective	To show the threat timeline and how it is handled by SentinelOne
Frequency	Every time the page loads
Data to be exported	Widgets (UI elements)

Code Output Product	**O-06**
Name	Activities
User	Clients, Helpdesk, IT admins

Objective	To show all activities that have been done in an endpoint for threat assessment
Frequency	Every time the page loads
Data to be exported	JSON XML

Code Output Product	**O-07**
Name	Conclude suspicious activity
User	Helpdesk, IT admins
Objective	To summarize and conclude when the threat happened based on the infected endpoint activities for incident identification
Frequency	On user’s input
Data to be exported	JSON XML

Code Output Product	**O-08**
Name	Installed applications
User	Helpdesk, IT admins
Objective	To show all installed application on a specific endpoint
Frequency	Every time the page reloads
Data to be exported	JSON XML

Code Output Product	**O-09**
Name	Risky outdated installed application
User	Helpdesk, IT admins

Objective	To show the users what are the risk in their environment by showing outdated applications that may impose potential risks
Frequency	Every time the page reloads
Data to be exported	JSON XML

Code Output Product	**O-10**
Name	Application Inventory
User	Clients, Helpdesk, IT admins
Objective	To show application inventory to the users
Frequency	On user’s input
Data to be exported	JSON XML

Code Output Product	**O-11**
Name	Settings
User	Helpdesk, IT admins
Objective	To show SentinelOne settings to the user
Frequency	On page reload
Data to be exported	JSON XML

Code Output Product	**O-12**
Name	Reports
User	Helpdesk, IT admins
Objective	To show monthly overview reports of the environment that the employee of Q-ICT or SentinelOne has made automatically to the users

Frequency	On page reload
Data to be exported	JSON XML

Code Output Product	**O-13**
Name	Download reports
User	Clients, Helpdesk, IT admins
Objective	To download reports from the dashboard
Frequency	On user's input
Data to be exported	PDF

Code Output Product	**O-14**
Name	E-mail notification and web alert
User	Helpdesk, IT admins
Objective	To notify the associate user in case of a cyber threat happening
Frequency	Every time a cyber threat happened on a client machine
Data to be exported	Emails, web alert, and sound

Code Output Product	**O-15**
Name	Cyber threat emergency guidance
User	Helpdesk
Objective	To inform the helpdesk about the instructions regarding what actions can be undertaken in case of an unmitigated cyber threat happening on a customer/company computer from SentinelOne
Frequency	Every time a cyber threat happened on a client machine

Data to be exported	Web pop-up

Code Output Product	**O-16**
Name	Newsletter
User	Clients, Helpdesk, IT admins
Objective	To show to the users the SentinelOne newsletters from the API
Frequency	Every time the page reloads
Data to be exported	JSON XML

Chapter 5 Required Input

The aim of this chapter is to show the desired input. It will consist of a table for each function, that is receiving input. The table will have these categories: Code Output (a unique code for every function), Name (name of the function), Authorization (the users that will have access to use this function), Objective (the goal of the function, what data is it supposed to received), Description (where the data will be inputted on the screen), Frequency (how often will the input be put in), Screens Used (which parts of the web application (tabs) will be used when entering the input).

Input Task Login

Code Input Task	**Input Task IT-01**
Name	User ID
Authorization	Clients, helpdesk, and IT admins
Objective	To log in into the QaaS app and present the system with the ID for give it the right authorization
Description	The following data must be inserted into the login screen: • Username (email). • Password. • TOTP code sent through mobile phone. If the user does not have a login credentials, that means they are not an official user/client of Q-ICT, and therefore should contact the IT admins for help
Frequency	Anytime a user wants to log in. The credentials can be saved in a cookie. When the user closes the browser or log off their device, the application should automatically log them off.
Screens used	Login Screen

Code Input Task	**Input Task IT-02**
Name	Site ID
Authorization	Clients
Objective	To give the system with the Site ID. It is required for the client user only, as they can only see security information and status of their own devices.
Description	In SentinelOne, “Sites” refer to **logical grouping** of endpoints (devices or systems protected by SentinelOne), within an organization’s network infrastructure. Sites are typically organized by SentinelOne based on geographical locations, departments, or other criteria that makes sense for the organization’s structure and management needs.
Frequency	Whenever the client wishes to navigate to SentinelOne page.
Screens used	QaaS app Dashboard Screen

Code Input Task	**Input Task IT-03**
Name	Incident/Threat ID
Authorization	Clients, helpdesk, and IT admins.
Objective	To be able to specify a threat to view more detailed information
Description	To be able to show each threat more detailed information along with the timeline on how that threat is mitigated by SentinelOne, the app will take the user to a different page.
Frequency	Once per user’s request
Screens used	SentinelOne Threat Timeline Page

Code Input Task	**Input Task IT-04**
Name	Agent ID

Authorization	Clients, helpdesk, and IT admins
Objective	To pass in the Agents ID, to view the device information in more detailed
Description	Agents in SentinelOne refers to the AV itself that is installed in a specific endpoint to provide comprehensive security protection. These agents utilize the behavioural AI and continuous monitoring; therefore, it is crucial that the Agents get access to the highest permission possible to give real-time information to the central system.
Frequency	Once per user's request
Screens used	SentinelOne Device page

Code Input Task	**Input Task IT-05**
Name	Input search/filter/ sort
Authorization	Clients, helpdesk, IT admins
Objective	To filter, search, and sort the data in either the front-end/back-end.
Description	The data which will be displayed in tables, need to have functionalities such as searching, filtering, and sorting based on various criteria (customer name, device type, health status) to streamline monitoring efforts.
Frequency	Once per user's request
Screens used	SentinelOne Overview page

Code Input Task	**Input Task IT-06**
Name	Visualization type
Authorization	Clients, helpdesk, and IT admins
Objective	To be able to select different visualization types of the data displayed (pie chart, bar, graphs, etc,)

Description	The app will visualize the SentinelOne API data (whether it is about the endpoint or threat information) to make it easier to analyze
Frequency	Once per user’s request
Screens used	SentinelOne Overview page

Chapter 6 Menu Structure and Authorization

This project will have 3 different layers authorization: the Customers, the Helpdesk, and the Developers. Because of the security compliance regulation as the intern is not a full-time employee of the company (as stated in the Chapter 2 in Thesis), this project will only handle the GET request (Read operations) of the API call and nothing more than that, as any additional request would have a substantial adversity effect on the system.

The clients have the most basic authorization and authentication across the platform. They need to be registered on the Q-ICT database, therefore making them an official client, and register their user email, password, and phone number to the system. Once registered and the phone number verified, they can log in to the QaaS web application by entering their registered email username and password. Once they put in the right information, an OTP verification code will be sent to their registered phone number, thus ensuring MFA. When viewing the SentinelOne page, the clients can only view their own device security information given by SentineOne. The page

The helpdesk has more rights and authority than the client user. Once logged in with 2FA, they can see the overview of all the client's device health and status.

The IT user has the most rights in the QaaS app where they can update the modules and templates of the QaaS app, limiting what the user can see. The IT point of view will be mostly out of scope for this project, as the author will have limitation of access in both the QaaS app permission and SentinelOne requests.

***Please note** that the scope of this project, as defined in the Thesis will not include doing POST, PUT, PATCH, and DELETE requests from the API. This table represents the ideal complete implementation of the SentinelOne integration and will treat the author to have a full access to the API. Otherwise, the whole entirety of the table will only include Read operation.*

Main	**Sub-menu**	**Authorization**		
		Clients	Helpdesk	IT admins
Dashboard Page	Incident Page Endpoint Page	C, R, U, D	C, R, U, D	C, R, U, D

Incident Page	Incident Detail Page Alert Page	R	R, U	C, R, U, D
Endpoint Page	Application Page Endpoint Detail Page	R	R	R
Alert Page		R	R	R
Incident Detail Page	Incident timeline	R		
Activity Page		R	R	R
Application Inventory Page		R	R	R
Policy Page		R	C, R, U, D	C, R, U, D
Settings Page		R	C, R, U, D	C, R, U, D
Helpdesk page		N/A	C, R	C, R

Chapter 8 Technical Consequences

Are any extra workplaces required, and if so, with which technical facilities?

The author will use Q-ICT's official office as his main workplace along with other premises such as NHL Stenden University Emmen campus, the Library Emmen, and the author's own office in his room if some situations desire the author to not to be there. A supply of secure and good internet connection will be provided by Q-ICT, in the form of an internal network connection, linked to his docking stations. Additionally, the company will provide 2 additional ultra-wide screens to the author's workstation, with additional resources available for disposal upon request.

Which special other technical equipment is required? (E.g., bar code equipment)

Besides the customer's own device that they want to install the SentinelOne agents on, there is no special equipment required.

On which software or stand-alone computer will the software be used?

The project itself is a continuation of a pre-existing infrastructure, if said infrastructure is missing or not up to the required specifications, an update will be recommended. There should not be a need to purchase any other server or equipment, as the new website should be able to run on the same server that the existing platform is currently running, and therefore it will be using the same domain name.

Which internet facilities are required for the software?

The product should work on any cross-platform web browser, but in order the user to get the best performance and all its existing functionality, Microsoft Edge and Google Chrome is recommended by the author, as Flutter and Firebase is powered by Google. Another thing that should be taken into consideration is constant internet connection to be able to use the application, as some of the pages use StreamBuilder in which it requires constant internet connection to streamline the data. For the system to run perfectly and qualitive enough to fulfil its purpose, users are required to have at least 250Mbps connection with below 10 milliseconds response time to the web server.

Is special system software or other technical equipment required to run the software?

No, the user only needs to be registered as an official Q-ICT customer by subscribing to one of our offered subscriptions and have an Internet connection to access it.

How will the system be supported?

The back-end code will be hosted by Google in the form of Firebase Cloud Functions, as the data communication facilities. The QaaS app itself is also hosted by GCP (Google Cloud Platform), and the author will be working on the test environment. Once the company sees that his work has fulfilled as expected with the stakeholder's requirements, it will be moved to the live environment in which it can be accessible to Q-ICT and its clients to access on the Internet.

Which data communication facilities are required?

E-mail addresses, passwords, and phone numbers are required for creating an account.

Which (special) printers are required or is a network printer sufficient?

No printers are required to access the web application.

Which back-up facilities are required? Are separate back-up facilities required or can the back-up be included with the other data from the network?

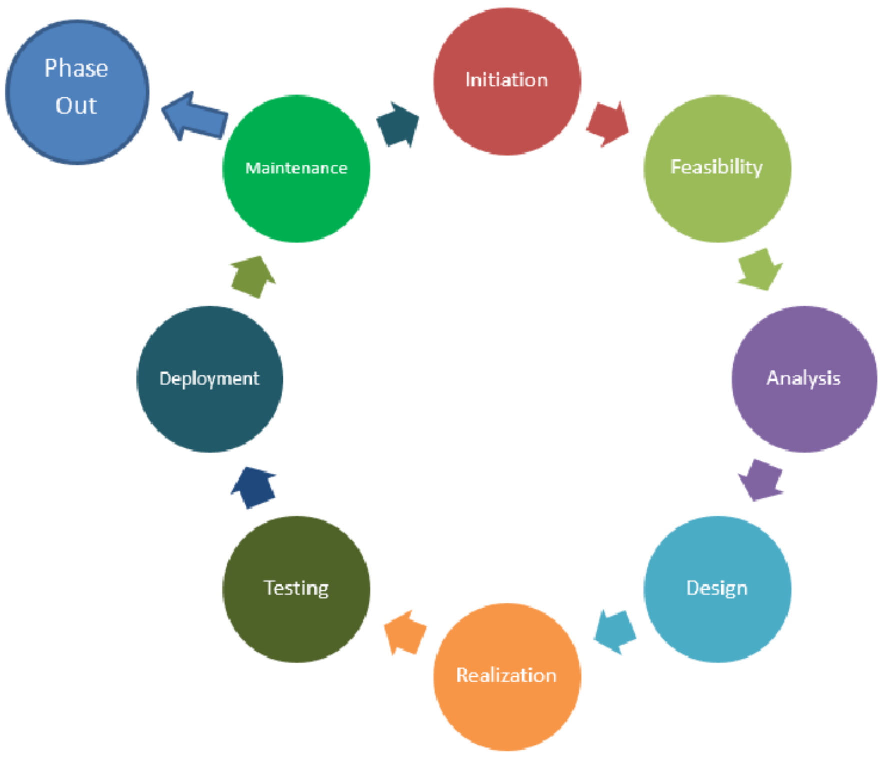
The Firestore DB from Firebase will serve as the back-up facilities in case of data loss.

Are extra security measures required for data protection?

Upon logging in (authorization) the app will do a Captcha check to make sure that the user is a human and not a bot.

Who will take care of implementation and maintenance?

The maintenance of the final product is left to the Q-ICT software development department. In the end, the author ought to produce a comprehensive documentation (in the form of README.md and code commenting or User and Developer Manual), as the means of product usage, counsel, and recommendation for the Q-ICT official developers and upcoming interns to ensure its continuity. However, the author will not be taken into any account regarding the maintenance or phasing out of the product beyond completion as specified in the graduation project module-book (*See Graduation Project Manual Information Technology 2023-2024 Appendix C*).


Phase Out
Initiation
Feasibility
Analysis
Design
Realization
Testing
Deployment
Maintenance