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

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 and Uses Cases

For customers

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 devices.

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 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.

For the helpdesk

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 SentinelOne whenever a security threat is detected, so that I can promptly and effectively assists our clients.

As a helpdesk employee, I want to system to automatically send an e-mail to me in case of a high severity unsolved cyber threat happened to one of our clients, so that I can be notified.

For IT user and Q-ICT

As a cybersecurity consultant, I want to leverage SentinelOne Vigilance package to be able to offer my customers a 24/7 real-time proactive threat mitigation services and checks that is included for showing real-time metrics and mitigations that are available so that we can effectively safeguard their digital assets and mitigate cybersecurity risks.

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.
- **Guided Response Workflow:** Upon receiving an alert (whether by e-mailing the privileged users or by displaying a notification with a sound in the dashboard), the app should be able to present a guided workflow outlining step-by-step instructions on how to respond to the specific type of threat detected.
- **Interactive Response Options:** The guidance provided should offer interactive response options, including mitigation measures, containment strategies, and escalation procedures.

- **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.

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 DFD to visualizes the data flow between different types of users and the data sources/ sinks.

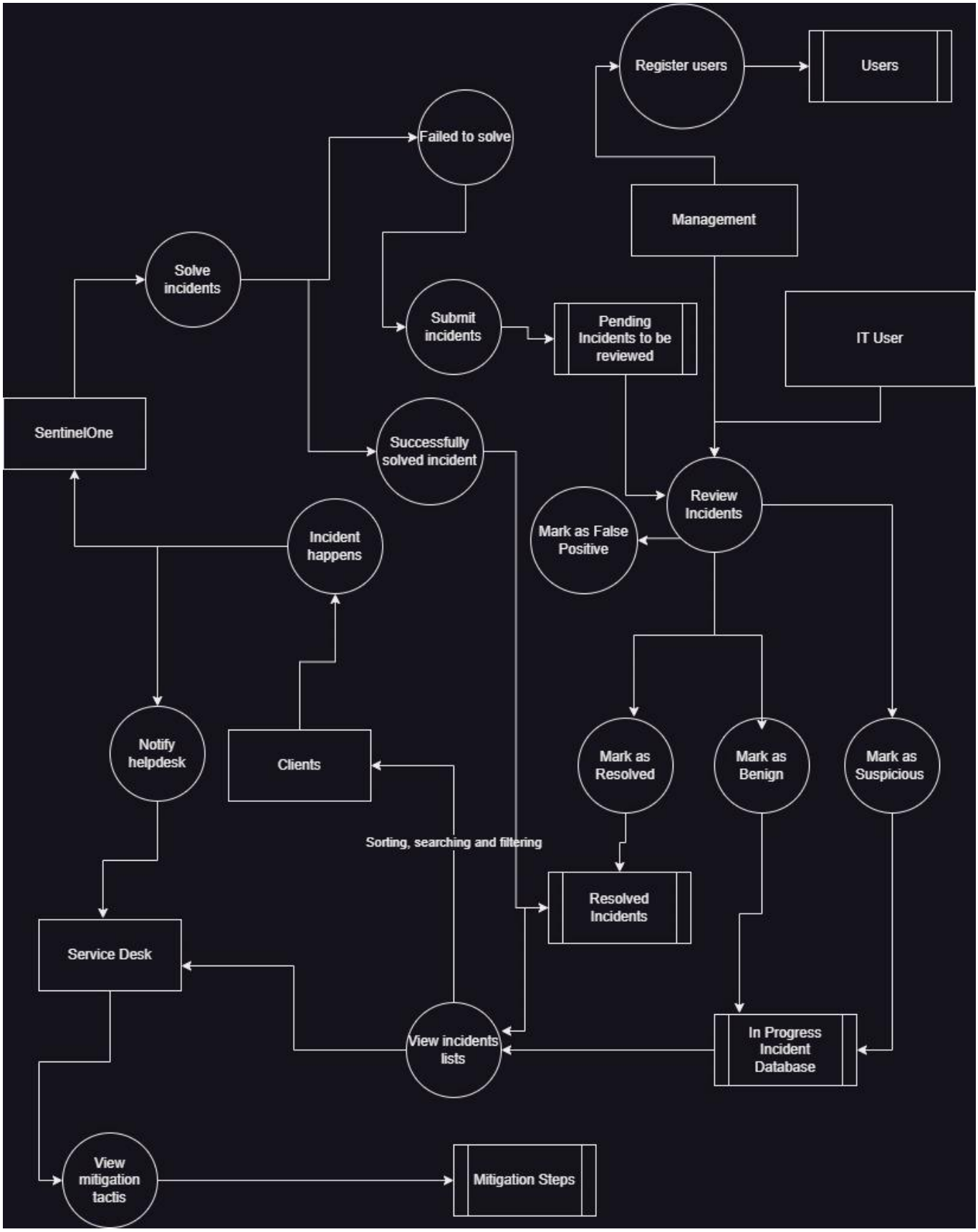


Figure 1 The DFD

Schemas

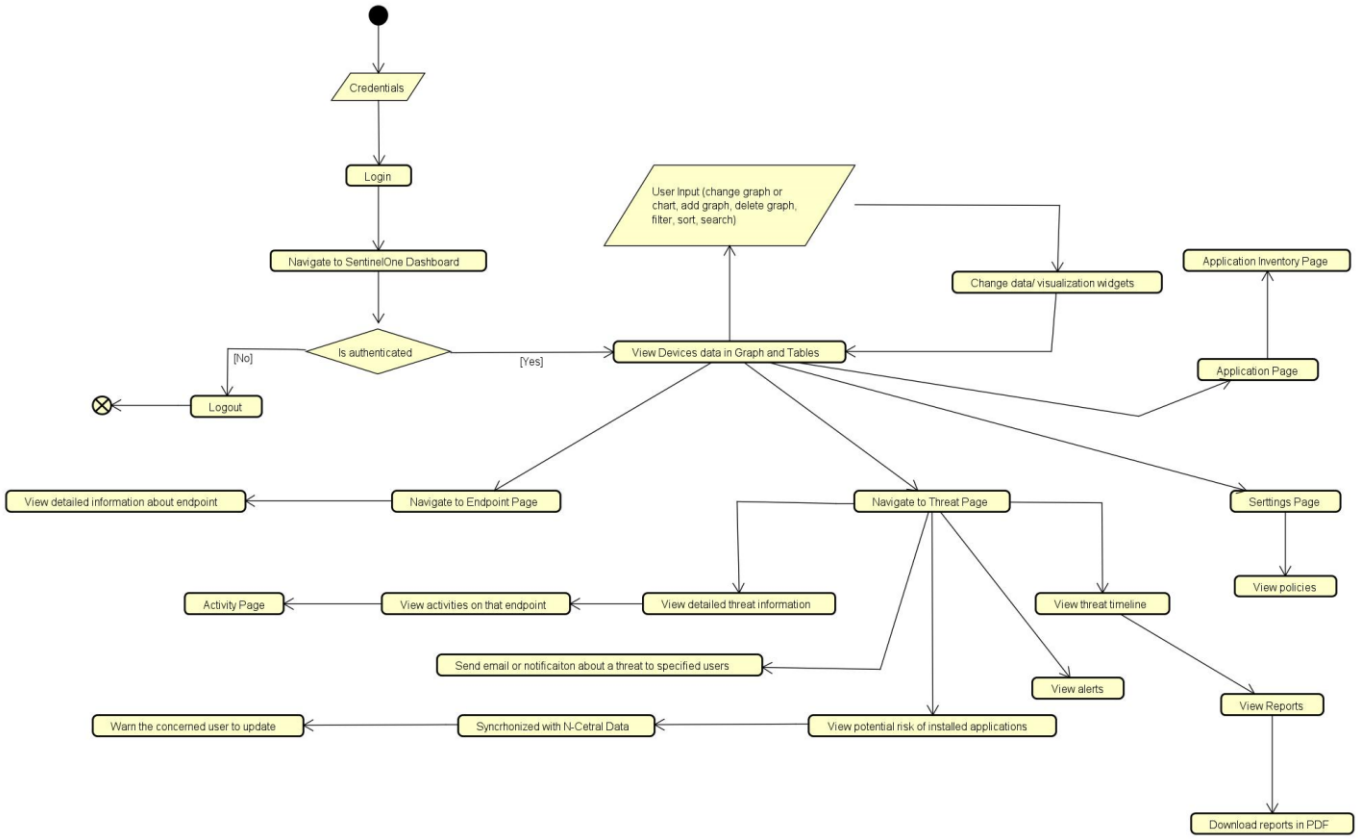


Figure 2 Flowchart Diagram

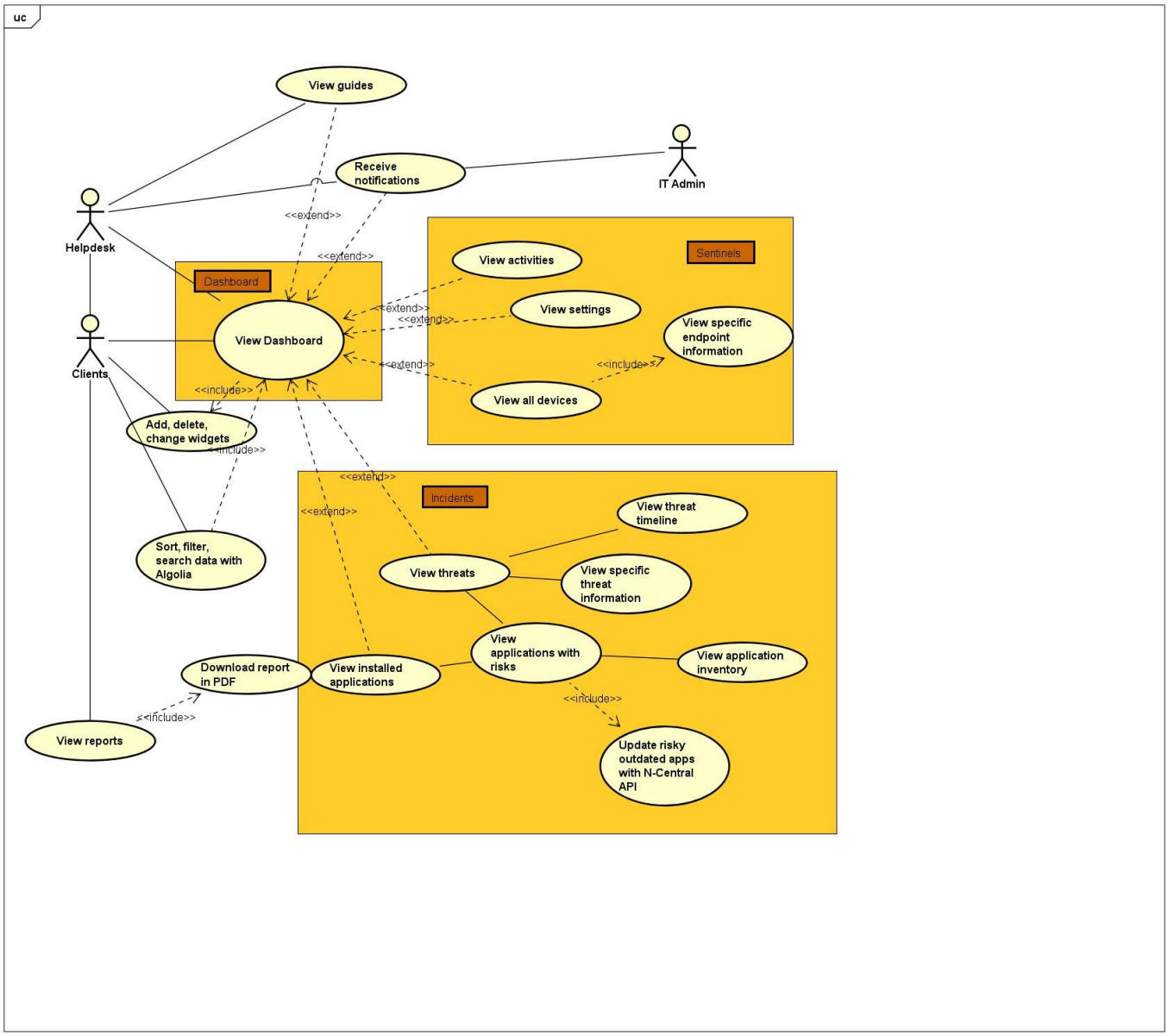
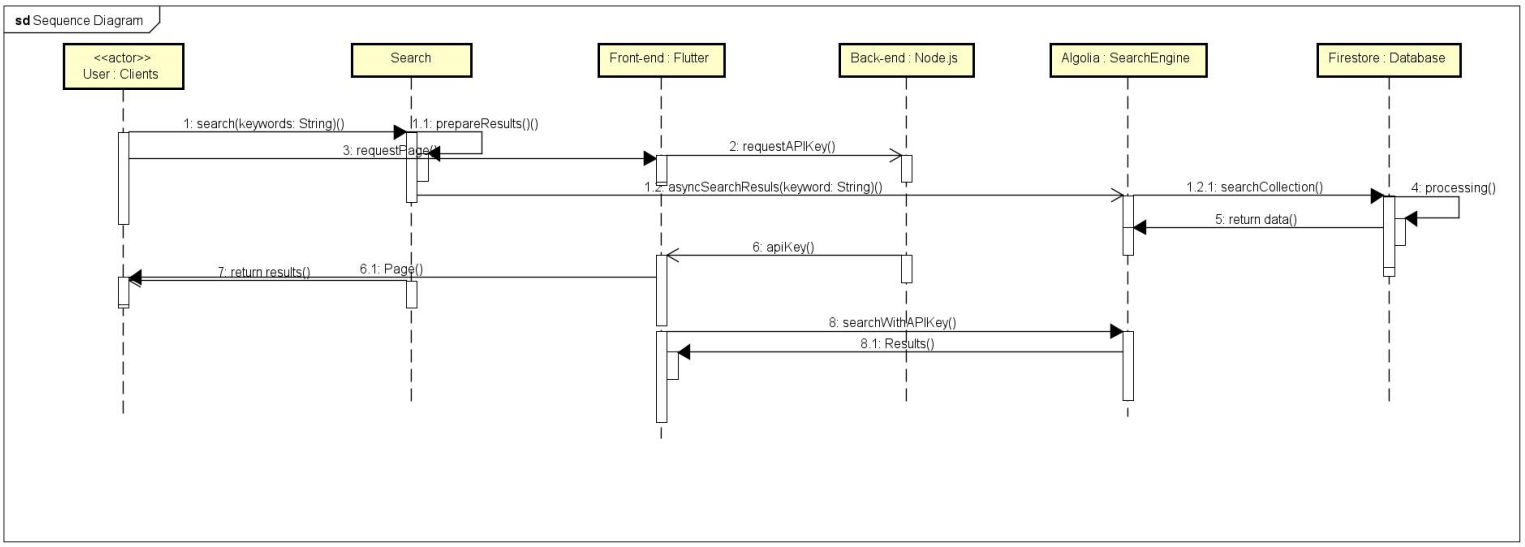
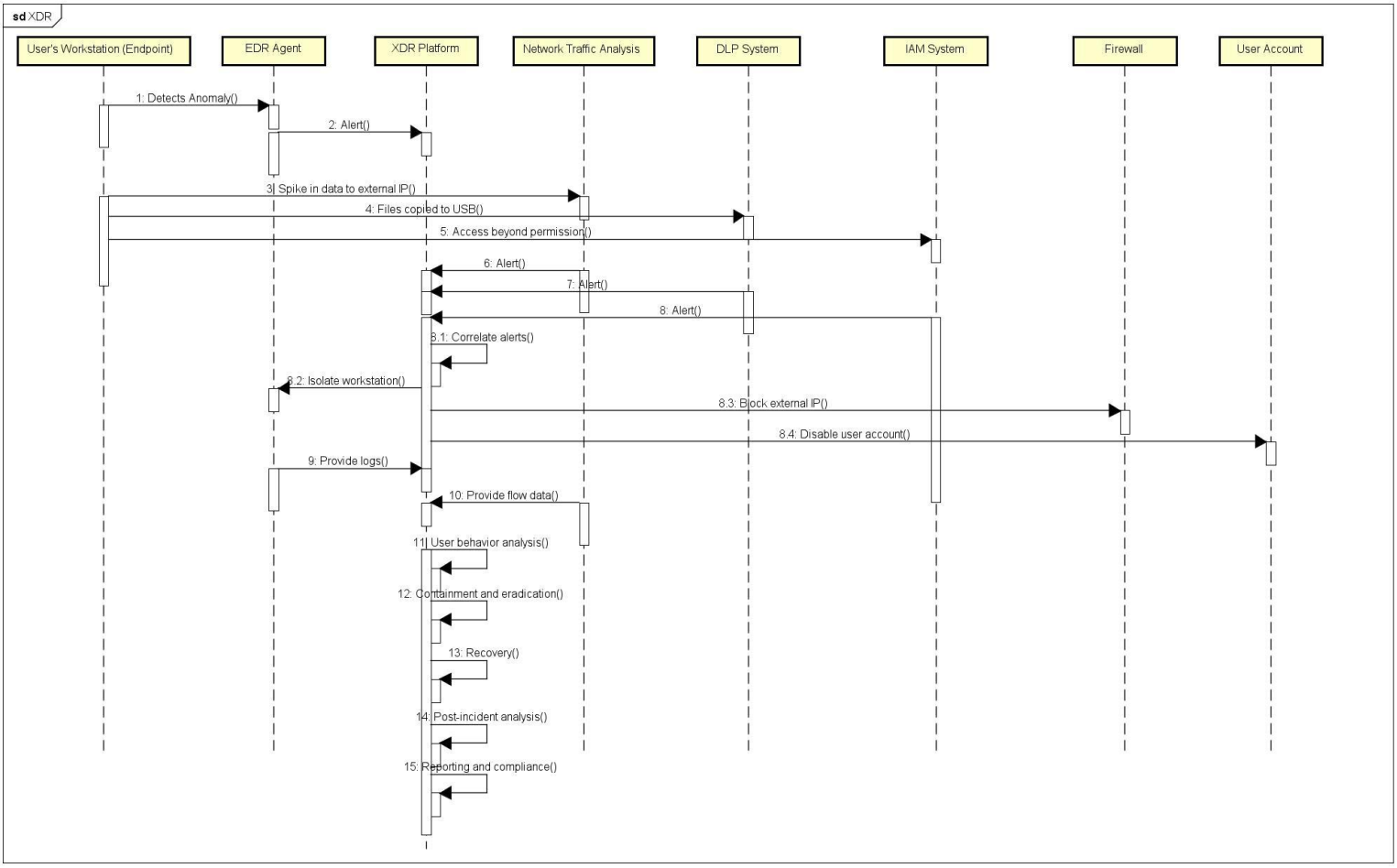


Figure 3 UseCase Diagram





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.

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.

Observer Pattern

This pattern will be useful, especially when dealing problems that may occur with sending e-mail and alert notifications in case of a cyber threat happening on a client's machine. It will guarantee the handling of events or updates in real-time, making it easier for the application to be aware when certain changes occur.

Abstract Factory Pattern

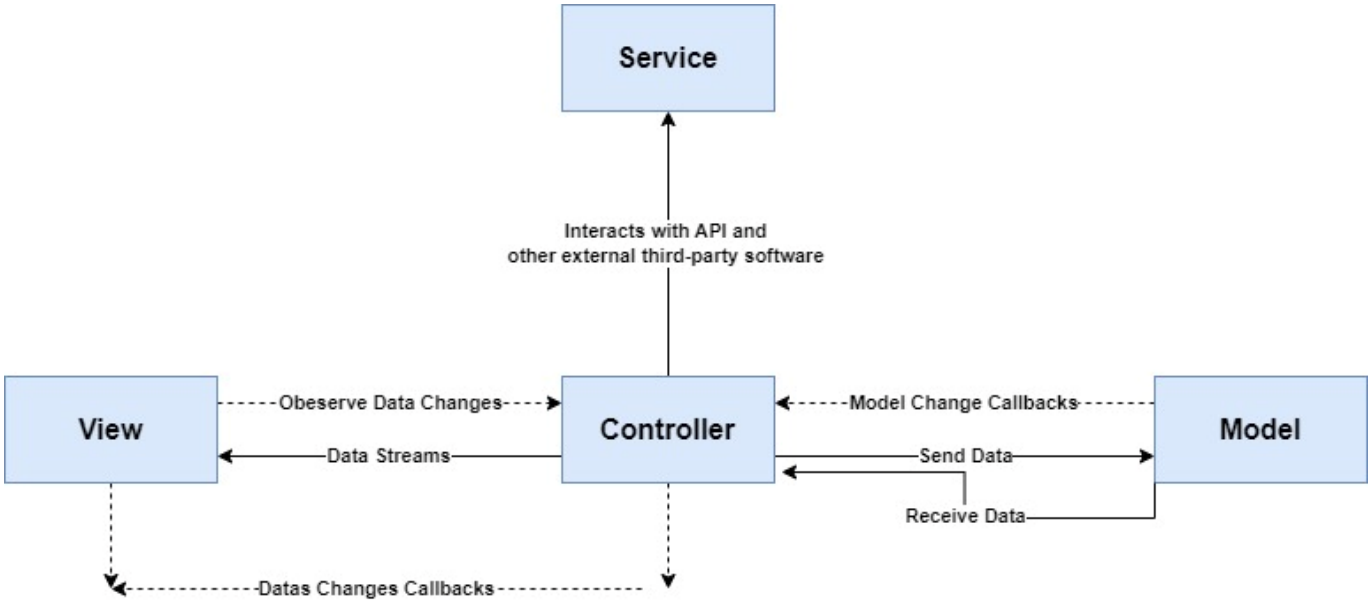
Provider

This is not a design pattern but a modelling concept of state management solution that is commonly used in Flutter applications. It is based on the concept of Inherited Widgets and allows for a clean way to manage state in Stateful and Stateless widgets throughout the application.

It will provide PODO (Plain Old Dart Object), *not to be confused in POJO (Plain Old Java Object) and JavaBeans classes in Java*, as a convenient way to manage the state of Flutter objects and propagate changes throughout the UI.

BLoC (Business Logic Component)

This is not a design pattern, but a concept related to MVC. It is particulary useful for managing complex state and business logic in Flutter apps. It separates UI from business logic and state management, providing a clear and scalable architecture.



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

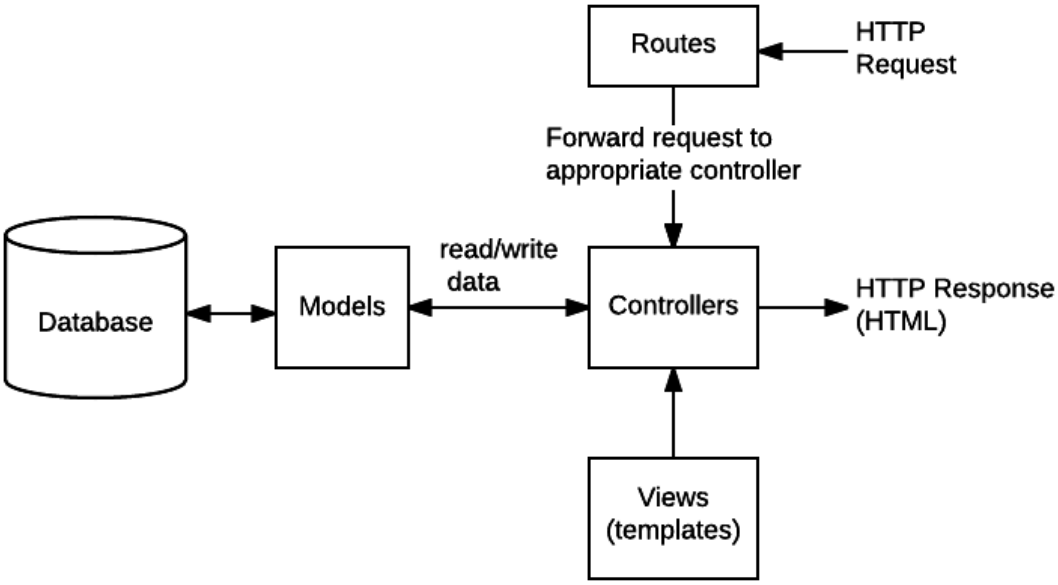
This directory will be responsible for storing the files that contain HTML elements to show the JSON response to the user in the server. The files of this directory will not be a lot.

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.



Adapter Pattern

The adapter is a great addition to use for the code scalability and maintenance for the future projects. It helps by providing a unified interface to interact with different kinds of external APIs or services, whether they are RESTful or SOAP API, abstracting away the implementation details.

Service Layer Pattern

This pattern can be used to encapsulate the business logic of the application. It provides a way to organize the code that performs operations on the data, separating it from the Controller and Model. This is useful for handling complex business logic and keeping the Controllers and Models clean.

Dependency Injection

This design pattern can be used to manage dependencies regarding the Node environment. It will make the code more modular, easier to test, and improves the scalability of the application. It is particularly useful in a server-side application where there are multiple services and repositories that need to be injected in the Controllers.

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.

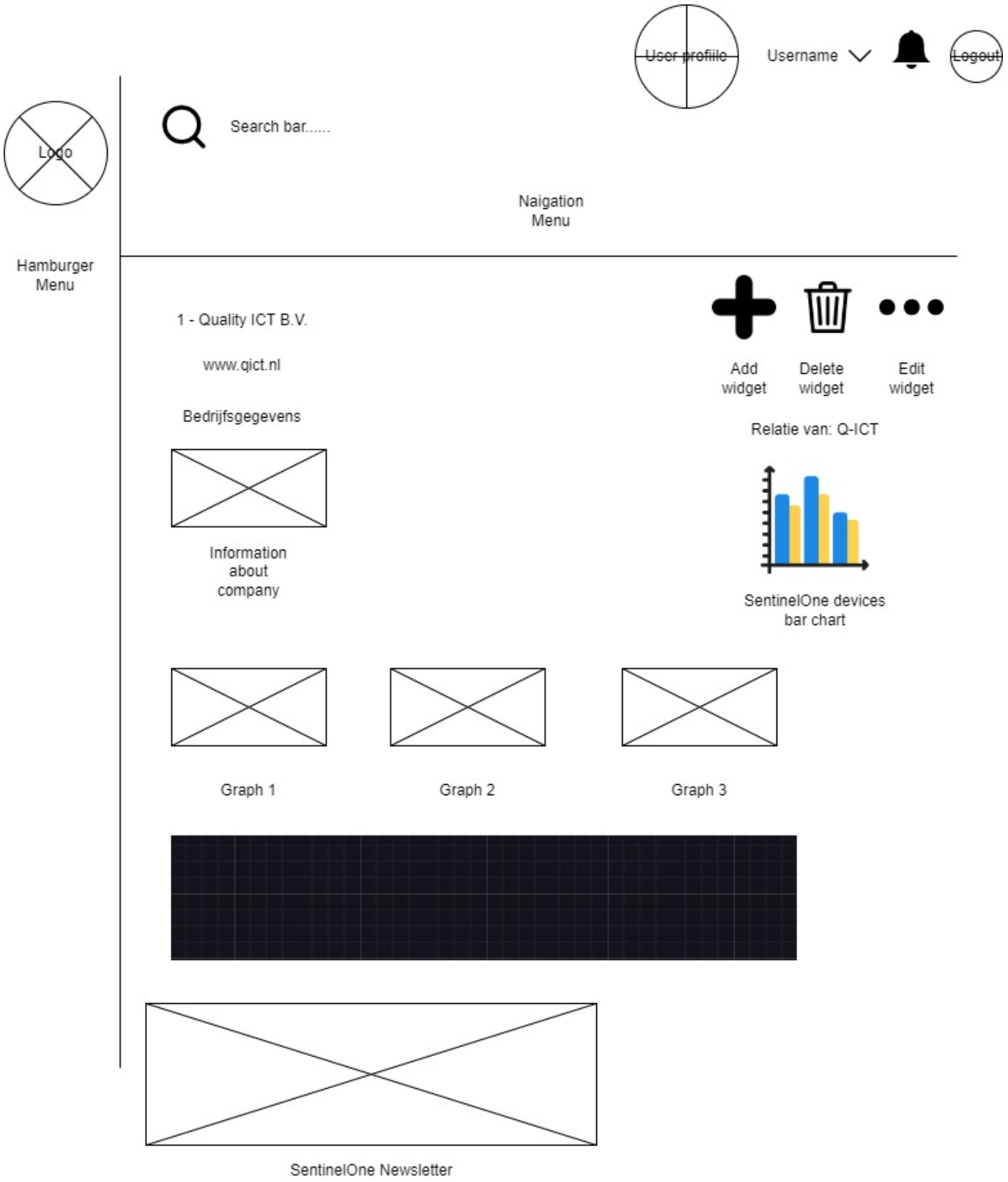
ODM (Object Document Mapping)

It is a concept derived from ORM (Object Relational Mapping).

User Interface and System Design

Wireframes

Sentinel One Dashboard



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)

Endpoint Page

Used SentinelOne API(s):

- Get Agents (filter: Site ID)

Specific Endpoint Page

Used SentinelOne API(s):

- Get Agents (filter: Agent ID, Site ID)

Threat Page

Used SentinelOne API(s):

-

Specific Threat Filter Page

Used SentinelOne API(s):

- Get Threat, filter by:
    - Account IDs (String array)
    - Agent IDs (String array)

Applications Page

Used SentinelOne API(s):

Activities Page

Used SentinelOne API(s):

Settings Page

Used SentinelOne API(s):

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

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…) 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

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				

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

Chapter 7 Organizational Consequences

This chapter describes what problems that may occur for the users when accessing this new feature.

How will the end-product will be tested?

The end-product which will be in the form of a functionality on top of an existing web application, with working servers will be tested by the author, the Company Supervisor, and the Company Stakeholders.

GitHub and Azure DevOps will be the main platform where the author and his supervisor (the development team) can use for developing, collaborating, and making the version control Each function in FCS and Flutter Widgets and pages will be tested with test input, along with their outputs will be written down in Test Log documentation to make sure the software meets the desired requirements. A Design Document (included in this document) containing desired product mock-ups and wireframes will be produced for product's UI to make sure it is easy to use for a good UX.

How will the acceptance by the users of the new system be arranged?

In the middle of the Realization phase, a pilot testing will be conducted with a small group of users, preferably the stakeholders and other people who do not know much about IT, to evaluate the system's usability, functionality, and performance in a real-world environment. Feedback will be gathered from pilot users and necessary adjustments will be made before full deployment.

Which conversion problems can be expected?

The class base model from the SentinelOne API data format may differ from the format expected by existing infrastructure of the QaaS app. If this is the case, the author may need to create a new model just for SentinelOne to make it work. Other than that, the author will make sure that the new implementation will follow the skeletal structure of the QaaS app, along with using the same Widgets.

Which training courses are required for the end users?

There are no specific courses to make use of the system, as the page itself is designed to be user friendly to a customer that does not know anything about IT and cybersecurity. However, the end-users are expected to know the details of their own computers (specifications, what are the installed software, drivers, and hardware). The end-users are encouraged to read the SentinelOne official documentations about their own products, as well as

Deployment Plan

Risk Assessment

Mitigation

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?

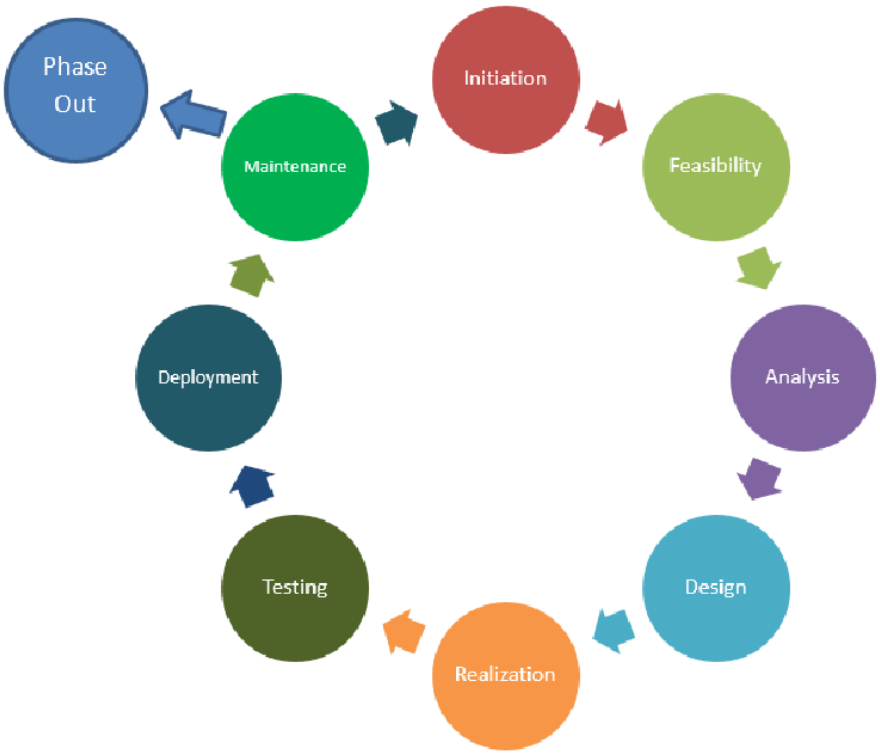
Are extra security measures required for data protection?

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*).



Chapter 9 Maintenance Plan

Chapter 10 Conclusion