IoT Intrusion Detection through Machine Learning

Research Group on Intelligent Engineering and Computing
for Advanced Innovation and Development

2020 / 2021

João Vitorino

1180851

IoT Intrusion Detection through Machine Learning

Research Group on Intelligent Engineering and Computing
for Advanced Innovation and Development

2020 / 2021

João Vitorino

1180851

Bachelor's Degree in Informatics Engineering

June, 2021

Supervisors: **Orlando Sousa**

Co-supervisors: **Isabel Praça**, **Rui Andrade**

« To both my grandfathers »

Acknowledgments

I would like to thank GECAD for this opportunity, as well as all those who supported me throughout this journey. In particular:

To my supervisor, Orlando Sousa, for his constant guidance and availability. To my co-supervisors, Isabel Praça and Rui Andrade, for all their advice and suggestions.

To my bachelor teammates, for their companionship and the moments we shared.

To my family, for the unconditional support. To my parents, Margarida and Mário, to whom I owe who I am today. To my brother André, who is always there for me.

To Joana, for all the patience and encouragement.

Abstract

The digital transformation faces great security challenges. In particular, the Internet of Things (IoT), a concept that expresses the interconnection of physical objects with the Internet, is exposed to several threats. The growing number of cyber attacks targeting IoT systems restates the need for a reliable detection of malicious network activity, to mitigate its impact. The application of Machine Learning (ML) to IoT intrusion detection is a promising approach to tackle the increasingly more complex threats.

This project presents a continuously improving Network-based Intrusion Detection System (NIDS) based on user feedback. The system consists of three modular applications and employs an adapted Deep Reinforcement Learning (DRL) methodology to incrementally improve the detection with the alerts validated by end users.

The binary and multi-class classification performances of the developed DRL model, a Support Vector Machine (SVM), a Light Gradient Boosting Machine (LightGBM), an Extreme Gradient Boosting (XGBoost), an Isolation Forest (iForest) and a Local Outlier Factor (LOF) were evaluated on several subsets of the IoT-23 dataset.

The obtained results indicate that the DRL model requires a large quantity of balanced data to be effective, whereas iForest and LOF are more suitable for a smaller quantity of unbalanced data. Overall, SVM, LightGBM and XGBoost obtained similar results. LightGBM achieved the most reliable performance.

Keywords (Subject): Internet of things, Intrusion detection system, Machine learning, Deep reinforcement learning

Keywords (Technologies): Python, Node.js, React, MongoDB Atlas

Resumo

A transformação digital enfrenta grandes desafios de segurança. Em particular, a Internet das Coisas (IoT), um conceito que representa a interligação de objetos físicos com a Internet, está exposta a diversas ameaças. O crescente número de ciberataques direcionados a sistemas IoT reforça a necessidade de uma deteção fiável de tráfego de rede malicioso, para que o seu impacto possa ser mitigado. A aplicação de técnicas de aprendizagem automática (ML) na deteção de ataques em sistemas IoT é uma abordagem promissora para enfrentar as ameaças cada vez mais complexas.

Este projeto apresenta um sistema de deteção de ciberataques que se melhora continuamente através da interação com utilizadores. O sistema consiste em três aplicações modulares e utiliza uma metodologia adaptada de aprendizagem por reforço profundo (DRL) para melhorar a deteção com os alertas validados por utilizadores.

Os desempenhos em classificação binária e multiclasse do modelo DRL, de Support Vector Machine (SVM), de Light Gradient Boosting Machine (LightGBM), de Extreme Gradient Boosting (XGBoost), de Isolation Forest (iForest) e de Local Outlier Factor (LOF) foram avaliados em vários subconjuntos do dataset IoT-23.

Os resultados obtidos indicam que o modelo DRL requer uma grande e equilibrada quantidade de dados, enquanto que iForest e LOF são mais adequados para pequenas e desequilibradas quantidades de dados. No geral, SVM, LightGBM e XGBoost obtiveram resultados similares. LightGBM alcançou o desempenho mais fiável.

Palavras-chave (Tema): Internet das coisas, Sistema de deteção de ataques, Aprendizagem automática, Aprendizagem por reforço profundo

Palavras-chave (Tecnologias): Python, Node.js, React, MongoDB Atlas

Contents

List of Figures

List of Tables

List of Abbreviations

6LoWPAN	Internet Protocol version 6 over Low-Power Wireless Personal Area Networks
ANN	Artificial Neural Network
API	Application Programming Interface
ARP	Address Resolution Protocol
BPMN	Business Process Model and Notation
CI/CD	Continuous Integration/Continuous Delivery
CNN	Convolutional Neural Network
CoAP	Constrained Application Protocol
CORS	Cross-Origin Resource Sharing
CPS	Cyber-Physical Systems
CPU	Central Processing Unit
DBN	Deep Belief Network
DDD	Domain-Driven Design
DDoS	Distributed Denial-of-Service
DDQN	Double Deep Q-Network
DL	Deep Learning
DoS	Denial-of-Service
DQN	Deep Q-Network
DRL	Deep Reinforcement Learning
DTO	Data Transfer Object
FURPS+	Functional, Usability, Reliability, Performance, Supportability, Plus
GECAD	Research Group on Intelligent Engineering and Computing for Advanced Innovation and Development
GOSS	Gradient-based One-Side Sampling
GPU	Graphics Processing Unit
HIDS	Host-based Intrusion Detection System
HTTP	Hypertext Transfer Protocol

HTTPS	Hypertext Transfer Protocol Secure
IDS	Intrusion Detection System
iForest	Isolation Forest
IIoT	Industrial Internet of Things
IoT	Internet of Things
IP	Internet Protocol
IT	Information Technology
JSON	JavaScript Object Notation
JWT	JavaScript Object Notation Web Token
LightGBM	Light Gradient Boosting Machine
LOF	Local Outlier Factor
ML	Machine Learning
MQTT	Message Queuing Telemetry Transport
NIDS	Network-based Intrusion Detection System
OT	Operational Technology
POAHPS	PartOfAHorizontalPortScan
RAM	Random-Access Memory
ReLU	Rectified Linear Unit
REP	Experience Replay process in Learning Agent
REST	Representational State Transfer
REV	Revision process in Learning Agent
RPL	Routing Protocol for Low-Power and Lossy Networks
SaaS	Software as a Service
SeCoIIA	Secure Collaborative Intelligent Industrial Assets
SPA	Single-Page Application
SVM	Support Vector Machine
UML	Unified Modeling Language
XGBoost	Extreme Gradient Boosting

1 Introduction

This chapter consists of an overview of the project's context, the problem description and the structure of the report.

1.1 Project Context

This project was developed at Research Group on Intelligent Engineering and Computing for Advanced Innovation and Development (GECAD) [1]. GECAD is a research unit with the mission of carrying out scientific research to incorporate Intelligence in Engineering and Computing Complex Systems. Research is performed in several areas, such as Cyber-Physical Systems (CPS) and Internet of Things (IoT), with a focus on Artificial Intelligence and therefore Machine Learning (ML).

GECAD participates in multiple research and development projects. In particular, it integrates the Secure Collaborative Intelligent Industrial Assets (SeCoIIA) project [2]. This project counts with a consortium of twelve partners, including Airbus CyberSecurity, and is funded by the European Union's Horizon 2020 programme.

SeCoIIA aims at securing the digital transition of the manufacturing industry. It addresses Operational Technology (OT) security challenges on the aerospace, automotive and naval construction industries. Several use cases are under development to create Industrial Internet of Things (IIoT) solutions with intelligent security monitoring.

The work developed in this project was aligned with the research and development work of GECAD in collaboration with SeCoIIA and Airbus CyberSecurity.

1.2 Problem Description

The digital transformation is associated with the IoT concept, which was first proposed by Kevin Ashton in 1999 [3] and has rapidly evolved to describe decentralized and heterogeneous systems of interconnected devices. This field can be regarded as the networking infrastructure for CPS because it converges wireless sensor networks, real-time computing, embedded systems and actuation technologies [4].

IIoT is a subfield of IoT that focuses on the interconnection of industrial assets and the automation of manufacturing processes. It is closely related to the term Industry 4.0, which aims at the optimization of industrial operations. Furthermore, IIoT is bridging the gap between OT and Information Technology (IT) due to the integration of physical processes and control systems with the business processes and information systems [5].

However, the convergence of previously isolated systems and technologies poses great security challenges due to their underlying vulnerabilities. The growing number and dynamic nature of cyber attacks targeting these systems reinforces the need for reliable and continuously improving intrusion detection, to mitigate their impacts [6].

An Intrusion Detection System (IDS) monitors an environment with the purpose of identifying suspicious activity, so that possible threats can be mitigated. A Network-Based IDS (NIDS) monitors network activity, whereas a Host-Based IDS (HIDS) analyses the system and application files of a host machine. Due to the heterogeneous nature of IoT and the resource constraints of the devices, NIDS are employed to monitor the IoT network environment [6].

The application of Machine Learning (ML) to a NIDS is a promising approach to tackle the increasingly more complex threats targeting IoT systems.

1.2.1 Objectives

To address the described problem, this project aimed to apply ML to IoT intrusion detection. For that purpose, the following groups of objectives were established:

- **Development of a NIDS**
  - Development of a NIDS to detect possibly malicious network activity and alert end users;
  - Integration of an adapted Deep Reinforcement Learning (DRL) methodology to continuously improve the intrusion detection through user feedback.
- **Implementation of ML models**
  - Implementation of multiple supervised models;
  - Implementation of multiple unsupervised models;
  - Conception of a DRL model adapted to the intrusion detection context;
  - Evaluation and comparison of the performances of all models.

1.2.2 Work Methodology

Regarding the development of the NIDS, the Agile [7] methodology was adopted. It consisted of an iterative process following the six distinct phases displayed in Figure 1. Each iteration incrementally improved the system by performing an initial analysis/planning, design, implementation, testing, deployment and a review of an application.

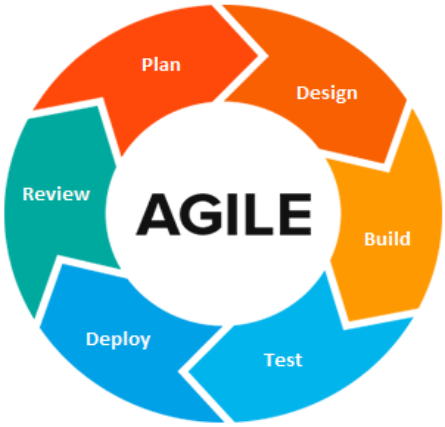

Figure 1. Agile software development methodology [8].

Regarding the implementation of each ML model, the workflow followed the six phases of the Cross Industry Standard Process for Data Mining [9], displayed in Figure 2. It consisted of an iterative process of business and data understanding, followed by data preparation and modelling, as well as an evaluation of the achieved model. This process was repeated until a final model was ready for deployment.

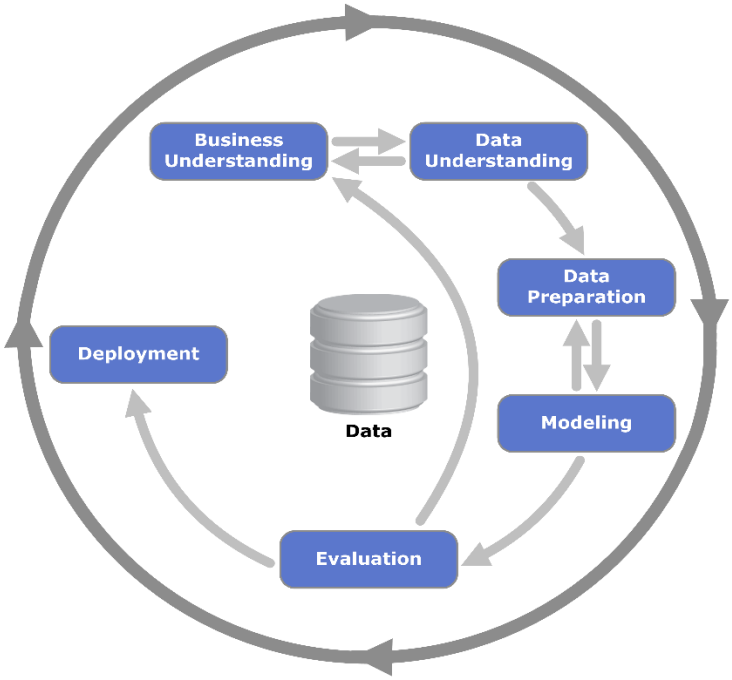

Figure 2. Cross Industry Standard Process for Data Mining [9].

1.2.3 Contributions

This project contributed to the research and development work of GECAD. Overall, the key contributions were the following:

- Development of a modular NIDS employing the DRL methodology to continuously improve the detection through user feedback and adopting a hybrid placement strategy to benefit from distributed monitoring devices;
- Conception and evaluation of a DRL model and training process adapted to the IoT intrusion detection context;
- Evaluation of the performance of three supervised models, namely SVM, LightGBM and XGBoost, assessing their applicability to IoT intrusion detection;
- Evaluation of the performance of two unsupervised models, namely iForest and LOF, assessing their applicability to IoT intrusion detection.

1.2.4 Work Plan

The work developed in this project was divided into several tasks. Table 1 provides an overview of the planned tasks, with their start and end dates.

Table 1. Overview of planned tasks.

Task	Start Date	End Date
Problem analysis	2021-03-12	2021-03-19
Scientific survey	2021-03-19	2021-04-09
Technology survey	2021-04-02	2021-04-16
Implementation of the supervised models	2021-04-16	2021-04-23
Implementation of the unsupervised models	2021-04-23	2021-04-30
Conception of the DRL model	2021-04-30	2021-05-07
Evaluation and comparison of the models	2021-05-07	2021-05-14
Development of the general system	2021-05-14	2021-05-21
Development of the Master Data application	2021-05-21	2021-06-04
Development of the SPA application	2021-06-04	2021-06-11
Development of the Learning Agent application	2021-06-11	2021-06-18
Writing of the report	2021-03-19	2021-07-02

1.3 Report Structure

Besides the Introduction, the report is organized into multiple chapters. Chapter 0 reviews the current state of the art, subdivided into scientific and technology surveys. In Chapter 3, a thorough description of the development of the NIDS solution is provided. Chapter 0 describes the implementation details of the NIDS, as well as the performed tests and an overall assessment of the system. In Chapter 0, the implementation of the ML and DRL models is detailed and the obtained results are presented and discussed. Chapter 6 addresses the accomplished objectives and the constrains of the solution, as well as future work that can be developed to improve it.

2 State-of-the-Art

This chapter consists of a scientific survey of the project's subject, followed by a survey of existing technologies.

2.1 Scientific Survey

To develop a reliable solution, it is essential to first understand the security challenges IoT systems face, analyse the possible attacks to these systems and explore previous applications of ML to intrusion detection.

2.1.1 Security Challenges

Throughout the years, the underlying vulnerabilities of each system and technology used in IoT have been surveyed [10]–[13]. To consolidate common approaches, in [14], Neshenko et al. presented three broad layers of vulnerabilities, briefly described below.

Device-based vulnerabilities. Weak physical security allows access to sensitive data and configurations. Additionally, insufficient power supplies and limited computational capacity inhibit the adoption of resource-intensive security mechanisms.

Network-based vulnerabilities. Communication protocol weaknesses, as well as inadequate port blocking policies, expose networks and devices to malicious activity.

Software-based vulnerabilities. Default or weak user credentials, outdated software and bad programming practices make devices more susceptible to being compromised.

Even though each layer poses different challenges, they are connected. Therefore, it must be taken into consideration that IoT is only as secure as the most vulnerable layer.

The Confidentiality, Integrity and Availability Triad is an acknowledged security model. The three principles establish that data cannot be read by third parties, cannot be modified during transmission and must always be available when needed [15]. In IT, the C-I-A format is typically followed, prioritizing data Confidentiality in information systems. However, in OT, the priorities are reversed and the A-I-C format is adopted due to the importance of data Availability in physical processes [16].

Consequently, IoT and especially the IIoT subfield face the challenge of securely converging systems and technologies with distinct purposes. In that regard, several communication protocols have been established, such as Hypertext Transfer Protocol (HTTP), Message Queuing Telemetry Transport (MQTT), Constrained Application Protocol (CoAP), Routing Protocol for Low-Power and Lossy Networks (RPL) and Internet Protocol version 6 over Low-Power Wireless Personal Area Networks (6LoWPAN) [17]. Nonetheless, all these protocols can be exploited.

Furthermore, in [11], Meneghello et al. clustered IoT-specific requirements into three operational levels, namely Information, Access and Functional. A key aspect of the Functional level is that the authors opted for Resilience and Self-Organization, instead of the Availability principle. Therefore, in case of an attack or a malfunction, an IoT system must not only stay operational, but also be capable of reorganizing itself to ensure permanent security. Table 2 provides an overview of the requirements.

Table 2. Summary of IoT security requirements. Based on [11].

Operational Level	**Requirements**
Information	Integrity Confidentiality Anonymity Privacy
Access	Authentication Authorization Access Control
Functional	Resilience Self-Organization

2.1.2 Attack Categorization

Overall, malicious activity targeting wireless networks can be divided into four broad groups, namely Denial-of-Service (DoS), Probing, User to Root and Remote to Local [18].

To provide a clearer view of the origin, target and impact of malicious activity specifically targeting IoT, the wide range of known attacks have been thoroughly surveyed [19]–[23]. Nonetheless, zero-day attacks are a permanent and unforeseeable threat.

Attacks targeting IoT systems can exhibit four distinct behaviours towards the functionalities of IoT devices [11], described below.

Ignoring the functionality. The functionality is disregarded and only the device's communication capabilities are exploited, which is common to all computer networks.

Reducing the functionality. The functionality is limited or blocked to create malfunctions in a wider system. For instance, a flight control sensor failure can be catastrophic in the aerospace industry.

Misusing the functionality. The functionality is used with a malicious purpose. For instance, a temperature sensor can be misused to damage a climate-controlled room.

Extending the functionality. The device is used to achieve unintended functionalities. For instance, a motion sensor of a light system can be extended to track the position of a victim, even when the light system is off.

In [21], Panchal et al. proposed a taxonomy that separates physical from cyber attack vectors. The first involve physical tampering and match the layer of Device-based vulnerabilities. The latter are done over the network and combine the Network-based and Software-based vulnerabilities, which is the focus of a NIDS. Furthermore, in [22], Tahsien et al. divided cyber attacks into Passive and Active, identifying the types of attacks and affected security requirements. A Passive attack does not impact the operation of a network, whereas an Active attack does. Table 3 summarizes the categorization.

Table 3. Categories of cyber attacks targeting IoT systems. Based on [21], [22].

Category	Types of Attacks	Affected Requirements
Passive	Traffic analysis Eavesdropping	Privacy
Active	Denial-of-Service Man-in-the-Middle Brute-Force Probing Spoofing Injection Malware	Authentication Authorization Confidentiality Integrity Resilience

The Malware type is a major threat to IoT due to the high susceptibility of the devices to malicious software. These attacks led to the rise of botnets, which are established when a malware, such as the self-propagating Mirai, compromises vulnerable hosts. Botnets can use IoT devices to launch several other types of attacks, but are mainly associated with Distributed Denial-of-Service (DDoS) [23].

Additionally, in [4], Butun et al. subcategorized Active attacks following the layers of the Open Systems Interconnection protocol stack [24]. This provides a better grasp of the exploited communication protocols. Still, the Session and Presentation layers were merged with Application, similarly to the Internet Protocol Suite [25]. Table 4 provides an overview of the attacks common to each layer.

Table 4. Layers of Active cyber attacks targeting IoT systems. Based on [4].

Protocol Layer	**Common Attacks**
Application	Path-based DoS, False Data Injection, Cross-Site Request Forgery, CoAP exploit, Re-programming, Sensor overwhelming
Transport	SYN flooding, MQTT exploit, Session hijacking, Desynchronization
Network	HELLO flooding, RPL exploit, Node replication, Sybil attacks, Routing attacks, Hole attacks
Data Link	Link layer jamming, Link layer flooding, ARP spoofing, Denial of sleep, Collusion, Exhaustion, Unfairness, 6LoWPAN exploit, TSCH desynchronization
Physical	Physical layer jamming

2.1.3 Intrusion Detection

There are several aspects to be considered for the deployment of a NIDS, namely the detection method, the source of the analysed data and the placement and validation strategies [23], [26], [27]. Table 5 provides and overview of the different aspects. The four main categories of detection methods are briefly described below.

Signature-based. The behaviour of known attacks is defined and the network is monitored for equivalent patterns. Despite correctly identifying known threats, the patterns of zero-day attacks cannot be detected.

Anomaly-based. A baseline of the regular network behaviour is built and the network is monitored for anomalous activity. The baseline is built according to the network topology and activity that exceeds a certain threshold is considered an anomaly, which is effective against both known and unknown attacks. The drawback is that unusual but benign activity may be misclassified as a threat.

Specification-based. The regular network behaviour is manually specified and, similarly to Anomaly-based, the network is monitored for anomalous activity. Even though the manual configuration can decrease the misclassification of benign activity, it is prone to errors and difficult to adapt to topology changes.

Hybrid. Two or more methods are combined to overcome their disadvantages.

Regarding IoT, anomaly-based detection is a well-established methodology due to its adaptability and reliability against the dynamic nature of cyber attacks [6]. The techniques utilized to detect anomalies can be clustered into Statistical-based, Knowledge-based and ML-based [18], [20]. The latter are the focus of this project and will be detailed in the next section.

Table 5. Characteristics and alternatives of NIDS deployment. Based on [23], [26], [27].

Characteristic	**Alternatives**
Detection Method	Signature-based Anomaly-based Specification-based Hybrid
Data Source	Packet-based Flow-based Session-based
Placement Strategy	Centralized Distributed Hybrid
Validation Strategy	Simulation Empirical Theoretical Hypothetical

2.1.4 Machine Learning

From an ML standpoint, intrusion detection is regarded as a classification problem [18]. Therefore, only ML models capable of sorting data into classes can be employed. ML models performing classification are referred to as classifiers and can be divided according to the three paradigms of ML [28], [29], described below. Table 6 summarizes the main characteristics of each paradigm and the possible applicability to a NIDS.

Supervised Learning. An outcome is predicted based on labelled input data. The outcome can be a binary or multi-class label. The first corresponds to two classes, either benign or malicious activity. The second further classifies malicious activity into specific types of attacks, labelling network traffic into either benign or one of multiple malicious classes.

Unsupervised Learning. Patterns are discovered in unlabelled input data. One-class classification can be performed by recognizing the patterns of a single benign class and considering anomalies as malicious activity. Since the data is divided into two groups, it is comparable to binary classification.

Reinforcement Learning. An agent iteratively performs actions based on the observations obtained from an environment. Each action is assigned a reward, which the agent utilizes to improve the decision process that leads to the choice of future actions, attempting to maximize the rewards. This distinctive reward system can be utilized to continuously improve the classification [30]. Figure 3 displays a simplified view of the interactions between an agent and an environment.

Figure 3. Reinforcement Learning interactions. Based on [31].

Additionally, supervised and unsupervised characteristics can be combined to create a Semi-supervised approach with a small quantity of labelled data and a large quantity of unlabelled data [32].

Table 6. Characteristics and applicability of ML models. Based on [28], [29].

Paradigm	Main Characteristics	NIDS Applicability
Supervised Learning	Class prediction Labelled data	Binary or Multi-class Classification
Unsupervised Learning	Pattern recognition Unlabelled data	One-class Classification
Reinforcement Learning	Decision process Reward system	Iterative Improvement

Besides the three paradigms, models are commonly grouped as traditional ML and Deep Learning (DL). The latter is based on the Artificial Neural Network (ANN) model combined with representation learning. When DL is applied to the agent in the reinforcement learning paradigm, the term Deep Reinforcement Learning (DRL) is utilized [33].

2.1.5 Datasets and Generalization

To effectively identify anomalies, a model must be trained with relevant data. However, public datasets of IoT network traffic are scarce and mostly unbalanced towards benign activity due to the low rate of the recorded attacks. This is regarded as a major obstacle to ML-based intrusion detection [20]. Table 7 provides an overview of suitable labelled datasets.

A model is trained with the objective of achieving a good generalization. This term describes a model's ability to make correct predictions on previously unseen data. However, to achieve a good generalization, neither overfitting nor underfitting can occur. The first occurs when a model learns overly specific details and noise of the training data, performing well on the training data but poorly on new data. The latter occurs when the model does not learn sufficient details, performing poorly on both training and new data [34].

To improve the generalization of a model, its configuration must be optimized to a specific problem and dataset. The parameters utilized to control the learning process of a model, referred to as hyperparameters, can have numerous possible combinations. Two common approaches to search through different configurations are a grid search of values within a certain set and a random search of some arbitrary values within that set. These approaches can also be combined to perform a grid search of several arbitrary values [35].

A well-established technique to assess the generalization of a model is K-fold Cross-Validation, which divides the dataset into k subsets and performs k distinct training iterations. Each iteration uses k-1 subsets to train the model and 1 subset to validate it, evaluating its performance on that unseen data. This technique can be combined with a parameter search to discover the optimal configuration [36].

Another technique is to holdout a small portion of the dataset specifically for testing the trained model. Therefore, the dataset is split into training and test sets. This technique is commonly utilized together with K-fold Cross-Validation to perform a final evaluation of the optimal configuration [37]. Figure 4 displays this process utilizing 5-fold cross-validation as an example, where blue rectangles represent validation data.

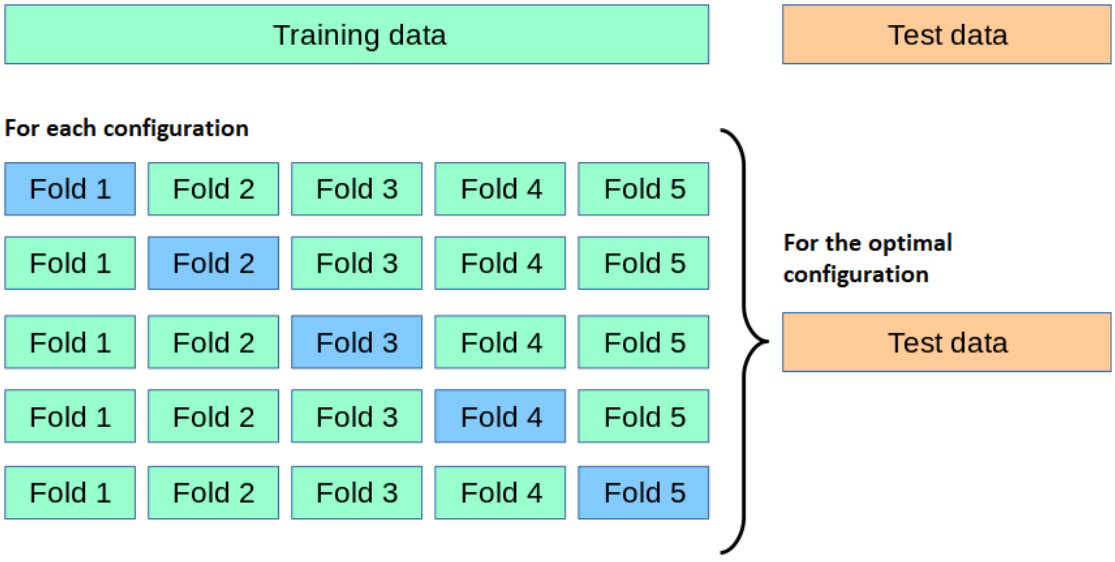

Figure 4. 5-fold Cross Validation and Holdout techniques combined. Based on [38].

Table 7. Overview of suitable labelled datasets.

Dataset	Year	IoT-specific	Types of Attacks	Description
NSL-KDD [39]	2009	✗	Probing Denial-of-Service User to Root Remote to Local	Broad groups of attacks targeting wireless networks
AWID2 [40]	2016	✗	Probing Denial-of-Service Injection Eavesdropping	General types of attacks targeting wireless networks
N-BaIoT [41]	2018	✓	Malware	Attacks caused by the Mirai and BASHLITE malwares
Bot-IoT [42]	2019	✓	Probing Denial-of-Service Eavesdropping	Attacks focused on information theft
MQTT-IoT-IDS2020 [43]	2020	✓	Probing Brute-Force	Attacks using the MQTT communication protocol
IoT-23 [44]	2020	✓	Malware	Attacks caused by several malwares, such as Mirai, Torii, Gafgyt, Kenjiru and Okiru
AWID3 [45]	2021	✗	Probing Denial-of-Service Injection Eavesdropping	New version of AWID2 with more recent attacks

2.1.6 Performance Evaluation

The performance of a model can be evaluated using the values reported by the confusion matrix. It reports the number of True Positives (TP), True Negatives (TN), False Positives (FP) and False Negatives (FN), regarding the predicted classes [46].

Utilizing binary classification as an example, the commonly utilized metrics and their interpretation are described below [18], [27], [46].

Accuracy measures the proportion of correctly classified network traffic. However, in unbalanced datasets, a high accuracy can be achieved without correctly classifying a minority class. For instance, in IoT datasets unbalanced towards benign activity, a high accuracy can be achieved by only predicting this class. It can be mathematically expressed by Equation (1).

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{1}$$

Precision measures the proportion of predicted attacks that were actual attacks, which indicates the relevance of the model's predictions. It can be calculated by Equation (2).

$$Precision = \frac{TP}{TP + FP} \tag{2}$$

Recall, which corresponds to the True Positive Rate (TPR) and is also referred to as Sensitivity and Detection Rate, measures the proportion of actual attacks that were correctly predicted. This metric reflects the model's ability to identify malicious activity. It can be defined by Equation (3).

$$Recall = \frac{TP}{TP + FN} \tag{3}$$

False Positive Rate (FPR), also referred to as False Alarm Rate, measures the proportion of benign activity that was incorrectly predicted to be an attack. It can be mathematically expressed by Equation (4).

$$FPR = \frac{FP}{FP + TN} \tag{4}$$

F1-Score, also referred to as F-Measure, calculates the harmonic mean of Precision and Recall, which consolidates the FP and FN. It can be calculated according to Equation (5).

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{5}$$

The Receiver Operating Characteristic (ROC) curve is a graphical plot with FPR on the x-axis and TPR on the y-axis. The Area Under the Curve (AUC) is a scale-invariant measure of a model's ability to distinguish between two classes, obtained from the two-dimensional area beneath the ROC curve.

The mentioned metrics, except for Accuracy, can be macro-averaged to treat all classes equally. Since the minority classes are given the same relevance as the overrepresented, macro-averaging is well suited for unbalanced datasets.

2.1.7 Related Approaches

There is a growing number of NIDS being developed for IoT systems, with a focus on ML-based anomaly detection methods. Table 8 provides an overview of recent research that addresses these developments, either by surveying other approaches or analysing specific models. Additionally, Table 9 highlights some distinctive approaches.

Several approaches opted for a distributed placement strategy because the activity captured throughout the network exhibited device-specific characteristics that aided the building of the network baseline. This strategy is associated with edge computing and Edge AI. The resource constraints of IoT devices imposed the use of only lightweight models, such as SVM, LightGBM and the C4.5 algorithm [20], [47].

Even though the distributed strategy achieved good performances, many devices do not support software updates, which inhibits large-scale deployment. To tackle this limitation, some approaches placed dedicated devices across the network to monitor neighbouring devices, acting as watchdogs. Despite supporting lightweight software, the watchdogs presented similar resource constraints [17].

On the other hand, other approaches opted for capturing and analysing network traffic in a centralized component with fewer resource constraints. In some cases, the analysis was performed in the cloud to take advantage of cloud computing, although it increased network bandwidth consumption. This centralized placement strategy enabled the use of more computationally intensive methods, which performed well against most types of attacks. However, it created a single point of failure especially susceptible to DoS [27], [48].

The most pertinent models employed the supervised learning paradigm. They ranged from traditional ML classifiers, such as Random Forest, to DL models, such as Convolutional Neural Network (CNN) and Deep Belief Network (DBN) [20], [48]. Nonetheless, some unsupervised models, such as iForest, also presented good performances [49].

To overcome the drawbacks of each strategy, hybrid strategies were developed based on the fog computing concept. These approaches captured the network traffic in distributed devices and sent it to a centralized component or the cloud for a more complete analysis. In some cases, the distributed devices performed a lightweight analysis and applied dimensionality reduction techniques before transmitting the data, to decrease network bandwidth consumption [6].

Due to the lack of IoT-specific public datasets at the time, most approaches were trained and validated on more generic datasets, such as NSL-KDD. Nonetheless, many authors opted for creating a dataset from network traffic captured on their own devices or using a simulated environment. Regarding the data source, the majority of approaches consider the traffic flow patterns, adopting a flow-based analysis [23], [50].

The overall drawback is that if the network topology is changed, which includes the addition of a new device, the models must be retrained to take into consideration the updated topology and the new traffic patterns.

To tackle this challenge, several approaches adapted the reinforcement learning paradigm to the intrusion detection context with the purpose of continuously improving a model. These approaches were mostly based on the traditional Q-Learning model, which is an off-policy learner because it is improved regardless of the agent's actions. It was combined with ANNs to create models such as Deep Q-Network (DQN) and Double Deep Q-Network (DDQN), employing an adapted training process based on the DRL methodology [30].

Table 8. Overview of related research papers.

Paper	Year	IoT-specific	Subject
[17]	2018	✗	Survey of NIDS approaches, emphasizing the advantages of watchdogs.
[18]	2019	✗	Analysis of detection methodologies, public datasets and IDS evasion techniques.
[27]	2019	✗	Detailed comparison of ML and DL models for intrusion detection.
[30]	2019	✗	Detailed analysis of several DRL models and training processes for intrusion detection.
[23]	2019	✓	Survey of detection methodologies for IoT systems, open test beds and existing simulators.
[47]	2019	✓	Review of approaches focused on the detection of DoS attacks in IoT systems.
[50]	2020	✓	Review of approaches focused on the detection of botnet attacks in IoT systems.
[48]	2020	✓	Analysis of several ML and DL models for intrusion detection in IoT systems.
[6]	2020	✓	Detailed comparison of IoT NIDS approaches, emphasizing the advantages of fog computing.
[20]	2020	✓	Detailed comparison of ML and DL models for IoT intrusion detection.
[49]	2020	✗	Detailed analysis of several unsupervised ML models for intrusion detection
[51]	2021	✗	Detailed comparison of single-flow and multi-flow perspectives for intrusion detection

Table 9. Highlights of distinctive approaches.

Approach	Year	Method			Data Source	Training / Validation	Placement	Highlight
		Paradigm	Group	Model				
[52]	2017	Supervised Learning	ML	Random Forest	Session-based	Own dataset	Centralized	Performs whitelisting, building a baseline of the allowed device types.
[53]	2019	Supervised Learning	ML	C4.5	Packet-based	Own dataset	Distributed	Uses a lightweight model based on a decision tree, with pruning techniques.
[54]	2019	Supervised Learning	ML + DL	LightGBM + CNN	Flow-based	Own dataset	Hybrid	Uses a lightweight model with embedded feature selection ahead of a neural network.
[55]	2019	Unsupervised Learning	DL	DBN	Flow-based	NSL-KDD	Centralized	Optimizes the structure of the neural network using an evolutionary algorithm.
[56]	2019	Reinforcement Learning	DL	DQN	Flow-based	NSL-KDD + AWID2	Centralized	Trains the main agent with samples iteratively selected by an adversarial agent
[57]	2020	Unsupervised Learning	ML	Isolation Forest	Flow-based	Own dataset	Centralized	Builds a baseline of benign activity only and deploys the model on a Raspberry Pi.
[58]	2020	Reinforcement Learning	ML	Q-Learning	Flow-based	Own dataset	Distributed	Trains the agent to select the best anomaly threshold using TPR and FPR as rewards

2.1.8 Deep Reinforcement Learning Methodology

Regarding the DRL methodology employed to train a model, there are several aspects to be taken into consideration. In particular, the training process of a DDQN [59] improved the agent of a simple DQN [59] by applying the following concepts:

Exploration. Instead of always acting according to the predictions of the ANN, an agent can occasionally explore other possible actions to avoid converging to a suboptimal solution. An efficient strategy is to apply the Epsilon-Greedy method, choosing predicted or random actions according to an exploration ratio [60].

Experience Replay. Instead of immediately training the model after an interaction with the environment, an agent can store those experiences in a finite memory. Then, when the model is trained, a minibatch of past experiences is randomly sampled from the memory. This approach logically separates the interaction phase from the learning phase, mitigating the risk of catastrophic interference [61].

Target network. Instead of using the same ANN for computing both the predictions and their target values during experience replay, an agent can utilize two separate ANNs. An active network is actively trained while a target network is being utilized as a function approximator. The latter is a copy of the first that is only periodically updated. This approach performs a delayed synchronization of the target network to minimize the instabilities inherent to incrementally training an ANN with minibatches, improving its generalization [62].

2.1.9 Software Architecture

Another important aspect to consider is the architectural design of the NIDS. An acknowledged approach is Domain-Driven Design (DDD) [63]. It centres the development of a solution on the conceptual model of the problem domain, involving the use of Object-Oriented Design (OOD). The SOLID principles [64] and the General Responsibility Assignment Software Principles (GRASP) [65] establish the best-practices for OOD architectures.

Regarding the communication with an Application Programming Interface (API), the Representational State Transfer (REST) [66] architectural style defines the common practices. To access the resources of a REST API, the standardized communication protocol is HTTP, which can also be extended to Hypertext Transfer Protocol Secure (HTTPS) to encrypt the communication.

2.2 Technology Survey

To implement a reliable solution, it is important to select trustworthy technologies. This survey will be focused on technologies suitable for the adopted approach.

2.2.1 Model Implementation

Both Python [67] and R [68] are open-source programming languages that provide standardized approaches to the entire process of implementing an ML model [69]. GECAD utilizes the Python language and several of its libraries to simplify the technical aspects. Nonetheless, their key similarities and differences are compared in Table 10.

Table 10. Comparison of Python and R languages. Based on [69], [70].

Characteristic	Python	R
Data collection	Support for all filetypes and web communication	Support for limited filetypes
Data exploration	Standard data analysis	Enhanced data analysis and statistical tests
Data modelling	Enhanced ML, scientific computing and numerical modelling	Standard ML and scientific computing
Data visualization	Standard graphs, plots and charts	Enhanced graphs, plots and all types of charts

2.2.2 Data Storage

There are two primary options for data storage, namely distributed and centralized. Their main advantages and drawbacks are described below.

A distributed ledger is a database that is shared and synchronized across a peer-to-peer network, eliminating the need for a third party [71], as displayed in Figure 5. Blockchains are a type of distributed ledgers that implement a cryptographically secure append-only sequence of blocks, which could be beneficial for intrusion detection [72]. The main drawback is the computational load and memory capacity this technology would require from IoT devices. Additionally, compromising a single device could potentially allow access to the stored data.

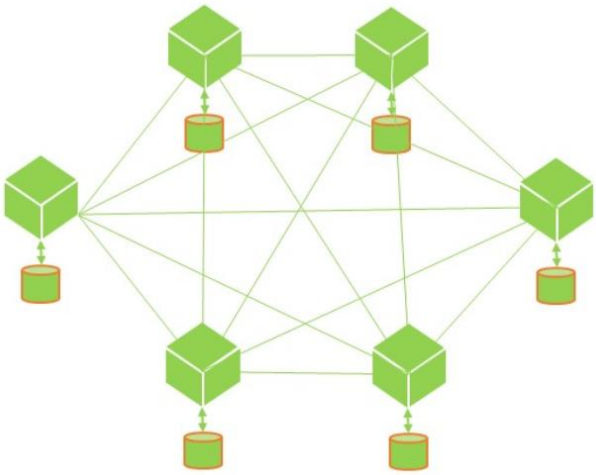

Figure 5. Peer-to-peer network of a distributed ledger [73].

On the other hand, centralized storage provides more control over access to the stored data. MongoDB Atlas [74] is an acknowledged cloud storage platform that securely stores serialized documents in NoSQL database clusters. It implements fault-tolerance by replicating the data written to the primary database into two secondary databases, as displayed in Figure 6. It provides databases with high availability and scalability for large quantities of data in a Software as a Service (SaaS) model [75]. Despite making the NIDS dependent on network connectivity to access the stored data, it would not exert any constrains on the IoT devices.

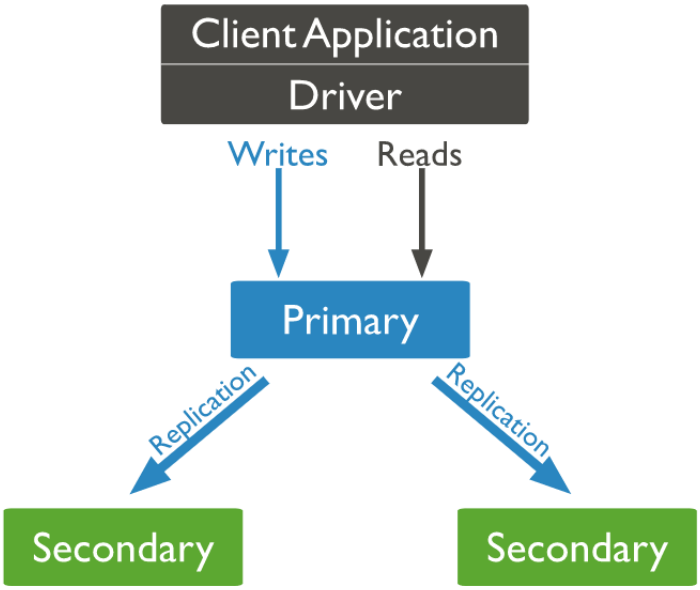

Figure 6. MongoDB database replication [76].

Similarly, Azure Cosmos DB [77] also provides databases in a SaaS model. Even though it has similar benefits to MongoDB Atlas, the key difference is that writing to a database is a slower process, as compared in [78]. Therefore, since the adopted approach will continuously need to store data, this technology is not as beneficial.

2.2.3 Web Application

Regarding the implementation of a web application, Node.js [79] and .NET [80] are two well-established open-source technologies. Table 11 compares their key differences.

Table 11. Comparison of Node.js and .NET technologies. Based on [81], [82].

Characteristic	Node.js	.NET
Primary Programming Language	JavaScript	C#
Usage	Server-side scripting	Web Forms
Processing	Asynchronous event-driven	Synchronous
I/O	Non-blocking	Blocking

Due to their characteristics, .NET provides a robust approach for large-scale applications, whereas Node.js provides a lightweight run-time environment that is more suitable for smaller-sized modular applications [82].

To implement back-end web applications on Node.js, the Express.js [83] framework can be utilized. It has become the standardly adopted framework because it enables the creation of minimalist and highly scalable applications, as well as REST APIs that can be enhanced with several middlewares [84], [85].

2.2.4 User Interface

Regarding the interaction with end users through an UI, the React [86] library and the Angular [87] framework are two widely adopted technologies. Table 12 compares their key similarities and differences.

Table 12. Comparison of React and Angular technologies. Based on [88], [89].

Characteristic	React	Angular
Type of Web Page	Support for single-page and multiple-page	Support for single-page and multiple-page
Usage	UI only	UI and software development
Processing	Virtual DOM	Real DOM (change detection)
Data Binding	Unidirectional (immutable data)	Bidirectional (mutable data)

Since React is a simplistic library focused on modular UI components, it reduces the workload of an end user's browser, especially when utilized to load only the necessary content in a single-page approach [89].

The Material-UI [90] library can be utilized together with React to improve the design styles of the front-end with user-friendly and intuitive components [91].

2.2.5 Deployment

Regarding the deployment of the NIDS, Heroku [92] is a Platform as a Service that offers a free tier for non-commercial projects, allowing the project to be focused on the applications instead of the infrastructure. The reliability of this free tier makes Heroku optimal for a simple proof-of-concept deployment, as compared with other platforms in [93].

3 Solution Development

This chapter addresses the development of the solution, covering both the analysis and the design of the employed approach.

3.1 Overview

The solution originated from the idea of alerting an end user to malicious activity and utilizing the validation of that alert to improve the detection. For that purpose, it was necessary to create a NIDS that employed the DRL methodology to incrementally improve the detection though user feedback.

Due to the dynamic nature of cyber attacks, as well as the constant evolution of IoT systems, a hybrid placement strategy was adopted. Therefore, the system is intended to store network flows captured by distributed devices, referred to as watchdogs. With this approach, it benefits from the flows captured and optionally analysed by the watchdogs, while being able to perform a more computationally intensive centralized analysis.

This analysis revises the label assigned to a flow, so it must not be performed on flows stored as malicious, to prevent the possibility of revising them as benign. Therefore, the analysis must consist of a benign flow revision, predicting if their label remains benign or becomes malicious, with the possibility of specifying a malicious class.

The alerts validated by end users correspond to flows stored as malicious. Additionally, the users must also be able to validate the “non-alerts”, the flows stored as benign. This is a safeguard to enable a manual search for undetected malicious activity.

From a security standpoint, it is essential to adopt the principle of least privilege. Consequently, access control must be performed to authenticate and authorize every interaction with the system. To provide control over the watchdogs and users allowed to access the system, these must be managed by users with administrator permissions. Since sensitive data is handled, the system is required to comply with the General Data Protection Regulation (GDPR) [94].

Considering the mentioned aspects, several functional and non-functional requirements were established to ensure the quality of the solution. These requirements were structured according to the categories of the FURPS+ model [65]:

- **Functionality**
  - Watchdogs must be able to store network flows;
  - Users must be able to view and validate network flows;
  - Administrators must be able to manage the watchdogs and users allowed to access the system;
  - The system must revise the stored benign flows;
  - The system must utilize the flows validated by users to continuously improve the detection;
  - The principle of least privilege must be enforced with access control;
  - The GDPR must be complied with.
- **Usability**
  - An easy-to-use and consistent interface must be provided;
  - GDPR-related information must be accessible to all users.
- **Reliability**
  - The flow revision must be up-to-date with the network topology;
  - Fault-tolerance must be enforced.
- **Performance**
  - The flow revision must be performed in real-time;
  - Malicious activity alerts must be quickly accessible.
- **Supportability**
  - IoT devices must be able to communicate with the system;
  - The system must be scalable;
  - The system must be easily configurable;
  - The system must be well-documented.
- **Design Constraints**
  - The system must be modular;
  - The system must follow software design best-practices.

The development of the solution was documented according to the levels and views of the C4 [95] and 4+1 [96] architecture visualization models. All the diagrams of the following sections were created using the Unified Modeling Language (UML) representation.

3.2 Domain Concepts

To fulfil the requirements, the system is based several conceptual classes, described below. Figure 7 displays the domain model.

Watchdog. Represents any device that is allowed to access the system to store network flows. A watchdog is identified by its assigned name, which must be unique, and can optionally have a more detailed description. The name of a watchdog will be utilized in the device authentication process, so it is required to have a minimum of 8 characters to enforce security best-practices.

Malicious/Benign Flow. Represent an abstract network flow that has been labelled as either malicious or benign, being considered as an alert or non-alert. Due to the large quantity of benign activity associated with IoT systems, alerts are expected to be a minority among non-alerts. Therefore, the separation of the malicious and benign flows fulfils the requirement of providing quick access to all the alerts, when stored in a database.

Additionally, malicious flows have a *MaliciousType* attribute, representing a specific type of attack. To support both binary and multi-class classification simultaneously, it is always utilized and is considered as "Unknown" when unspecified. For instance, a watchdog can perform binary classification with unspecified types of attacks, while the system performs multi-class classification.

Flows captured in a watchdog all have different timestamps, even if the variation is a millisecond. However, flows from different watchdogs can have the same exact timestamp. Therefore, these attributes can be repeated individually, but must be unique when combined. To provide a single identification attribute, a unique generated *FlowCode* is utilized.

A flow is created with a *FlowStatus* attribute as "Analysed", expressing the initial label. When a benign flow is revised by the system, its status is updated to "Reanalysed". When any flow is validated by an end user, it is marked as either "Correct" or "Incorrect". Then, when the system utilizes it to improve the detection, its final status is "Archived" and no more status updates are allowed.

To provide the relevant information to an end user, stored flows are required to have valid origin and response IP addresses, as well as valid origin and response ports. The features utilized for the flow analysis are not meant to be visible to an end user and are placed in the *FeatureList* attribute, as a way of permitting configurable feature layouts instead of forcing the same layout for every implementation.

User. Represents an end user. All users are identified by their unique username, which must be unique, and have a password, which is encoded by the system. These credentials will be utilized in the user authentication process, so they are required to have a minimum of 8 characters to enforce security best-practices.

The *UserRole* attribute creates the distinction between regular users and administrators, represented by the "Admin" role. To permit more restrictive policies regarding information access, two roles were created, namely "AlertsOnly" and "FullAccess". The first is meant to only validate alerts, whereas the latter is meant to validate all the captured flows, which corresponds to both alerts and non-alerts. This applies the principle of least privilege to restrict the access of each type of user.

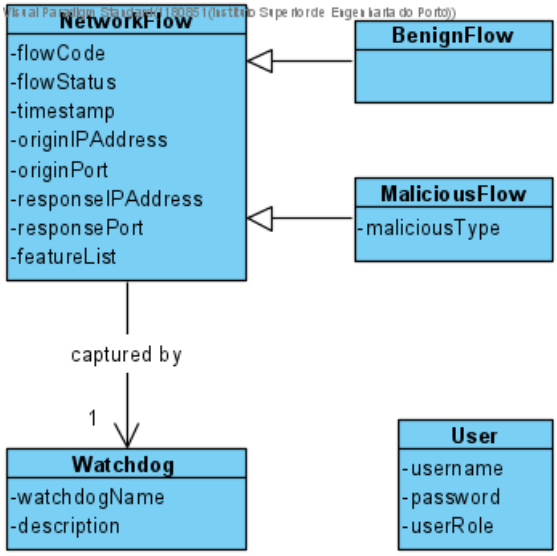

Figure 7. Domain model.

A DDD approach is employed to structure the development of the solution on the domain concepts and logic. For that purpose, the five conceptual classes are converted into entities, which are part of aggregates, and their attributes become value objects. The main improvement is the encapsulation of the malicious and benign flows as a single aggregate, *Network Flow*. This aggregate utilizes the *WatchdogName* value object to represent the "captured by" association, instead of directly referencing the *Watchdog* aggregate. Figure 8 displays the domain model with a DDD approach.

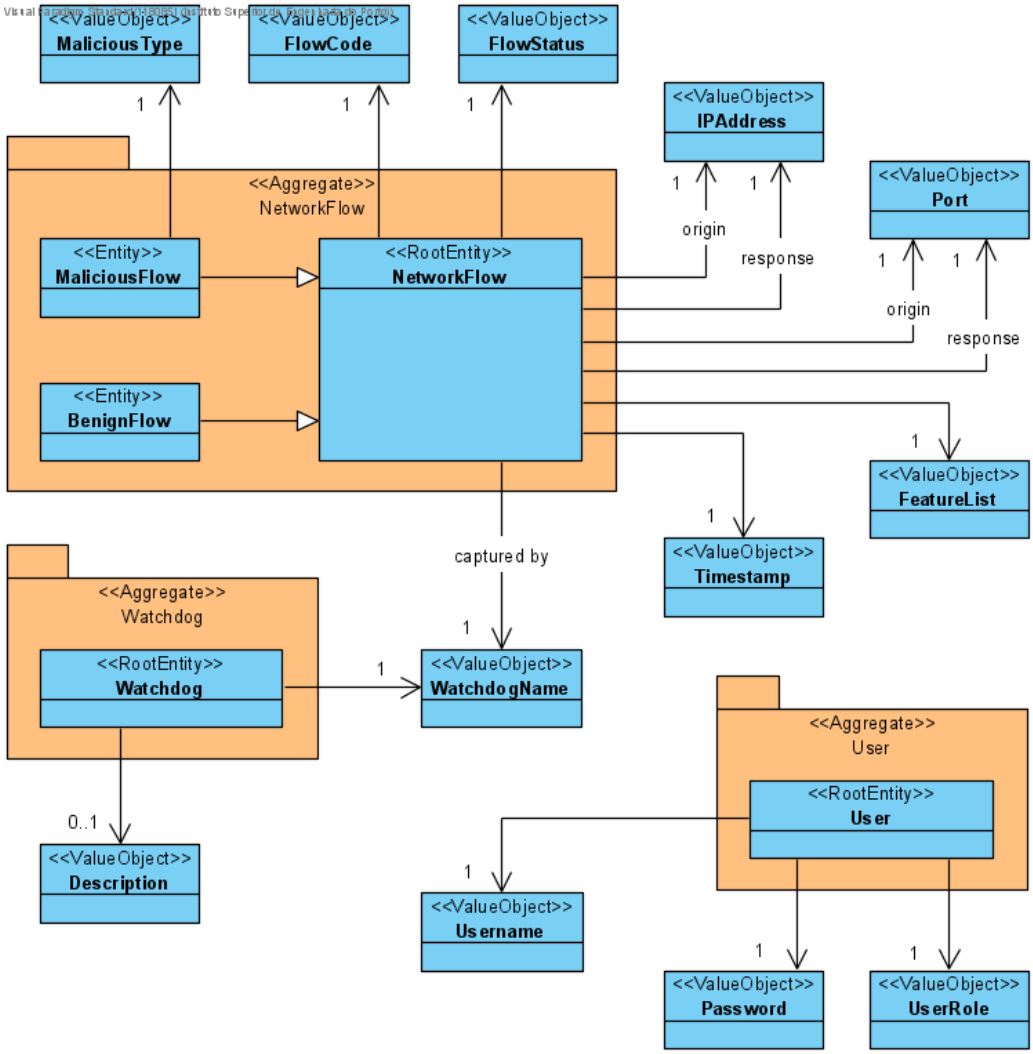

Figure 8. Domain model with Domain Driven Design.

3.3 Scenarios and Use Cases

External entities, referred to as actors, interact with system through six established scenarios. Allowed watchdogs can only store network flows, whereas end users can sign in and then perform distinct actions according to their role. Figure 9 displays the scenarios view.

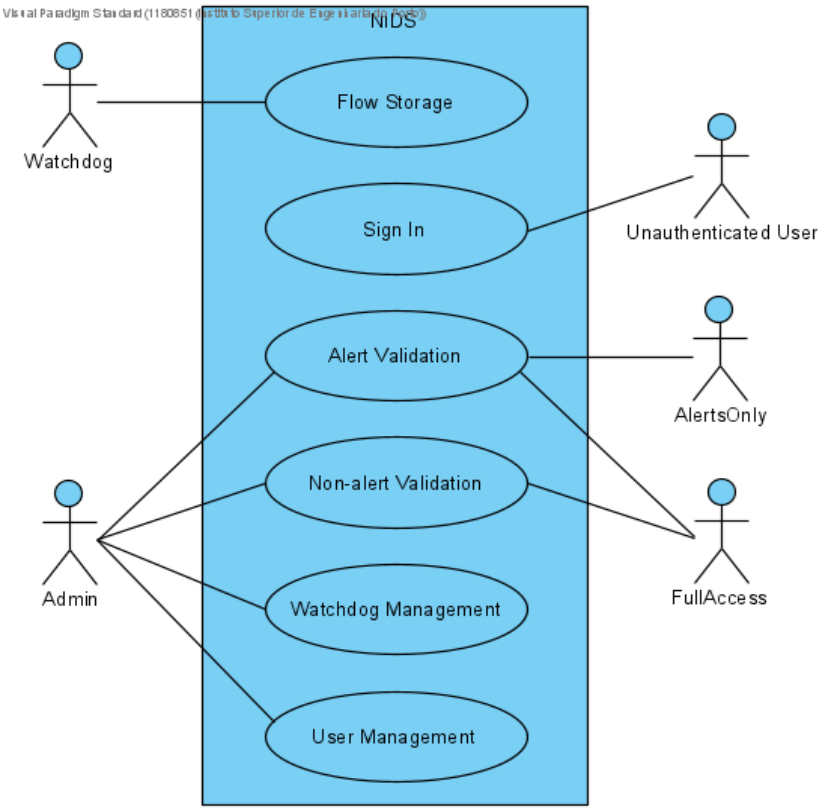

Figure 9. Scenarios view.

Each scenario can be converted to a sequence of interactions between an actor and the system, described below.

Flow Storage. An allowed watchdog performs a single action, which is the storing of network flows and corresponding labels, according to the attributes required by the domain. Figure 10 displays the process view of this scenario.

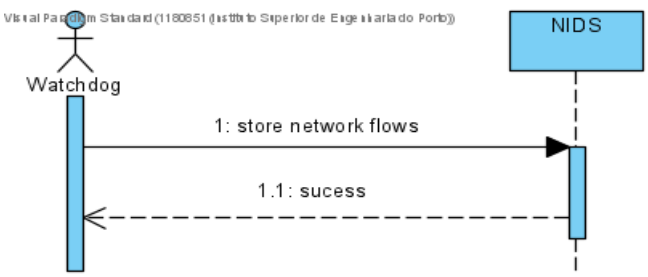

Figure 10. Process view of the Flow Storage scenario.

Sign In. An unauthenticated user attempting to gain access the system, being requested a username and a password. When correct credentials are inserted, the sign in is successful. Figure 11 displays the process view of this scenario.

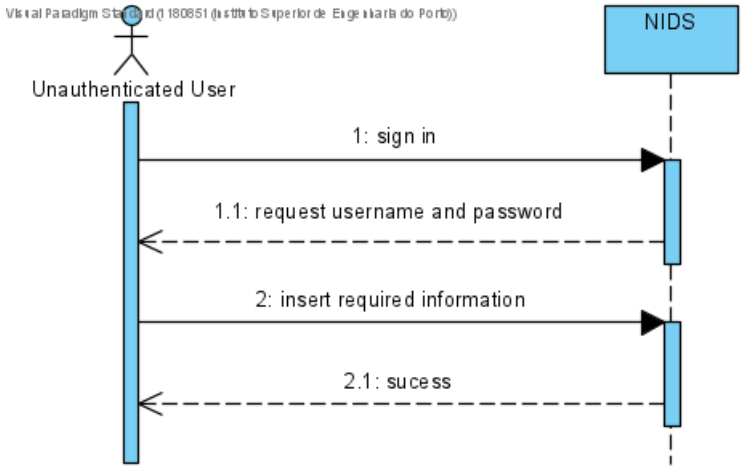

Figure 11. Process view of the Sign In scenario.

Alert/Non-alert Validation. Both scenarios are equivalent, starting with an initial listing of the unvalidated data, followed by the actual marking as correct or incorrect. Figure 12 displays the process view of the Alert Validation scenario.

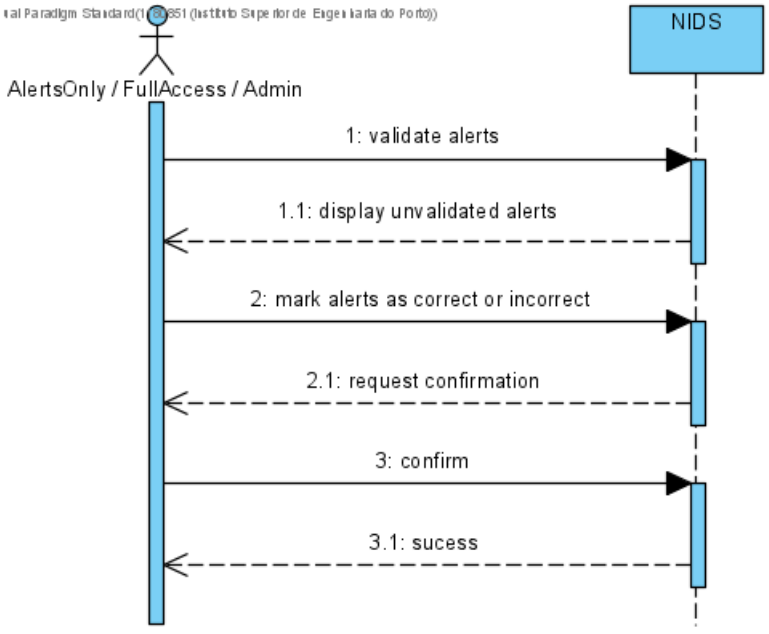

Figure 12. Process view of the Alert Validation scenario.

Watchdog/User Management. Both scenarios are equivalent and start with a listing of the allowed watchdogs/users. Then, an administrator can either opt to allow a new one or to disallow existing ones. For the first, the attributes required by the domain must be inserted, whereas for the latter a simple selection must be made. Figure 13 displays the process view of the Watchdog Management scenario.

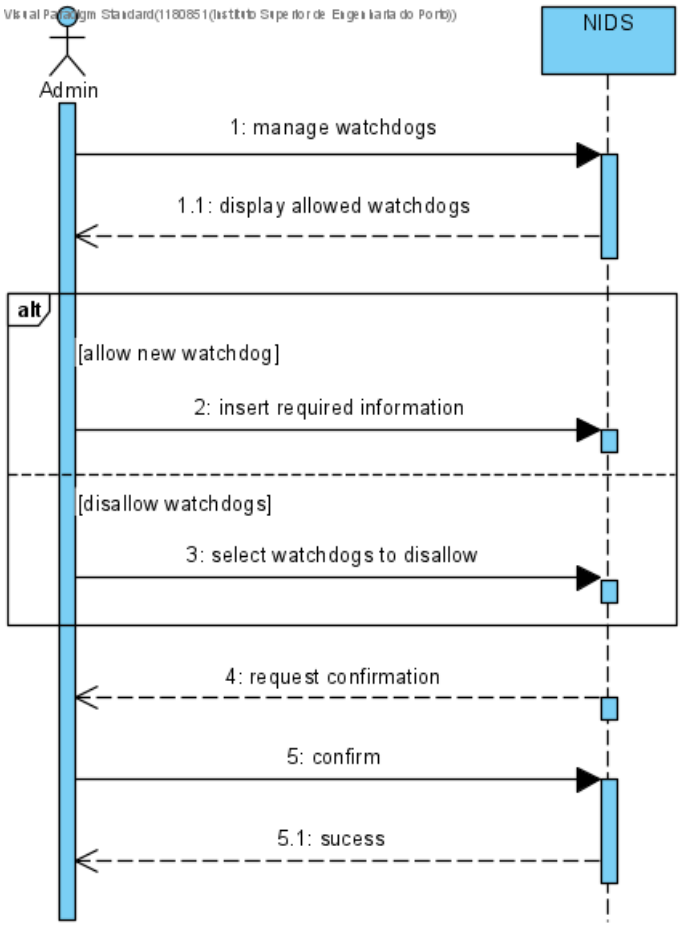

Figure 13. Process view of the Watchdog Management scenario.

A sequence of interactions can combine multiple use cases of the system, which are summarized in Table 13. The established use cases can be aggregated into List, Validate, Allow and Disallow groups, relying on the restrictions imposed by the domain. Still, some use cases required additional considerations and restrictions, which are described below.

Table 13. Summary of established use cases.

Use Case	Title	Scenario
UC1	Store Network Flows	Flow Storage
UC2	Sign In	Sign In
UC3	List Unvalidated Alerts	Alert Validation
UC4	Validate Alerts	Alert Validation
UC5	List Unvalidated Non-alerts	Non-alert Validation
UC6	Validate Non-alerts	Non-alert Validation
UC7	List Allowed Watchdogs	Watchdog Management
UC8	Allow New Watchdog	Watchdog Management
UC9	Disallow Watchdogs	Watchdog Management
UC10	List Allowed Users	User Management
UC11	Allow New User	User Management
UC12	Disallow Users	User Management

UC1, Store Network Flows. Both labelled and unlabelled flows can be stored. In the first case, they are created as malicious or benign. In the second case, they are all considered benign. This enables the system to benefit from watchdogs capable of performing a preliminary analysis, while also supporting more constrained monitoring devices.

UC8, Allow New Watchdog. When a new watchdog is successfully allowed, a token is generated and the administrator is expected to provide it to the physical device. This adds a human layer to the initial authentication process, effectively preventing unintended devices from being authorized to access the system. Considering that watchdogs can be scattered over an IoT system with a wide range of devices, it would not be a good practice to trust an unknown device trying to gain authentication.

UC9, Disallow Watchdogs. This use case was created to enable an administrator to immediately block suspicious watchdogs. Since it is inevitable that some tokens will be eavesdropped and even that some devices will be compromised, this enables an administrator to prevent the storage of distorted data.

UC10, List Allowed Users. Both the username and role of the user accounts are listed, but their passwords are never disclosed. An additional detail is that the administrator accessing the use case is not listed, since the objective is to view the remaining users.

UC11, Allow New User. It is an administrator who inserts the credentials of the new user account, selecting its role from the roles supported by the system. This gives administrators full control over new user accounts and their security.

UC12, Disallow Users. Following the same approach as UC10, the administrator accessing the use case cannot be removed, as a safeguarding measure.

Besides the specified use cases, every authenticated user can perform two simple universal actions, which are logging out and viewing the Privacy Policy. This document states how the system collects, processes and stores data, in compliance with the current GDPR guidelines (see 0).

3.4 Level 1

Overall, the solution can be described as an intermediary system connecting IoT monitoring devices to end users who are alerted to possibly malicious network activity. This system dynamically adapts to changes in network topology and attack patterns, continuously improving the detection.

From a logical standpoint, it is represented as a system providing a single standardized API for watchdog interaction, as well as UIs for end users. Database and filesystem APIs are consumed to store the information and obtain the DRL model, respectively. Figure 14 displays the logical view of level 1 of the C4 and 4+1 architecture visualization models.

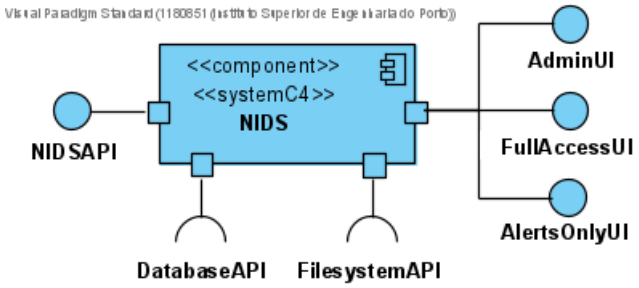

Figure 14. Logical view (Level 1).

3.5 Level 2

The system consists of three logical containers, each representing a fully functional application. The distinct containers and their responsibilities are described below. Figure 15 displays the logical view of level 2.

The Master Data is the main container of the system. It encapsulates the domain concepts and logic, ensuring data integrity and the database connection. It provides a REST API that is conceptually divided into an external interface for the watchdogs to store network flows, “NIDSAPI”, and an internal interface to be utilized by the other containers, “MasterDataAPI”. In that regard, the Master Data enforces access control, being responsible for the authentication and authorization of every actor.

The SPA container is the system’s entry point for user interaction. It provides UIs that work as intermediaries between Master Data’s internal API and the browsers of end users.

The Learning Agent container encapsulates the DRL model, which is obtained through the filesystem. It is responsible for the benign flow revision, as well as the improvement utilizing the user-validated alerts and non-alerts. Therefore, it is permanently interacting with Master Data’s internal API, behaving as an actor.

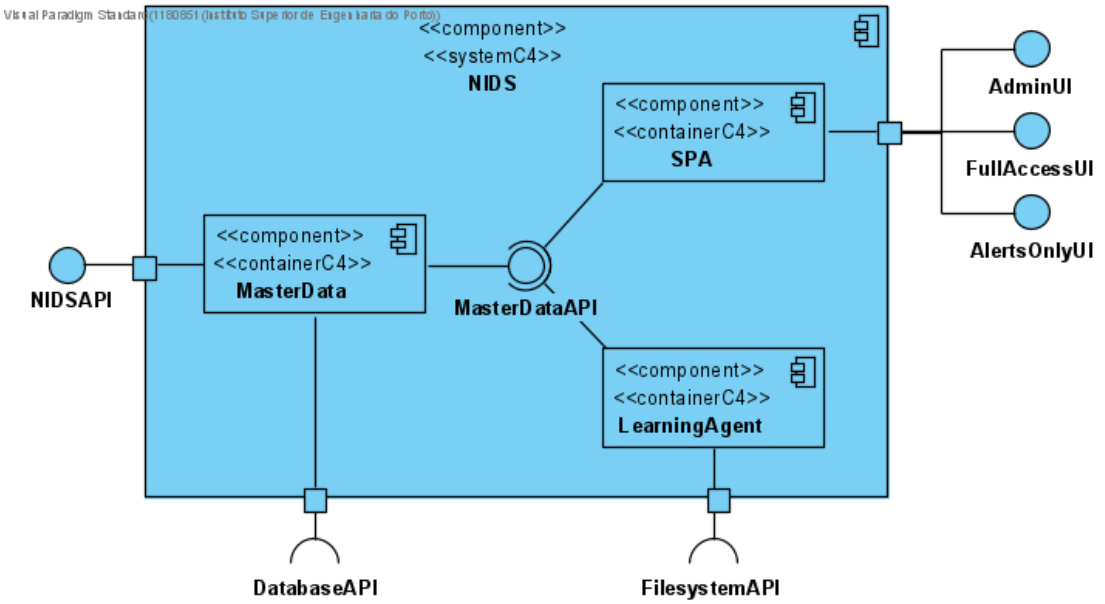

Figure 15. Logical view (Level 2).

From a software development perspective, each logical container corresponds to a software package and each API consumption to a dependency. Therefore, both SPA and Learning Agent packages depend on the Master Data package. Figure 16 displays the development view of level 2.

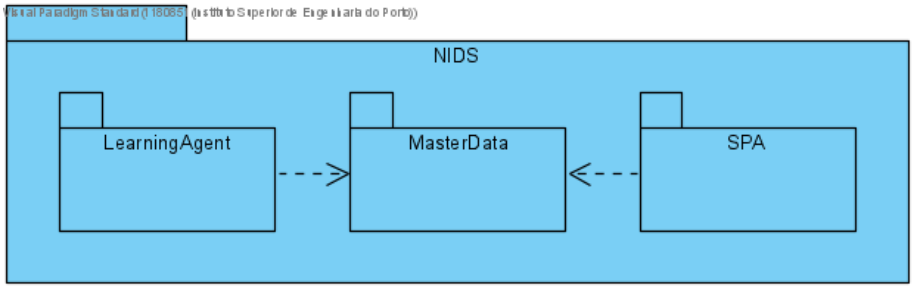

Figure 16. Development view (Level 2).

From a physical deployment perspective, each application can be deployed to an actual server or to a cloud solution. The REST architectural style is followed, so the applications communicate through HTTP/HTTPS requests authenticated by Master Data. Even though other communication protocols could be utilized, this approach standardizes the API. Figure 17 displays the physical view with the conceptual topology.

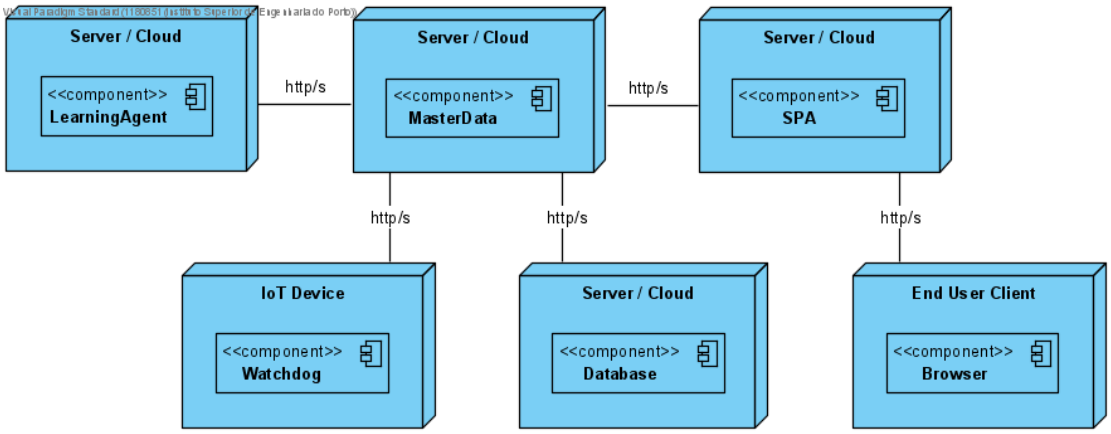

Figure 17. Physical view with conceptual topology (Level 2).

3.6 Level 3

The logical containers consist of several distinct components, which represent software subpackages with multiple OOD classes.

Overall, the design guidelines defined by the GRASP and SOLID principles are adopted. A responsibility is assigned to the class that has the required information or to a mediator between two classes, in conformity with the Information Expert and Indirection principles. This is connected to the Single Responsibility principle because each class serves a specific purpose. Additionally, the dependencies of a class are wrapped as interfaces, following the Protected Variations and Dependency Inversion principles.

Therefore, every class fulfils a specific responsibility and is isolated from its dependencies, achieving High Cohesion and Low Coupling. Figure 18 displays the general mapping between the logical and development views of this level, where each Type1 class depends on the interface exposed by a Type2 class.

The development of each container will be detailed in its respective sub-section.

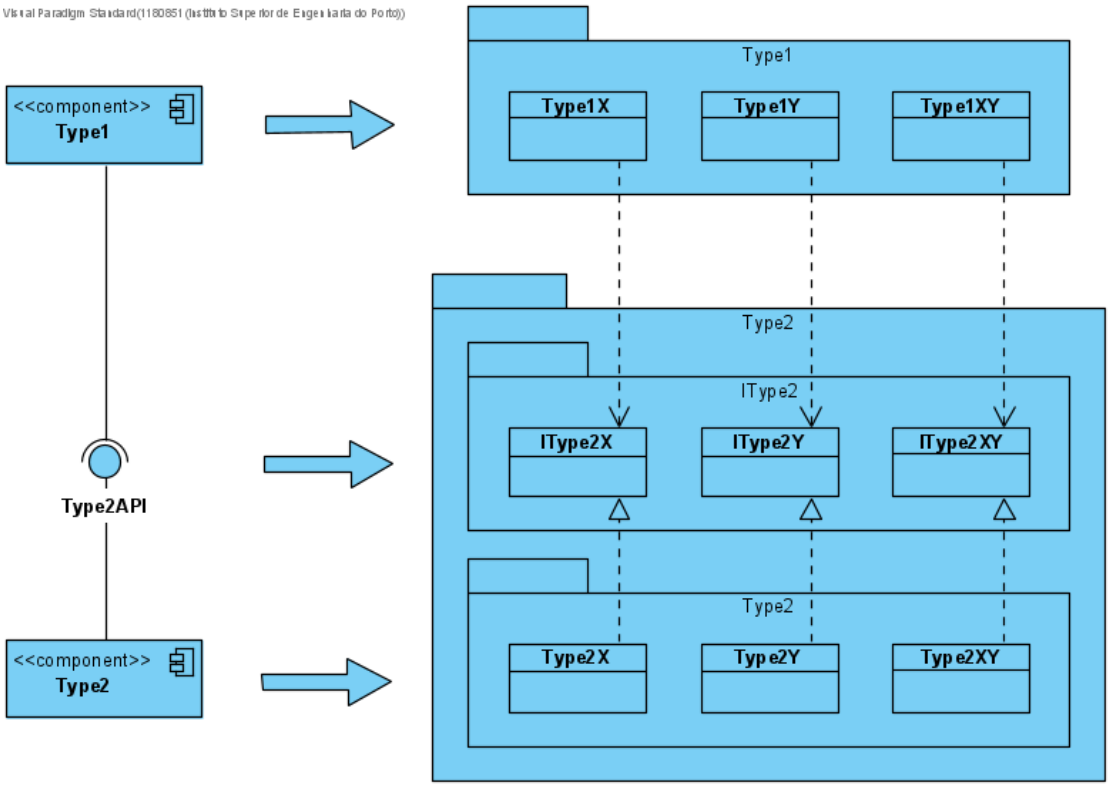

Figure 18. Mapping between logical and development views (Level 3).

3.6.1 Master Data

The Master Data container is based on the following components:

- ***Domain Model*** – Contains the domain concepts and logic;
- ***Application Services*** – Fulfil use cases by interacting with the *Domain Model*;
- ***Controllers*** – Coordinate use cases, applying the Controller pattern;
- ***DTOs*** – Carry data between the *Applications Services* and *Controllers*, applying the Data Transfer Object (DTO) pattern;
- ***Repositories*** – Wrap the communication with the *Persistence*, applying the Repository pattern;
- ***Persistence*** – Accesses a database;
- ***Data Model*** – Contains the data format utilized by the *Persistence*.

To structure these components into an application, both a standard layered architecture and an Onion architecture were considered. Figures 19 and 20 display the logical views with both alternatives.

A standard layered architecture organizes the components from bottom to top. The *Persistence* is the base of the application, followed by the *Repositories*, *Application Services* and *Controllers*. This is a scalable approach that successfully isolates the responsibilities of the different components. However, it makes the *Application Services* rely on the *Repositories* and the underlying *Infrastructure*.

On the other hand, an Onion architecture organizes the components from core to outer layers. The *Domain Model* and *Application Services* are the core of the application, being complemented by *Interface Adapters* that bridge the gap to the actual *Infrastructure*. Therefore, a robust core is created and the modularity of the application is improved, since the components of the outer layers can be replaced without impacting the inner layers.

Due to its benefits, the Onion architecture was employed. The corresponding software package is divided into four main subpackages, one per architectural layer. Figure 21 displays the development view of Master Data.

The Dependency Injection pattern is applied to provide the dependencies of each class, which consist of interfaces exposed by other classes. The exception is the *Application Services* layer, which depends on domain aggregates directly because it is part of the application core. Additionally, this layer exposes the interfaces of the set of *Repositories* it requires and the actual implementations are created in the next layer, fitting the needs of the application core.

To ensure the isolation of responsibilities, the Data Mapper pattern is applied to all conversions between data formats. Therefore, *Mapper* classes are utilized for the bidirectional conversions between the data format used by the *Routes* and *DTOs*, as well as between *DTOs*/*Domain Model* and *Domain Model*/*Data Model*.

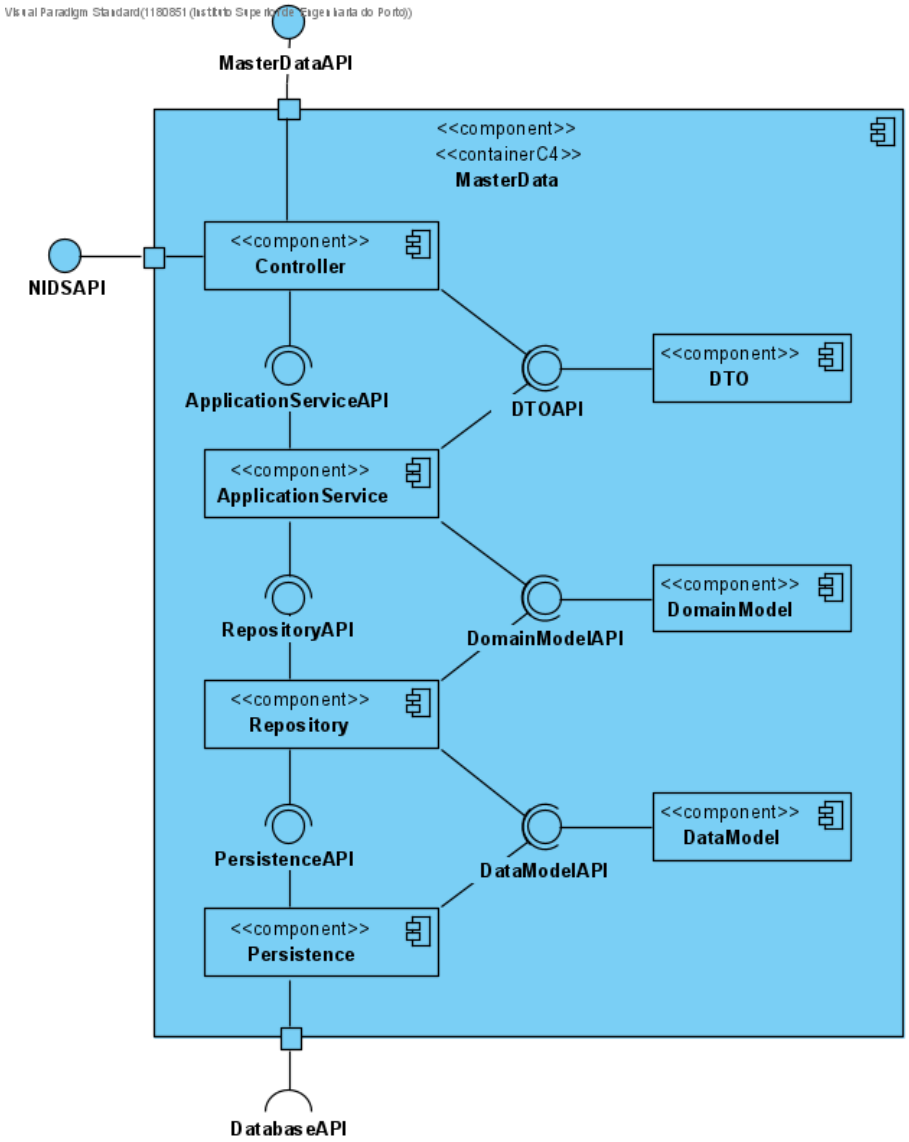

Figure 19. Logical view of Master Data with a standard layered architecture (Level 3).

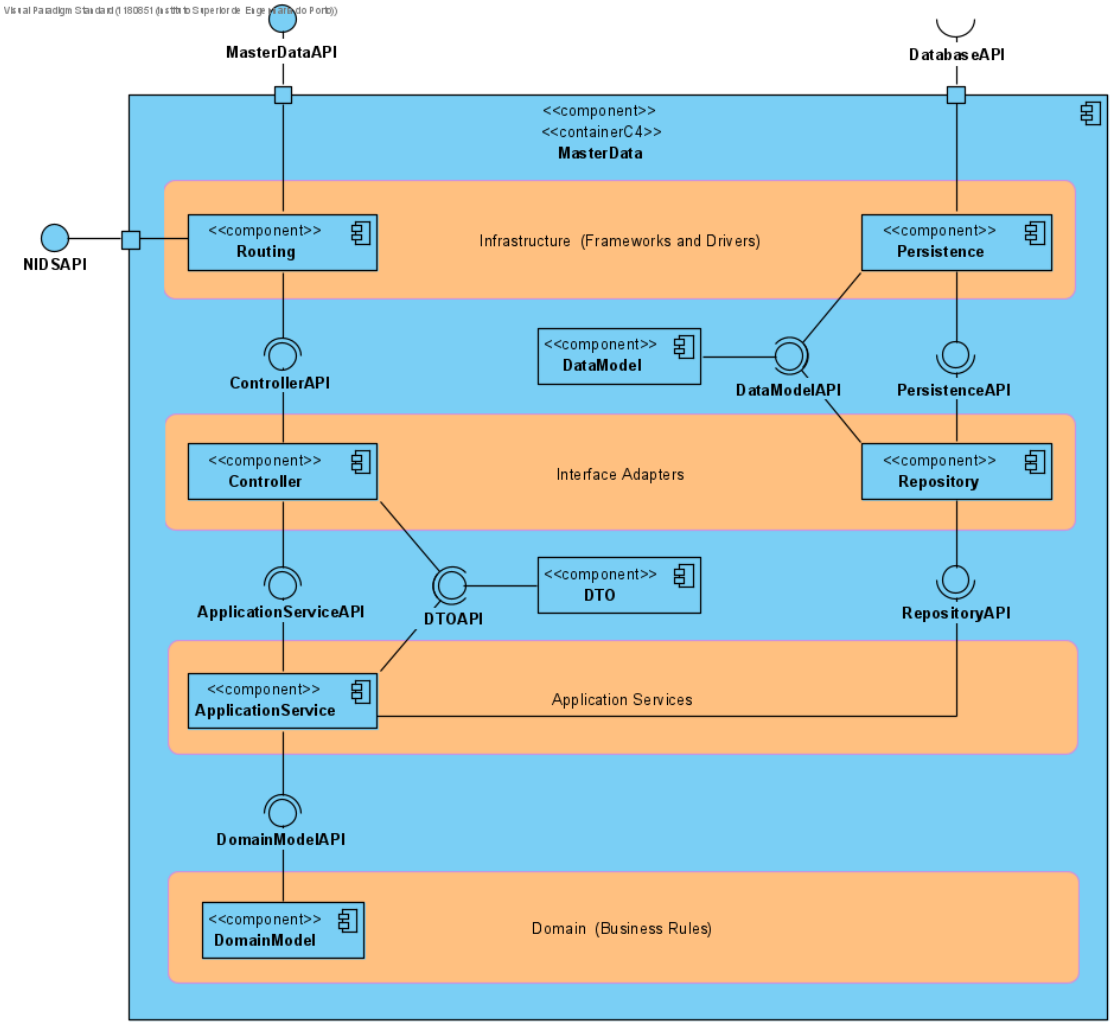

Figure 20. Logical view of Master Data with an Onion architecture (Level 3).

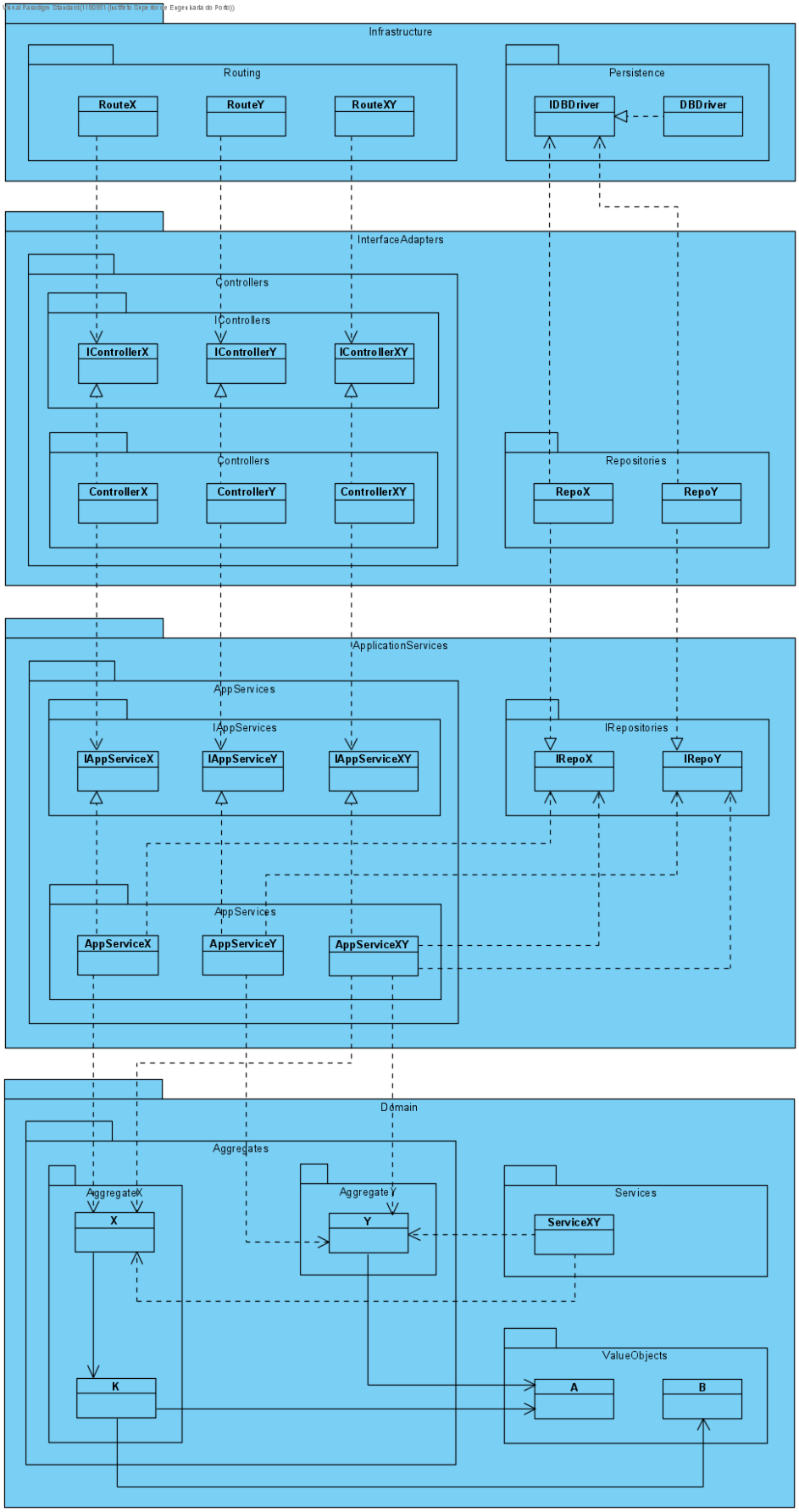

Figure 21. Development view of Master Data (Level 3).

The key aspect of the Master Data application is the access control, enforced by a Bearer scheme. A preconfigured secret is utilized to generate bearer tokens, which will be utilized to authenticate and authorize each actor.

Regarding the requests received from the Learning Agent, a predefined token is used in the authorization header. Similarly, the requests received from a watchdog must have an authorization header matching the token generated in UC8, when the watchdog was allowed.

In particular, end user authentication and authorization utilize a session cookie to store the token. When an end user signs in, a token is generated and sent in a cookie that will expire when the user session is terminated. Subsequently, all communication from the user's browser to SPA and from SPA to Master Data contains that cookie.

Additionally, the passwords received from SPA in both UC2 and UC11 are encoded in tokens, to avoid their transmission in plain sight. The passwords are then decoded using a secret that matches the one utilized by SPA to encode it.

To enforce security best-practices, in UC11, the password of the new user account is hashed and salted in a one-way encryption operation, which utilizes another preconfigured secret. The value stored in the database is the concatenation of both hash and salt. In UC2, to access the correctness of the received password, it is encrypted using the stored salt and then compared to the stored hash.

To exemplify how Master Data carries out the List, Validate, Allow and Disallow groups of use cases with the employed architecture and access control, Figures 22 to 25 display the process views of UC3, UC4, UC8 and UC9.

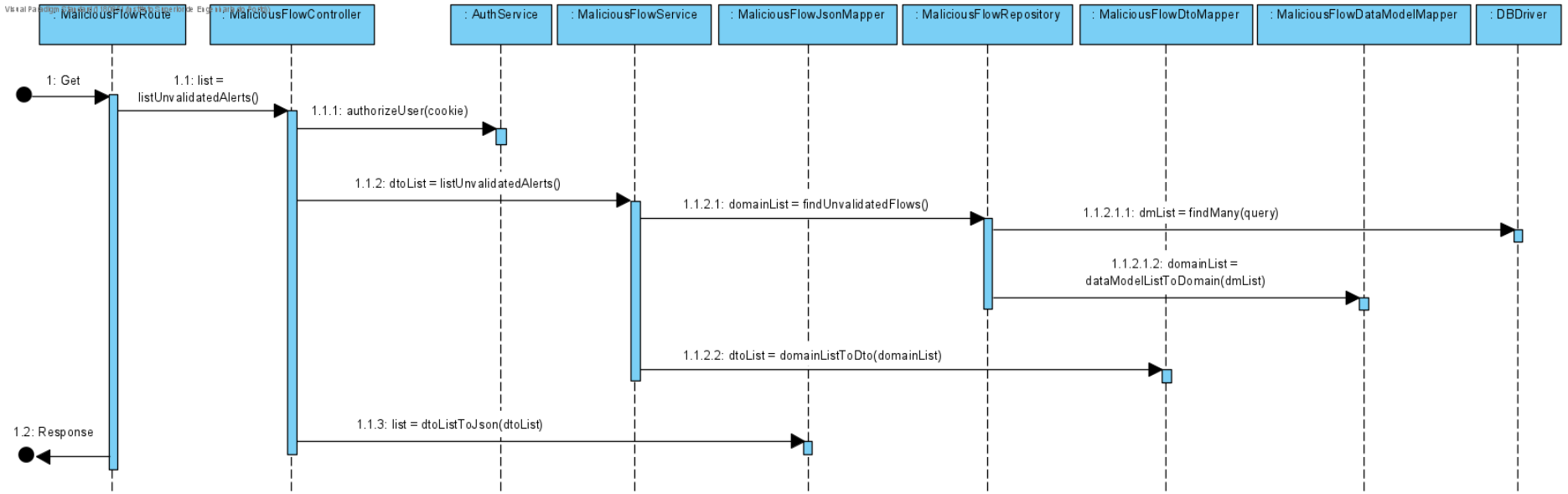

Figure 22. Process view of UC3 in Master Data (Level 3).

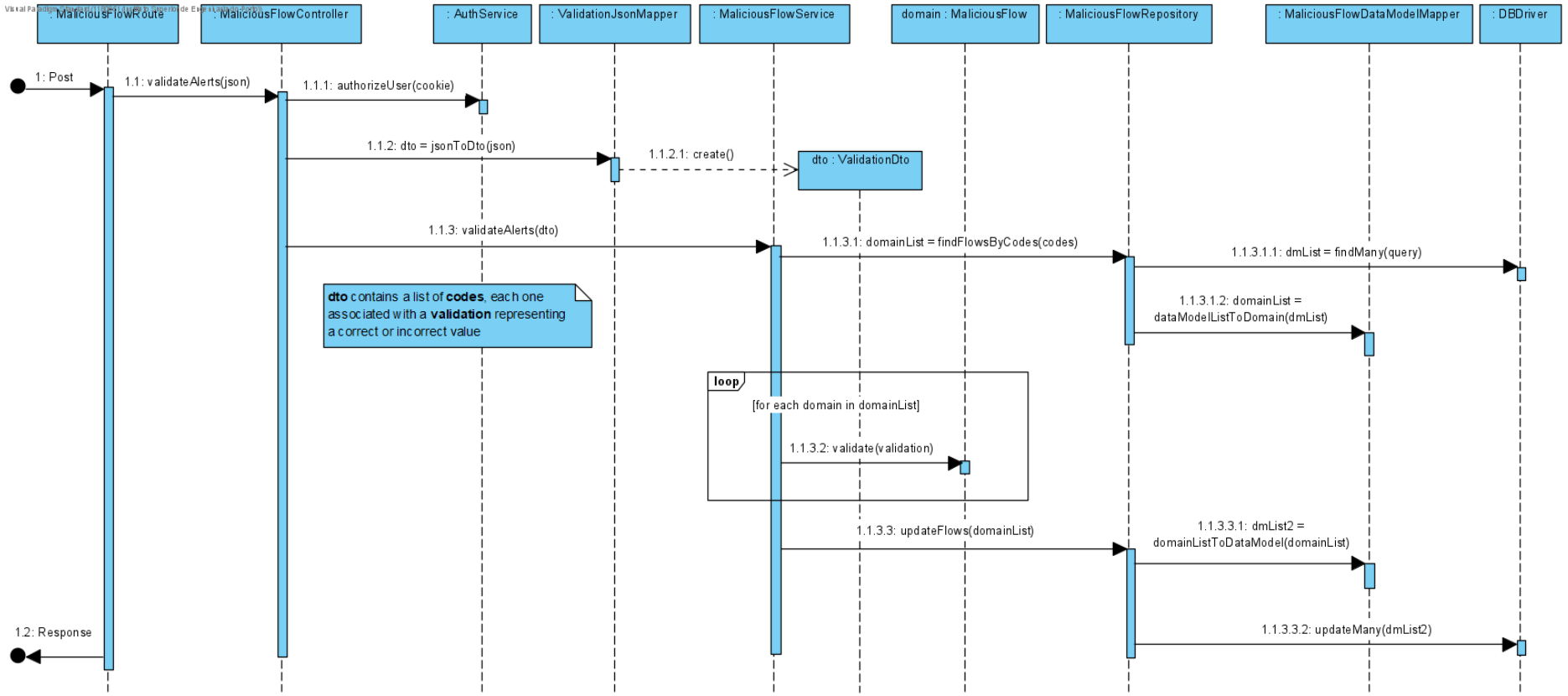

Figure 23. Process view of UC4 in Master Data (Level 3).

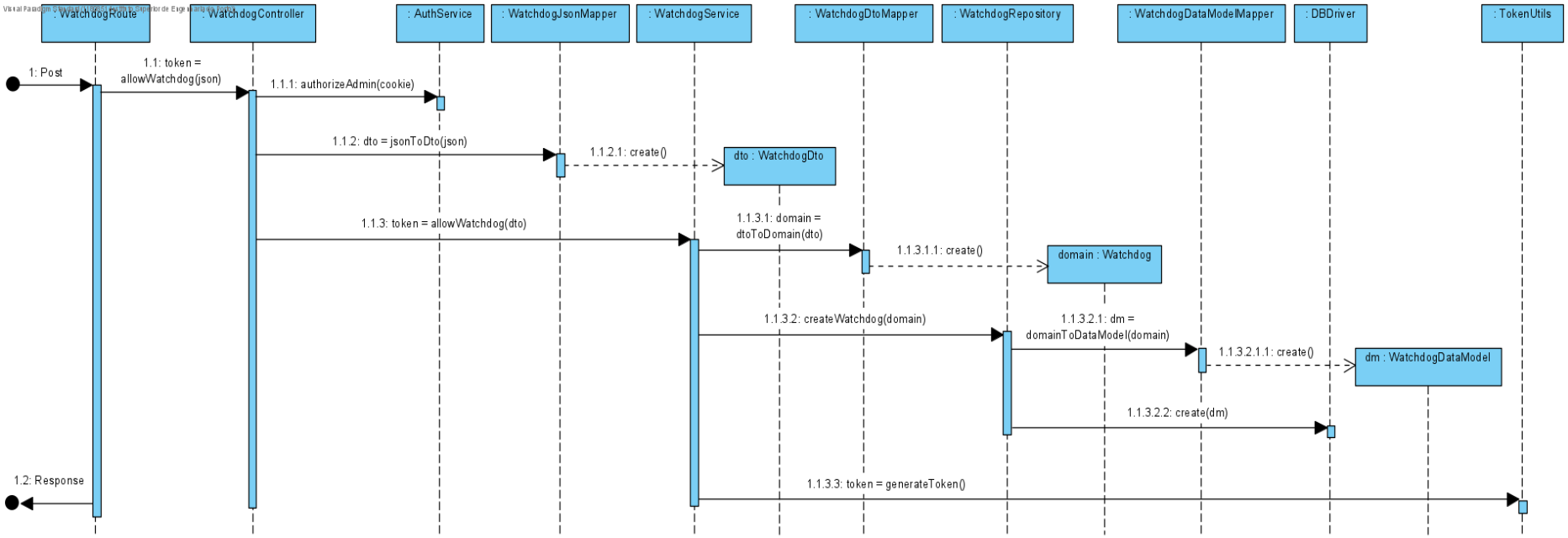

Figure 24. Process view of UC8 in Master Data (Level 3).

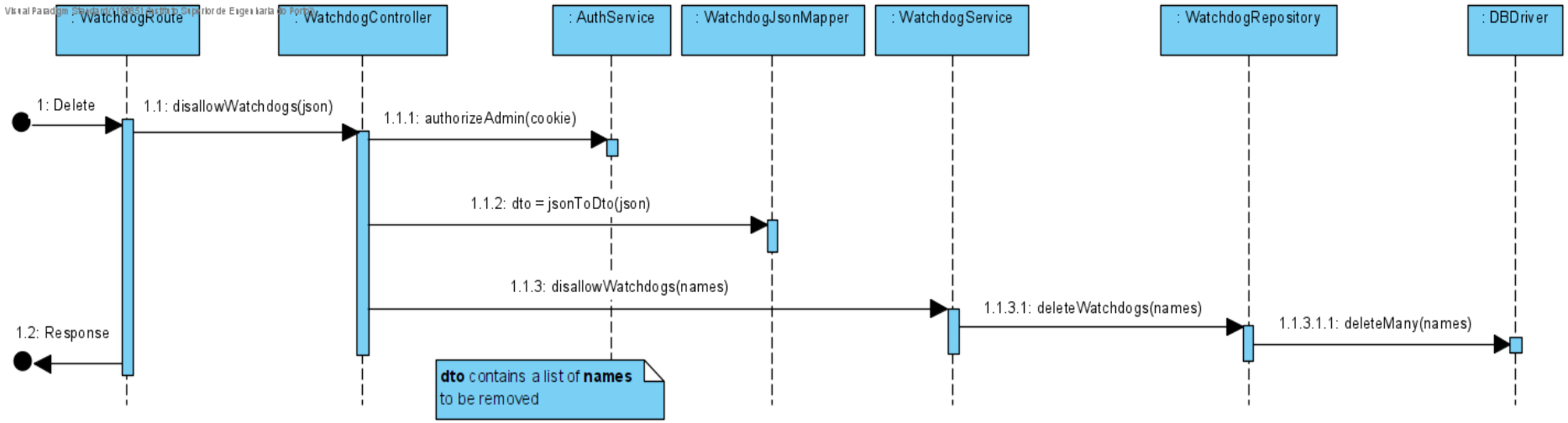

Figure 25. Process view of UC9 in Master Data (Level 3).

3.6.2 Single-Page Application

The SPA container is based on the following components:

- ***Adapters*** – Wrap the communication with Master Data's API, applying the Adapter pattern;
- ***DTOs*** – Carry data between the *Adapters* and the rest of the application, applying the DTO pattern;
- ***Views*** – Display output to end users and obtain their input.

To structure these components into an application, both an MVP architecture and an MVVM architecture were considered. Figures 26 and 27 display the logical views of SPA with both alternatives.

An MVP architecture provides a straightforward application with *Presenters* that coordinate entire sequences of user interactions, together with the *Views*. However, this leads to a centralization of responsibilities, which significantly increases the coupling of what should be isolated components.

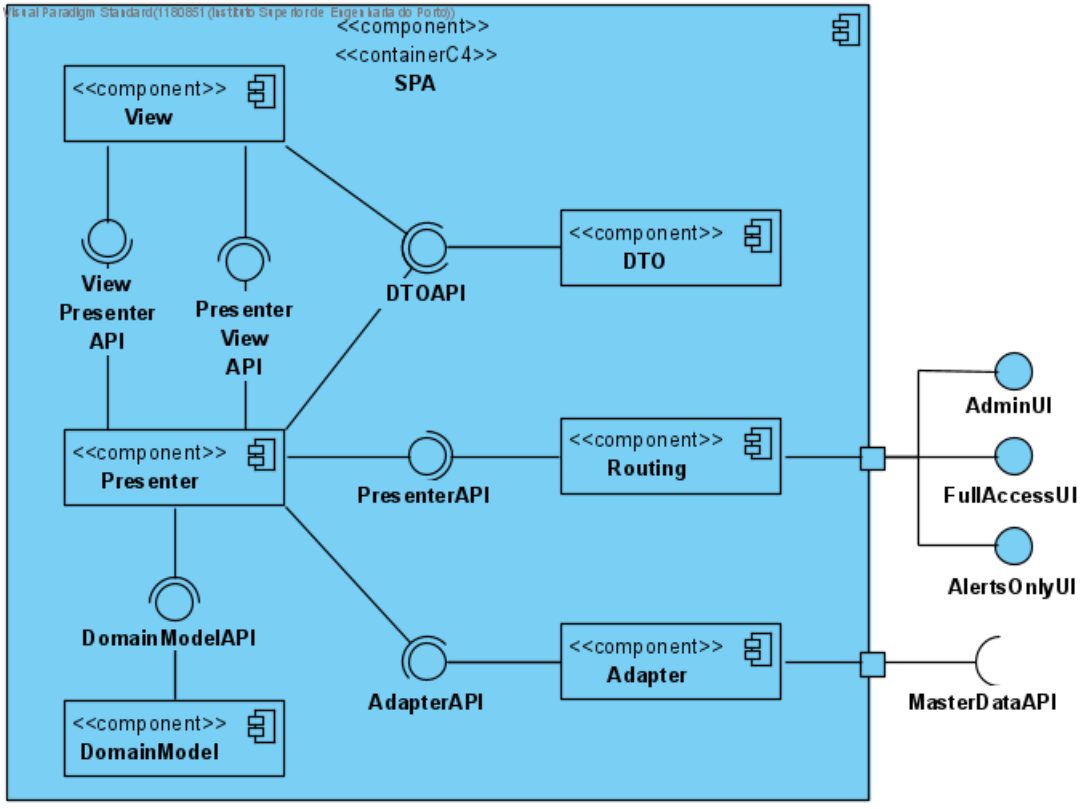

Figure 26. Logical view of SPA with an MVP architecture (Level 3).

On the other hand, an MVVM architecture utilizes *View Models* as intermediaries, separating the *Views* from the rest of the application. Furthermore, *View Controllers* can be used to coordinate user interaction, achieving the decoupling of this responsibility from the actual input and output of the Views.

It is important to note that although the *Domain Model* is represented in the logical views of the architectures, it is fully replaced by the *Adapters*. Therefore, instead of directly converting *DTOs* to domain aggregates, the application relies on the information obtained by the *Adapters*. This approach decouples the domain logic from SPA, safeguarding its encapsulation in Master Data.

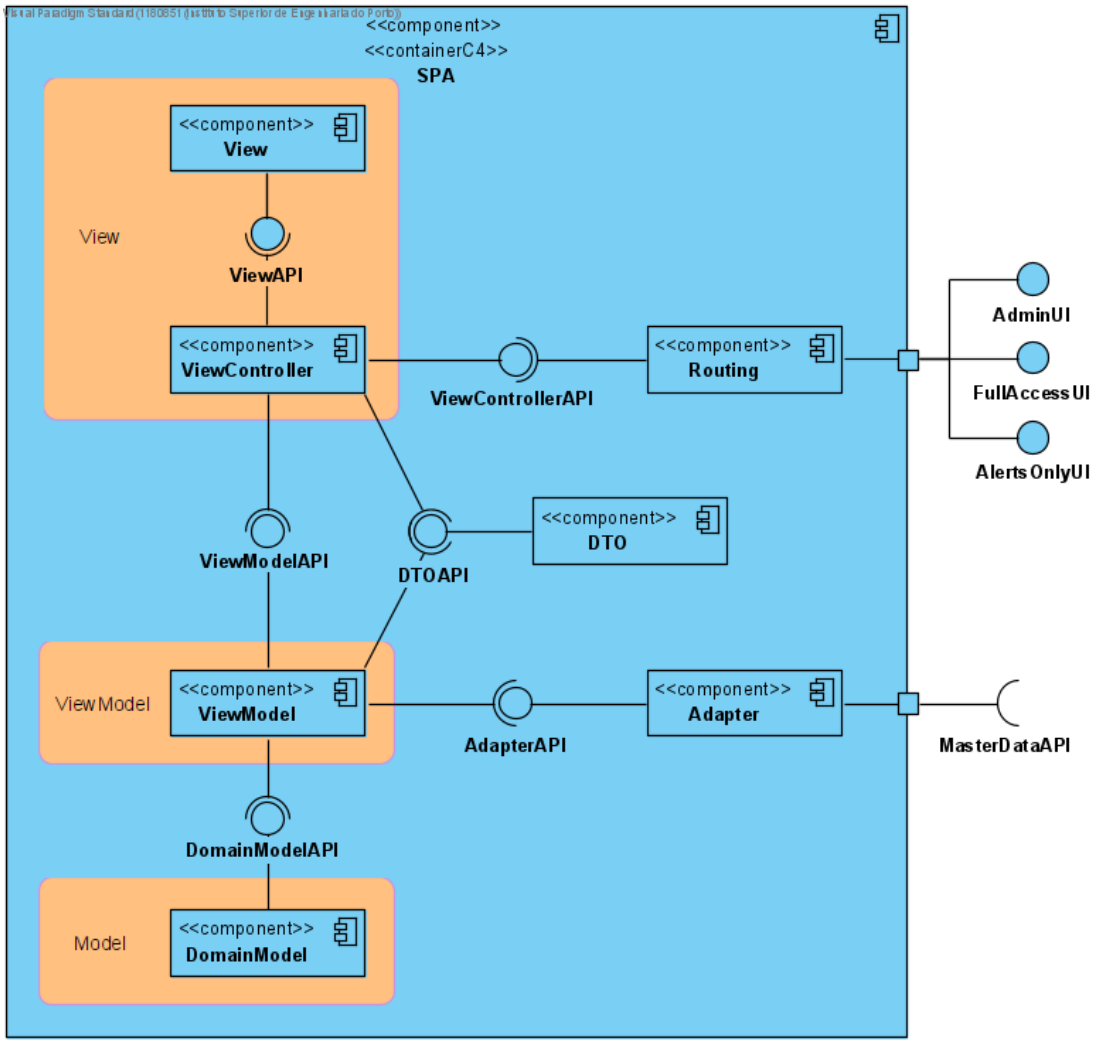

Figure 27. Logical view of SPA with an MVVM architecture (Level 3).

Due to its benefits, the MVVM architecture was employed. The corresponding software package is divided into four main subpackages, covering the *Views*, *View Models*, *Adapters* and *Routing*. Figure 28 displays the development view of SPA.

Following the adopted guidelines, the classes of a package depend on the interfaces exposed by the classes of other packages and the Dependency Injection pattern is applied, a similar approach to Master Data.

Replicating the approach of Master Data, the Dependency Injection pattern is applied to provide the dependencies of each class and the Data Mapper pattern is applied to the bidirectional conversions between the data format used by the *Adapters* and *DTOs*.

However, the *Routes* depend directly on the *View Controllers*, which in turn depend directly on the *Views*. Since these classes provide the entry point for user interaction, they are interconnected and their implementations will not be individually replaced.

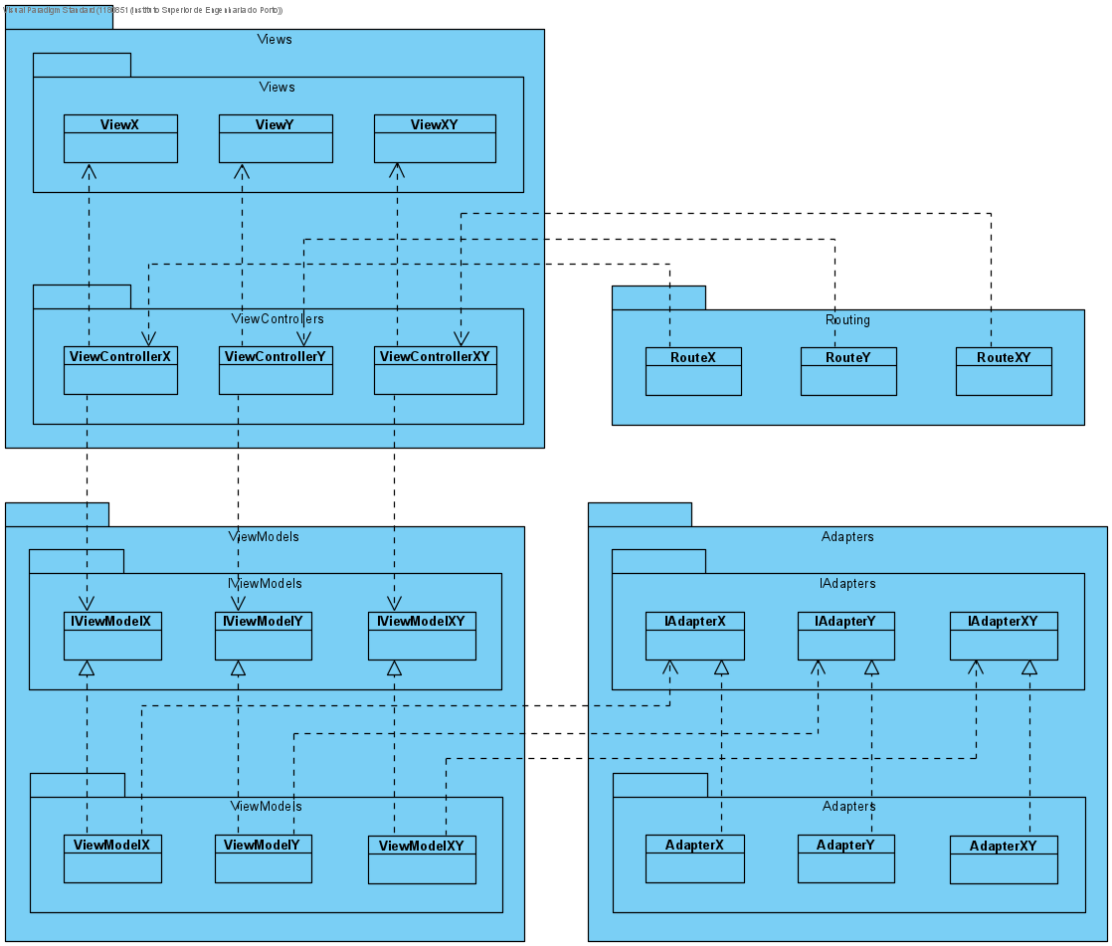

Figure 28. Development view of SPA (Level 3).

As described in the previous sub-section, the passwords inserted in both UC2 and UC11 are encoded in a token before being sent to Master Data, to avoid their transmission in plain sight. This is performed using a preconfigured secret that matches the one utilized by Master Data to decode it.

To exemplify how SPA carries out the List, Validate, Allow and Disallow groups of use cases with the employed architecture, Figures 29 to 32 display the process views of UC3, UC4, UC8 and UC9. These correspond to the process views of Master Data.

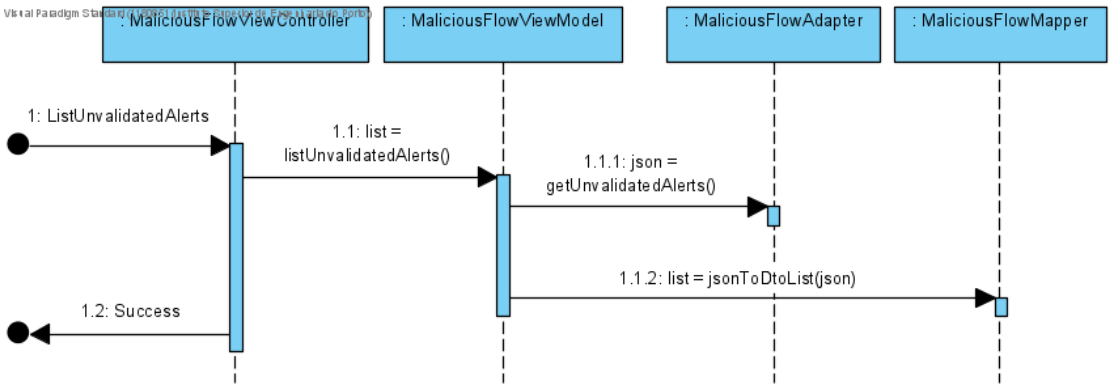

Figure 29. Process view of UC3 in SPA (Level 3).

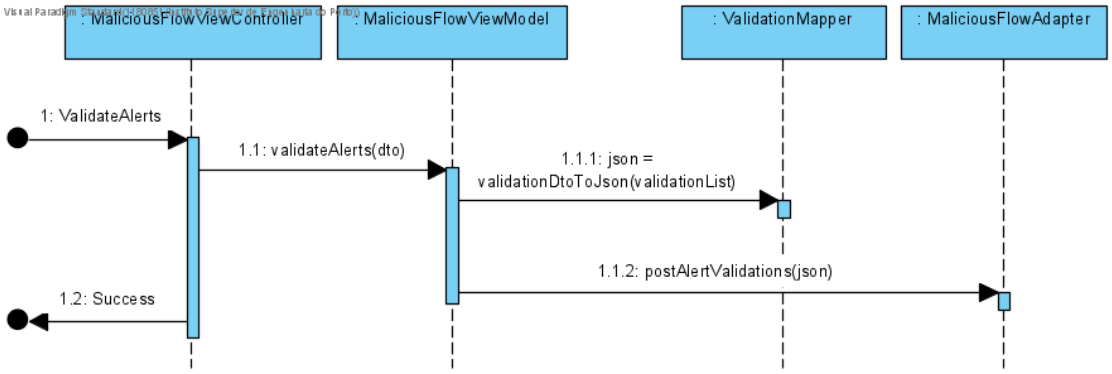

Figure 30. Process view of UC4 in SPA (Level 3).

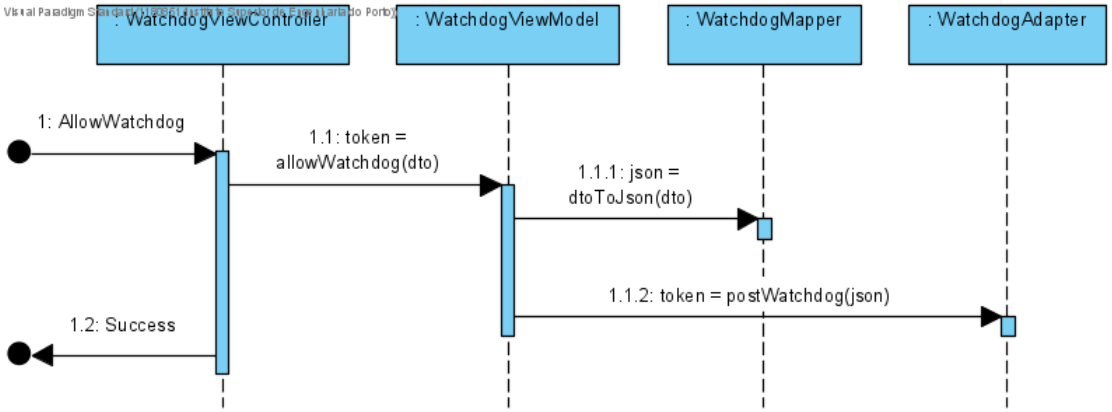

Figure 31. Process view of UC8 in SPA (Level 3).

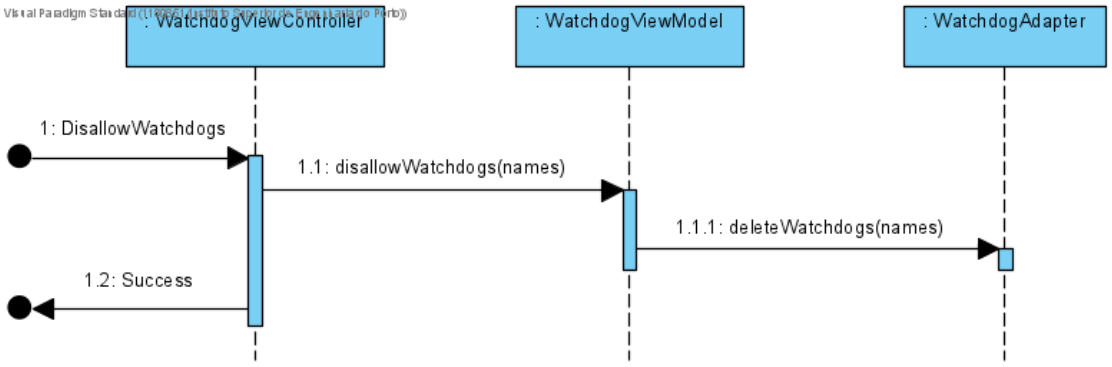

Figure 32. Process view of UC9 in SPA (Level 3).

3.6.3 Learning Agent

The Learning Agent container faces the challenge of simultaneously performing two tasks, namely revising benign flows and improving the utilized DRL model. The first corresponds to a real-time analysis, whereas the second consists of computationally intensive experience replay sessions, utilizing the validated flows as past experiences.

Since these two tasks must be executed in parallel, both multithreading and multiprocessing were considered for the execution in a multicore CPU. Even though multiple threads are more lightweight than processes, they share the initial memory and resources. On the other hand, multiple processes are fully isolated, which significantly improves the performance of computationally intensive tasks [97]. For that reason, multiprocessing was employed to perform the revision on an isolated process, referred to as REV, while performing the replay sessions on the main process, referred to as REP.

Considering the isolation of the two processes, the Learning Agent container is based on the following components:

- ***Agents*** – Encapsulate a DRL model;
- ***Adapters*** – Wrap the communication with Master Data's API, applying the Adapter pattern;
- ***Controllers*** – Coordinate the processes, applying the Controller pattern.

It is important to note that the *Agents* expect a previously trained model, suitable for continuous analysis and improvement. These components fully isolate the responsibility of handling of a model from the rest of the application, which includes accessing the filesystem to load the initial model and perform backups of the improved versions.

A simple architecture was employed to structure these components. The corresponding software package applies the Dependency Injection pattern to provide the *Adapters* and the *Agents*, enabling their replacement for different implementations. Figures 33 and 34 display the logical and development views.

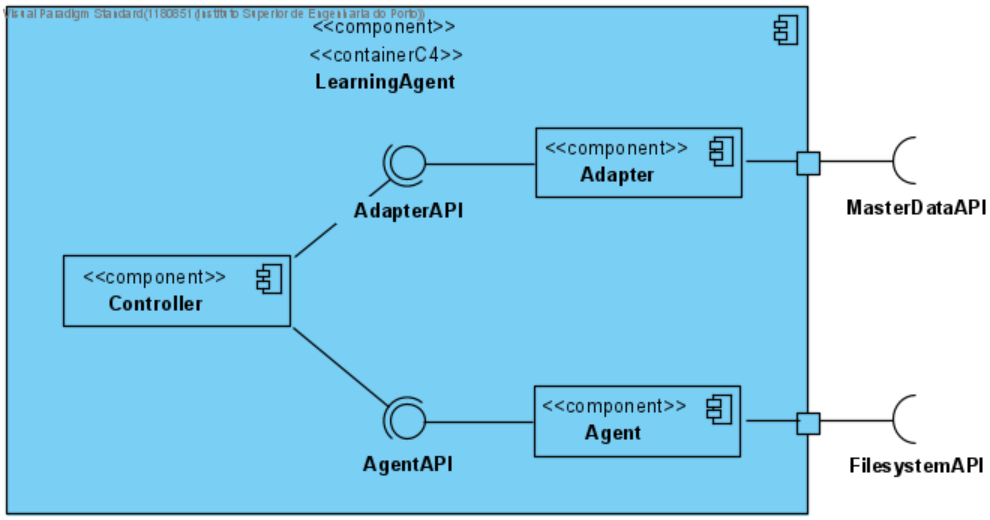

Figure 33. Logical view of Learning Agent (Level 3).

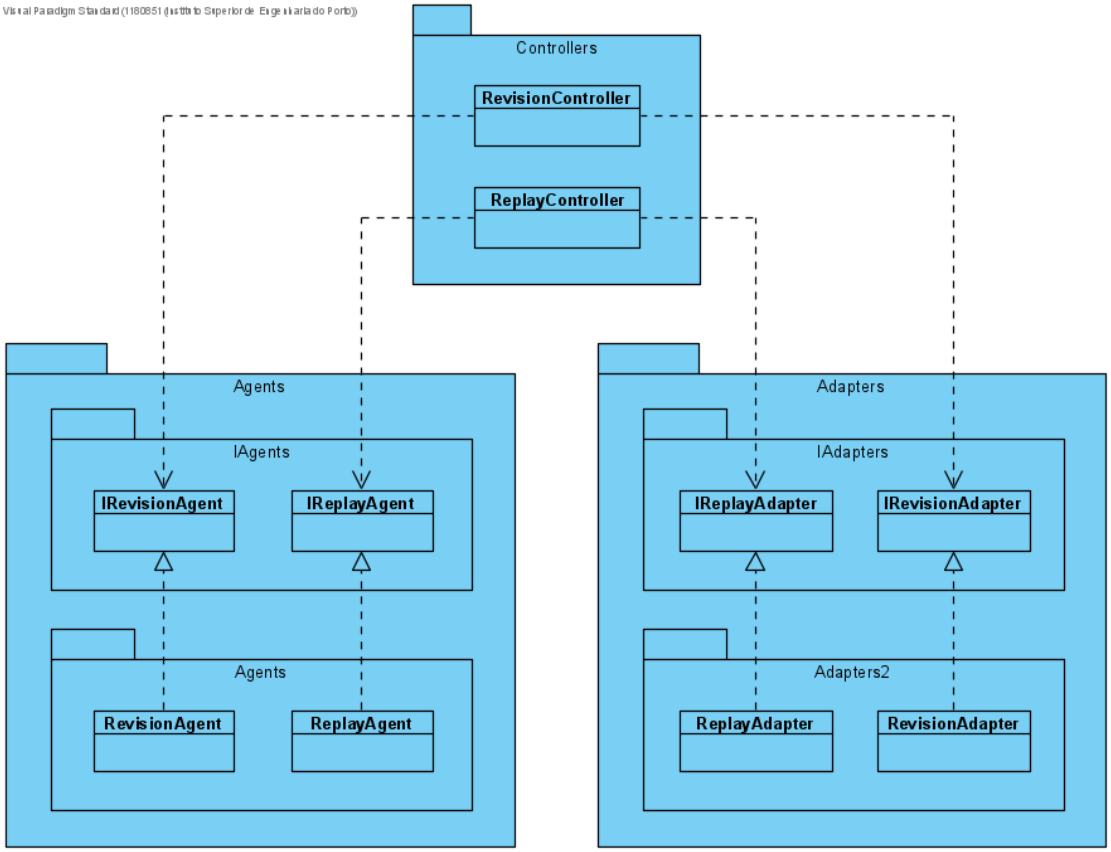

Figure 34. Development view of Learning Agent (Level 3).

To fulfil the requirement of keeping the flow revision up-to-date with the network topology, when REP finishes a training session, the model used by REV must be updated to the new version. Transmitting an entire model to REV would require it to be memory-mapped on disk due to its large size. Since an experience replay session only improves the weights of a model without changing its structure, the most efficient approach is to solely transmit these new weights.

For that purpose, a unidirectional connection is utilized to send the weights from REP to REV. Additionally, both processes communicate through a signal with the following states:

- ***Activated*** – REP sent new weights and activated the signal;
- ***Deactivated*** – REV received the weights and deactivated the signal.

By utilizing this signal, REV only interrupts the real-time analysis when new weights are available, instead of permanently accessing the connection and waiting for the weights to be sent. These interactions require synchronized access because both processes simultaneously accessing and changing the signal could lead to memory inconsistency errors.

The key aspect of this application is the continuous improvement of the model, which benefits from all user feedback. Therefore, in the eventuality that an end user validates a flow before it is revised, it is still obtained from Master Data and utilized in the experience replay sessions of REP. The sessions follow an episode-based methodology, where each session performs a preconfigured number of episodes, each training the agent with a preconfigured number of minibatches of validated flows.

Figures 35 and 36 display the process views of REV and REP, respectively. The utilized connection and signal are represented by classes to simplify their representation and abstract the synchronized access. Additionally, the *sleep()* method represents a preconfigured number of seconds of inactivity, which prevents sending a flood of requests to Master Data's API.

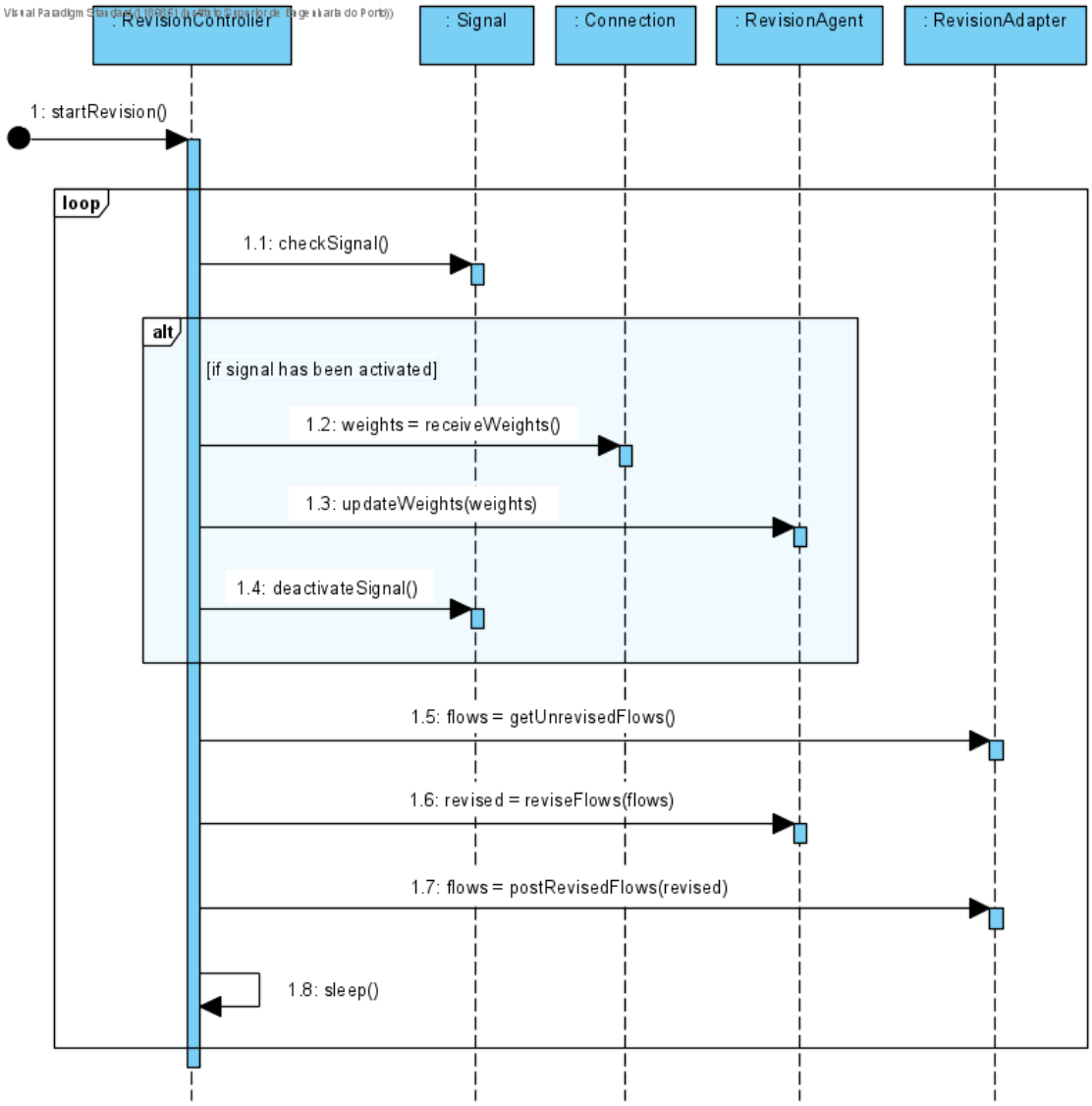

Figure 35. Process view of Revision process in Learning Agent (Level 3).

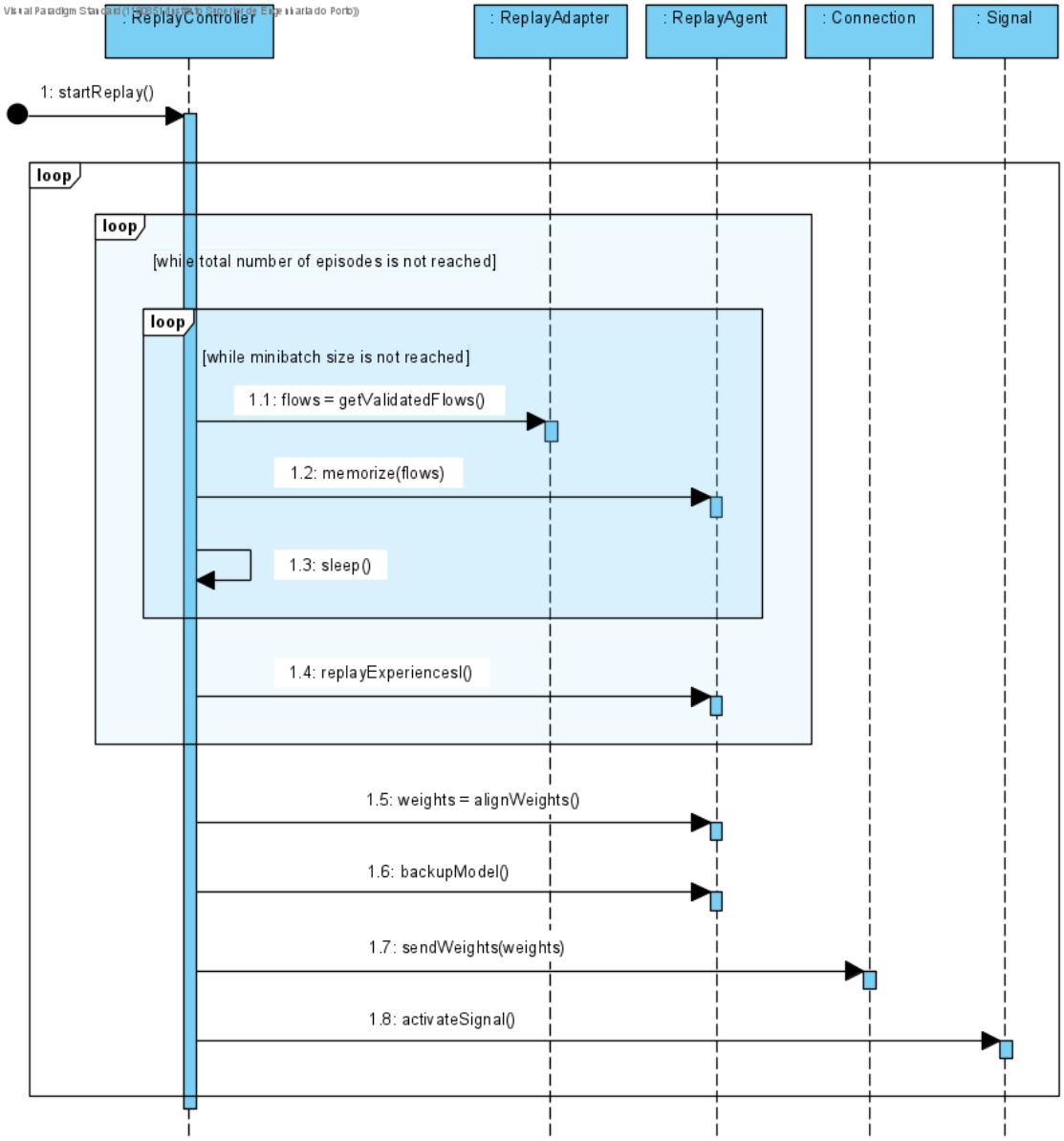

Figure 36. Process view of Experience Replay process in Learning Agent (Level 3).

4 System Implementation

This chapter addresses the implementation of the developed NIDS, including an assessment of the overall system.

4.1 Overview

The developed NIDS was implemented following a bottom-up approach, starting from specific functionalities and building up to complete use cases. Overall, the utilized technologies were selected due to the benefits described in Chapter 0.

For a proof-of-concept, the three software packages were deployed as separate Heroku applications and the data is stored in a MongoDB Atlas cluster. It is important to note that the cluster consists of two separate databases, one for the *User* aggregates and the other for the remaining domain aggregates, with the purpose of fully isolating user credentials. Figure 37 displays the physical view updated with the adopted topology.

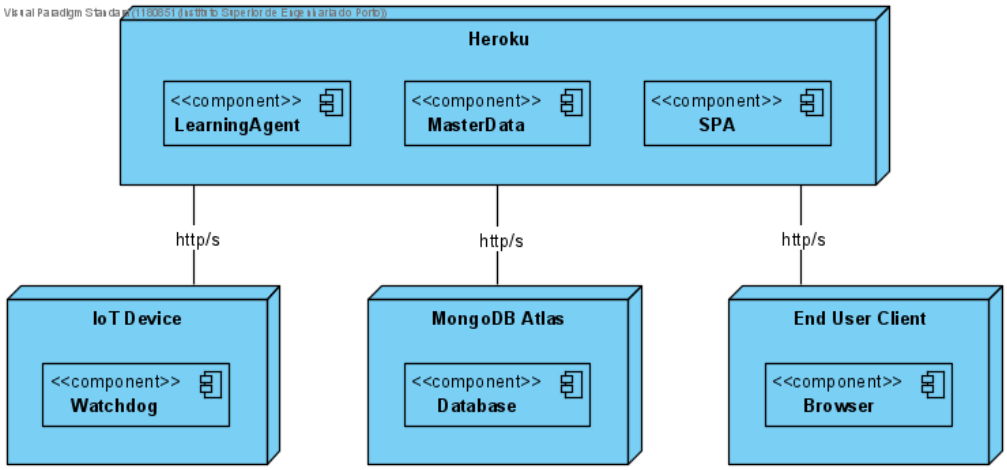

Figure 37. Physical view with adopted topology (Level 2).

4.2 Applications

Throughout the implementation, GitHub [98] was utilized for version control of the software packages. A Continuous Integration and Continuous Delivery (CI/CD) pipeline was utilized to automate the deployment of the applications, following the interconnected steps displayed in Figure 38. Additionally, scripts were created to simplify the setup, building, testing and execution of the final applications.

To enforce security in the pipeline, a static code analysis is performed before each release. Sensitive configurations, such as the utilized secrets, are provided through encrypted environmental variables to avoid exposing them in the source code.

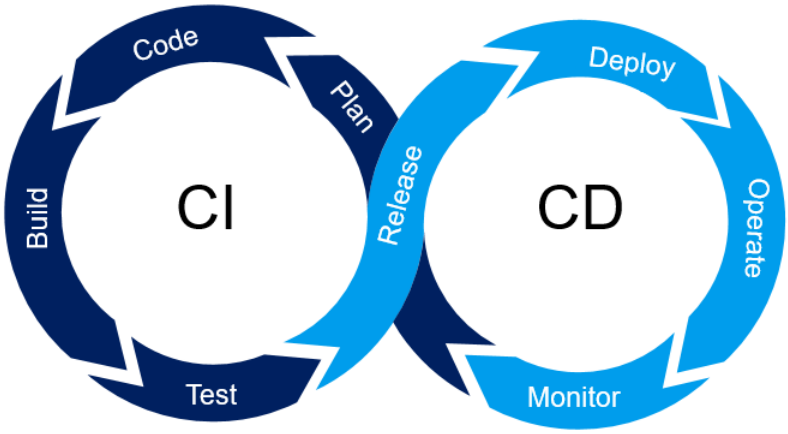

Figure 38. Continuous Integration/Continuous Delivery pipeline [99].

The implementation of each application will be described in its respective sub-section. The performed tests will be jointly described in the next section.

4.2.1 Master Data

The Master Data application relies on the Node.js technology and the Express.js framework for the operation of the back-end web server. Several well-established middlewares are utilized to secure the server:

- **Helmet** [100] – Enables the use of security-related HTTP headers;
- **CookieParser** [101] – Enables the use of signed session cookies;
- **CORS** [102] – Enables the use of the Cross-Origin Resource Sharing (CORS) mechanism, only allowing user access through SPA and with signed cookies.

It is important to note that with the utilized cookies have the *Secure* and *HttpOnly* attributes. Therefore, a cookie is only sent when the received request is HTTPS and it is inaccessible by client-side code, preventing an unsecure sign in and mitigating the risk of a Man-in-the-Middle attack.

Additionally, two “last resort” middlewares were created. The first handles any uncaught errors, preventing Master Data from exposing its internal operations. The second handles API requests for unknown resources, ensuring that a 404 HTTP code is sent.

Dependency injection is performed using the TypeDI [103] library. The dependencies of a class are provided in its creation, using a constructor-based approach to standardize the injection for the entire application.

Regarding the *Persistence*, the Mongoose [104] library is utilized as a wrapper for the communication with MongoDB Atlas. Therefore, the *Data Model* consists of several schemas that structure the domain entities and respective restrictions into serialized documents. Since the database cluster generates a unique code for each stored document, that code is used as the *FlowCode* of Benign and Malicious Flows.

The Bcryptjs [105] library is utilized to hash every new password with 12 salt rounds and the SHA-256 [106] algorithm, as well as to compare received passwords with the stored hash and salt. Even though the end user will not notice the extra milliseconds, this library aims at being slow to produce a better hash and therefore mitigate brute-force attacks.

The employed Bearer scheme utilizes tokens in the JavaScript Object Notation Web Token (JWT) format. These contain an "issued at" timestamp and specific information according to the actor. JWTs are encoded and decoded using the Jwt-simple [107] library. It utilizes the HS256 [108] algorithm to create secure tokens.

To access the environmental variables, the Env-cmd [109] library is used at the start of the application. In addition to the configuration variables, the username and password of an administrator account were preconfigured. Therefore, when the application is started, the configurations are loaded and a default administrator account is created to allow an initial access to the system. Then, new administrator accounts can be allowed and this one removed, through UC11 and UC12.

4.2.2 Single-Page Application

The SPA relies on the Node.js technology and Material-UI for the operation of the front-end React web application. Since the base technology is the same as Master Data, the selection of most libraries is replicated. Therefore, dependencies are injected using TypeDI, JWTs are encoded using Jwt-simple and environmental variables are accessed by Env-cmd.

Similarly, SPA also enforces the use of HTTPS to secure the communication between end users and Master Data. Regarding the *Adapters*, the Axios [110] library is utilized to act as an HTTP client. It wraps the communication with Master Data's API, exchanging JavaScript Object Notation (JSON) content.

End users access the system through the provided interface, which is referred to as Dashboard. The interface design complied with the restrictions of each use case and the sequences of interactions of the established scenarios (see Appendix B).

4.2.3 Learning Agent

The Learning Agent application relies on the Python programming language and several of its libraries. The Multiprocessing [111] library is employed for the interactions between the parallel processes. Their connection is achieved through a unidirectional Pipe and the utilized signal is a Value, which inherently enforces synchronized access.

Dependency injection and access to environmental variables are both performed by the Dependency-injector [112] library. It loads the configuration and provides the dependencies when a class is created. Regarding the communication with Master Data's API, the *Adapters* utilize the Requests [113] library, exchanging JSON content.

The implementation of the *Agent* component and the corresponding DRL methodology will be detailed in Chapter 0.

4.3 Tests

The implemented system was thoroughly tested to ensure it fulfils the requirements and performs as expected.

Unit tests were performed for the specific functionalities of each component. These tests were particularly important to ensure the restrictions of the domain aggregates and value objects. To assess the behaviour of multiple combined components, integration tests were performed on both Master Data and SPA, utilizing the Jest [114] and Sinon [115] frameworks. Regarding the React components of SPA, Enzyme [116] was also used.

Even though the three applications add up to a larger system, each one acts as a fully functional system. Due to the importance of Master Data's API, system tests were performed utilizing the Postman [117] platform to communicate with it and assess the behaviour of all use cases. Figure 39 displays a simplified view of the conducted system tests, where a black dot marks the requests sent by the allowed "Watchdog1" and a grey dot marks the requests sent by the Learning Agent application.

Finally, end-to-end tests were performed on the full NIDS to ensure it performs as expected in complete scenarios. The tests were conducted in a similar order to the one displayed in Figure 39, but utilizing the interface provided by SPA. Both manual and automated tests were performed, the latter using the Cypress [118] framework. However, since the storing of network flows by a watchdog requires manually providing the generated token, the automated tests utilized pre-stored flows.

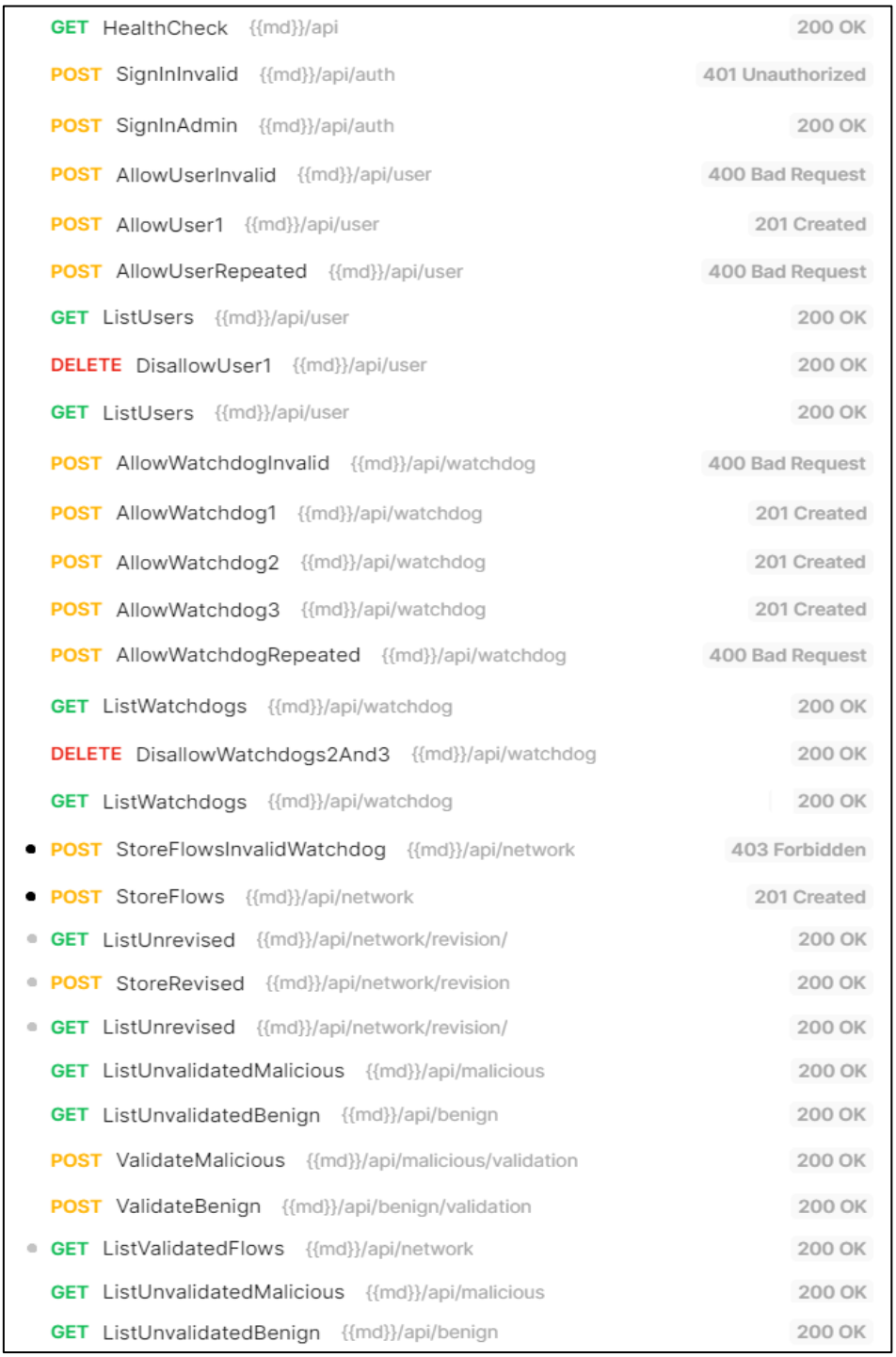

Figure 39. Simplified Master Data system tests.

4.4 Assessment

The final system is a scalable and reliable solution that fulfils the established functional and non-functional requirements. The three modular applications follow software design best-practices and have several straightforward configurations that can be adjusted to each specific deployment.

Due to the employed access control, actors can interact with the system exclusively through the scenarios corresponding to their roles. In particular, end users can utilize a simple, intuitive and consistent interface that provides a quick access to their respective scenarios, as well as to the Privacy Policy.

Regarding the benign flow revision, a real-time analysis is performed. It is only stopped during a brief and sporadic replacement for an improved model, to keep the revision up-to-date with the network topology.

5 Model Implementation

This chapter addresses the implementation of the ML and DRL models, including an assessment of the obtained results.

5.1 Overview

Since the focus of this project is the analysis of IoT network activity, several ML models, namely SVM, LightGBM, XGBoost, iForest and LOF, were implemented to evaluate their applicability. Regarding the conception of a DRL training process adapted to the intrusion detection context, it was based on the approach of a DDQN. These models were selected due to their promising results in the research surveyed in Chapter 0.

The models were trained, validated and tested in a machine with 16GB of RAM, an 8-core CPU and a 6GB GPU. The implementation relied on the following Python libraries:

- **Numpy** [119] – General data manipulation;
- **Pandas** [120] – Data preprocessing transformations;
- **Scikit-learn** [121] – Implementation of SVM, iForest and LOF;
- **Xgboost** [122]– Implementation of XGBoost;
- **Lightgbm** [123]– Implementation of LightGBM;
- **Tensorflow** [124]– Implementation of the DRL methodology;
- **Gym** [125] – Template for the DRL environment;
- **Matplotlib** [126] – Graphical visualization of the results.

5.2 Dataset

The utilized data was obtained from the IoT-23 dataset, which was created by the Stratosphere Research Laboratory as part of the Aposemat project. It consists of 23 labelled captures of malicious and benign network traffic, caused by different malwares targeting IoT devices, between 2018 and 2019. This is an extremely valuable dataset because it manifests real IoT network behaviour and provides a large quantity of labelled malicious flows. The structure of the recorded data contains the following fields:

- ***ts*** – Time of the connection;
- ***uid*** – Unique identifier for the connection;
- ***id.orig_h***, ***id.resp_h*** – IP Address of the original/response messages;
- ***id.orig_p***, ***id.resp_p*** – Ports used by the original/response messages;

- ***proto, service*** – Communication and service protocols;
- ***duration*** – Duration of the communication;
- ***orig_bytes***, ***resp_bytes*** – Number of bytes in the original/response messages;
- ***missed_bytes*** – Number of bytes missed in the communication;
- ***conn_state*** – State of the connection;
- ***history*** – History of the state of the connection;
- ***local_orig***, ***local_resp*** – Origin location of the original/response messages;
- ***orig_pkts***, ***resp_pkts*** – Number of packets in the original/response messages;
- ***orig_ip_bytes***, ***resp_ip_bytes*** – Number of IP bytes in the original/response messages;
- ***tunnel_parents*** – Unique identifier for the parent tunnels of the connection;
- ***label*** – Main label of the flow (i.e., malicious or benign);
- ***detailed-label*** – Additional label of the flow (i.e., specific malicious class).

From the 23 network captures, GECAD selected 6 due to their distinct characteristics. For instance, Capture-44-1 had an abnormally large number of packets comprising a small number of flows. Table 14 summarizes the characteristics of the selected captures.

Table 14. Overview of selected network captures.

Network Capture	Malware	Duration (hours)	Total Packets	Total Flows
Capture-1-1	Hide and Seek	112	1,686,000	1,008,749
Capture-20-1	Torii	24	50,000	3,210
Capture-21-1	Torii	24	50,000	3,287
Capture-34-1	Mirai	24	233,000	23,146
Capture-42-1	Trojan	8	24,000	4,427
Capture-44-1	Mirai	2	1,309,000	238

From these captures, several datasets were established. Since Capture-1-1 displayed a large number of recorded flows and the best balance between malicious and benign labels, three smaller balanced subsets were created. Table 15 provides an overview of the size and class proportions of the utilized datasets. The malicious class label PartOfAHorizontalPortScan was shortened to POAHPS.

Table 15. Overview of utilized datasets.

Dataset	Total Samples (#)	Malicious Samples (# per Class)	Malicious Samples (% per Class)	Malicious Class Label
1-1-full	1,008,749	539,465	53.47%	POAHPS
		8	< 0.01%	C&C
1-1-large	400,000	199,996	49.99%	POAHPS
		4	< 0.01%	C&C
1-1-medium	200,000	99,999	49.99%	POAHPS
		1	< 0.01%	C&C
1-1-small	20,000	10,000	50.0%	POAHPS
20-1	3,210	16	0.50%	C&C-Torii
21-1	3,287	14	0.43%	C&C-Torii
34-1	23,146	14,394	62.19%	DDoS
		6,706	28.97%	C&C
		122	0.53%	POAHPS
42-1	4,427	3	0.07%	FileDownload
		3	0.07%	C&C-FileDownload
44-1	238	14	5.91%	C&C
		11	4.64%	C&C-FileDownload
		1	0.42%	DDoS

5.3 Data Preprocessing

Before utilizing the datasets, a preprocessing stage was required. The transformations performed on the data were divided into seven steps, described below.

First. The features were separated from the labels and the latter were converted to an integer representation. For binary classification, only the *label* field was considered and the general benign and malicious labels were assigned to 0 and 1. For multi-class classification, the *label* and *detailed-label* fields were merged, keeping benign labels as 0 and assigning each malicious class to a positive integer, instead of the general malicious label.

Second. The features that did not provide any valuable information, namely *ts* and *uid*, were discarded. Since *local_orig*, *local_resp* and *tunnel_parents* contained only null values, they were also discarded in every dataset.

Third. The numerical features were checked for outliers and these values were removed, being replaced by a null value. It is important to note that outliers were determined only on samples with a benign label, since malicious flows are intended to have anomalous values. Both Tukey's method, based on the interquartile range, and Kernel Density Estimation method, based on bimodal distributions, were considered. The latter could not be performed on larger datasets, such as 1-1-full, so Tukey's method was selected for standardization.

Fourth. The null values of numerical features, either originally missing or created by the previous step, were imputed with the median of their respective field. This approach ensures that all samples have valid data and effectively avoids the issues associated with discarding samples, such as the introduction of biases. Following the approach of the previous step, the medians were calculated using only samples with benign labels.

Fifth. The categorical features, which correspond to the *id.orig_h*, *id.resp_p*, *id.orig_h*, *id.resp_p*, *proto*, *service*, *conn_state* and *history* fields, were adapted to numeric values through one-hot encoding. Due to the high cardinality of these fields, the very low frequency categories were replaced with a single category named "Other", to avoid encoding categories with almost no samples and therefore small relevance. Additionally, that category also included the null values of the field, as a way of expressing that those samples did not belong to the remaining categories.

Sixth. The holdout technique was applied by randomly splitting the dataset into training and test sets with 70% and 30% of the samples, respectively. To ensure that both sets have the same class proportions as the original dataset, the split was performed with stratification.

Seventh. The data was normalized to values in the range [0,1] using Min-Max scaling. The minimum and maximum values were obtained from the training set only, to avoid contaminating the training data with information from the test data.

An additional aspect is that if a class only contains a single sample, it cannot be simultaneously used for training and testing. Therefore, these individual samples must be discarded. This is the case of the 1-1-medium and 44-1 datasets, when utilized for multi-class classification. Regarding 1-1-medium, since there is only one malicious class remaining, in practice it is only suitable for binary classification. Table 16 displays the applicability of each dataset to binary and multi-class classification, as well as the specific malicious class labels.

Table 16. Classification applicability of utilized datasets.

Dataset	Binary Classification	Multi-class Classification	Malicious Class Label
1-1-full	✓	✓	POAHPS
			C&C
1-1-large	✓	✓	POAHPS
			C&C
1-1-medium	✓	✗	POAHPS
1-1-small	✓	✗	POAHPS
20-1	✓	✗	C&C-Torii
21-1	✓	✗	C&C-Torii
34-1	✓	✓	DDoS
			C&C
			POAHPS
42-1	✓	✓	FileDownload
			C&C-FileDownload
44-1	✓	✓	C&C
			C&C-FileDownload

5.4 Models

The configurations of the implemented ML models result from a grid search of possible hyperparameter combinations for both binary and multi-class classification.

To obtain the optimal configurations, 5-fold cross-validation was employed on the training sets. Therefore, each iteration was trained with 4/5 of the training set and validated with the remaining 1/5. The validation was performed using the macro-averaged F1-Score due to its adequacy for unbalanced datasets and consolidation of Precision and Recall.

Even though the unsupervised models, namely iForest and LOF, perform one-class classification with unlabelled data, they can be compared with the binary classifiers. Therefore, cross-validation was still performed to assess their predictions when trained with only a subset of the samples.

Regarding the DRL methodology, the training process was devised through a manual optimization of several aspects. Due to the characteristics inherent to the adopted approach, cross-validation was not utilized.

After their optimization, the models were retrained with the full training sets and a final evaluation was performed with the test sets. The implementation of each model will be described in its respective sub-section. The obtained cross-validation and test results will be presented and discussed in the next section.

5.4.1 Support Vector Machine

SVM [127] is a supervised classifier that attempts to find a hyperplane that successfully segregates two classes in an n-dimensional space, where n is the number of features. Even though SVMs only inherently performs binary classification, the implementation available in the Scikit-learn library handles multi-class classification as a One-vs-Rest scheme by default [128]. This scheme, also referred to as One-vs-All, creates k binary classifiers for k total classes, where each one determines if a sample belongs to a certain class or not.

This classifier can utilize several kernels. In particular, the model performed better with the Linear kernel, which evidences the linear separability of the data. The two classes are expressed as 1 and -1, with 0 being the boundary. This kernel utilizes the Squared Hinge loss function [129], which is very sensitive to outliers. The Squared Hinge loss of a single sample can be calculated according to Equation (6).

$$l(y, p) = max(0, (1 - y \cdot p))^2 \quad (6)$$

where y is the class of the sample and p is the predicted distance to the boundary.

The default L2 penalty and 0.0001 tolerance for stopping criteria were utilized. Additionally, since the number of samples is much higher than the number of features, the dual parameter was set to false to solve the primal optimization problem. Table 17 summarizes the configuration common to all utilized datasets.

Table 17. Summary of common SVM configuration.

Parameter	**Value**
Kernel	Linear
Loss Function	Squared Hinge
Penalty	L2
Tolerance	0.0001
Dual	False

Besides the common configuration, the regularization needed to be adjusted to each individual dataset. The strength of the regularization is inversely proportional to the C parameter. The optimized values for this parameter are displayed in Tables 18 and 19.

Table 18. Summary of dataset-specific binary SVM configuration.

Parameter	1-1-full	1-1-large	1-1-medium	1-1-small	20-1	21-1	34-1	42-1	44-1
C	0.001	0.001	0.001	0.001	0.01	0.1	0.1	0.1	0.1

Table 19. Summary of dataset-specific multi-class SVM configuration.

Parameter	1-1-full	1-1-large	34-1	42-1	44-1
C	0.001	0.001	0.01	0.01	0.1

5.4.2 Extreme Gradient Boosting

XGBoost [130] is a supervised classifier that performs gradient boosting using an ensemble of decision trees. A level-wise growth strategy is employed to split nodes level by level, seeking to minimize the value calculated by a loss function.

By default, this classifier utilizes the acknowledged Cross-Entropy loss function [131], which corresponds to the *binary:logistic* and *multi:softprob* objectives for binary and multi-class classification, respectively. The Cross-Entropy loss of a single sample can be calculated according to Equation (7).

$$l(y, p) = -\sum_{i=1}^{C} (y_i \cdot \log(p_i)) \tag{7}$$

where C is the total number of classes, p_i is the predicted probability that the sample belongs to class i and y_i is 1 if it does or 0 otherwise.

XGBoost was optimized in conformity with its official documentation [132]. The configuration common to all utilized datasets is described below and summarized in Table 20.

To prevent overfitting to the training data, the maximum depth of the trees was set to 5 levels and the minimum loss reduction required to split a node was set to 0.01. These parameters limit tree growth and ensure that splits with negligible improvement are disregarded, which also increases the training speed.

Both L1 and L2 regularization can be applied on node weights, which indirectly affect the regularization of the features. L2 has a default value of 1.0 to slightly influence the weights to be small, improving the generalization of the model. On the other hand, L1 is disabled by default because the ensemble of decision trees already selects the optimal splits and therefore the relevant features.

Another important aspect is subsampling, which XGBoost can perform on the actual samples and/or the utilized features, with specific ratios for subsampling features by tree, by level and by node. Regarding the actual samples, the ratio was kept at 1.0 because subsampling significantly decreased the performance of the classifier. Regarding the utilized features, the classifier performed better with 0.7 ratios, using only 70% at each step.

Besides the common configuration, several aspects needed to be optimized to each individual dataset, being similar for both binary and multi-class classification. The corresponding parameters are described below and summarized in Tables 21 and 22.

Table 20. Summary of common XGBoost configuration.

Parameter	Value
Loss Function (via Objective)	Cross-Entropy
Max Depth	5
Min Loss Reduction (Gamma)	0.01
L2 Regularization (Lambda)	1.0
Subsample	1.0
Feature Subsample By Tree	0.7
Feature Subsample By Level	0.7
Feature Subsample By Node	0.7

Three main methods can be utilized to build the decision trees, namely Exact, Approximated and Histogram. The first is a greedy method that considers all candidates for node splitting. Even though it is not suitable for large amounts of data, it provides the best splits for the smaller datasets. The second and third methods calculate approximations, being both suitable for the larger datasets. The main advantage of the third is its faster histogram-based approximations.

The key parameters that need to be adjusted to each dataset are the number of estimators and the learning rate. The first represents the number of decision trees and therefore boosting iterations. The latter controls how quickly the classifier adapts its weights to the training data. These parameters were optimized to provide the best balance between underfitting and overfitting. Overall, the learning rate was set to a small value and the number of estimators to a relatively large value, to avoid a fast convergence to a suboptimal solution.

In addition to the minimum loss reduction required for node splitting, the minimum child weight can also be specified. If a child node created by a split has a total Hessian weight below this parameter, it cannot be further split. Establishing a minimum weight indirectly affects the minimum samples required to create a node, which prevents the creation of overly specific and therefore overfit nodes. The value of this parameter was increased with the size of the dataset. Exceptionally, despite being the smallest dataset, 44-1 performed better with a value of 3.0.

Table 21. Summary of dataset-specific binary XGBoost configuration.

Parameter	1-1-full	1-1-large	1-1-medium	1-1-small	20-1	21-1	34-1	42-1	44-1
Method	Histogram	Histogram	Histogram	Histogram	Exact	Exact	Histogram	Exact	Exact
Nº of Estimators	80	70	60	60	80	80	70	60	60
Learning Rate	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.01	0.01
Min Child Weight	100.0	80.0	60.0	40.0	2.0	2.0	40.0	1.2	3.0

Table 22. Summary of dataset-specific multi-class XGBoost configuration.

Parameter	1-1-full	1-1-large	34-1	42-1	44-1
Method	Histogram	Histogram	Histogram	Exact	Exact
Nº of Estimators	80	70	60	60	60
Learning Rate	0.001	0.001	0.01	0.01	0.01
Min Child Weight	100.0	80.0	40.0	1.2	3.0

5.4.3 Light Gradient Boosting Machine

LightGBM [133] is a supervised classifier that also utilizes an ensemble of decision trees to perform gradient boosting. The main distinction between XGBoost and this classifier is that a leaf-wise growth strategy is employed for a best-first approach, resulting in an asymmetrical tree. This strategy directly splits the leaf with the maximum loss reduction, calculated through histogram-based approximations.

Even though LightGBM's configuration is similar to XGBoost's configuration, an equivalent parameter can have a different effect on its performance. Therefore, this classifier was optimized in conformity with its official documentation [134]. The configuration common to all utilized datasets is described below and summarized in Table 23.

The main similarity is the Cross-Entropy loss function, which this classifier utilizes by default for both *binary* and *multiclass* objectives. To improve the generalization of the model, the minimum loss reduction and L2 regularization parameters were set to 0.01 and 1.0, respectively.

Since a leaf-wise strategy is employed, the essential parameter to regulate the complexity of the trees is the maximum leaves, instead of the maximum depth. Nonetheless, to prevent overfitting, the official documentation recommends that both parameters are adjusted according to Equation (8).

$$Max\ Leaves < 2^{(Max\ Depth)} \tag{8}$$

Therefore, with the maximum depth set to 5, the maximum leaves were set to 25 to further limit tree growth.

The decision trees can be built by three main methods, namely traditional gradient boosting, Dropouts meet Multiple Additive Regression Trees and Gradient-based One-Side Sampling (GOSS) [133]. The main advantage of GOSS is its computationally lighter and therefore faster convergence.

Regarding subsampling, LightGBM can perform it on the actual samples and/or the features utilized by the trees. Since GOSS inherently performs subsampling, no additional ratio was specified for the actual samples. Regarding the utilized features, the classifier performed better with a 0.7 ratio, using only 70% per boosting iteration.

Table 23. Summary of common LightGBM configuration.

Parameter	Value
Method (Boosting Type)	GOSS
Loss Function (via Objective)	Cross-Entropy
Max Depth	5
Max Leaves	25
Min Loss Reduction (Min Split Gain)	0.01
L2 Regularization (Lambda)	1.0
Feature Subsample By Tree	0.7

Besides the common configuration, several aspects needed to be optimized to each individual dataset, being similar for both binary and multi-class classification. The parameters are described below and their values are summarized in Tables 24 and 25.

The key parameters are the number of estimators and the learning rate, equivalent to XGBoost's parameters. Overall, learning rate was set to a small value and the number of estimators to a relatively large value. Nonetheless, the smaller datasets required the learning rate to be increased to counteract the shortage of training data.

It is important to note that unbalanced datasets required a significant increase of these parameters. This can be observed in the binary configuration of 34-1. Even though it contains an equivalent number of samples to 1-1-small, the learning rate had to be 10 times higher for an extra 20 iterations to be able to account for the minority classes.

Instead of specifying the minimum Hessian weight for child nodes, the minimum child samples can be directly specified. However, high values can more easily cause underfitting. For instance, if the minimum number of samples in a node is 100 and only 99 samples exhibit a certain feature, the model will not be able to benefit from the characteristics of that feature. This parameter was established at approximately 0.2% of the total number of samples of a dataset, since higher values would significantly decrease the performance of this classifier.

Table 24. Summary of dataset-specific binary LightGBM configuration.

Parameter	1-1-full	1-1-large	1-1-medium	1-1-small	20-1	21-1	34-1	42-1	44-1
Nº of Estimators	80	70	60	60	100	100	100	100	80
Learning Rate	0.001	0.001	0.001	0.001	0.02	0.02	0.01	0.04	0.02
Min Child Samples	2000	800	400	40	5	5	40	5	2

Table 25. Summary of dataset-specific multi-class LightGBM configuration.

Parameter	1-1-full	1-1-large	34-1	42-1	44-1
Nº of Estimators	80	70	100	80	80
Learning Rate	0.001	0.001	0.01	0.02	0.02
Min Child Samples	2000	800	40	5	2

5.4.4 Isolation Forest

The iForest [135] model performs an unsupervised isolation of anomalies through an ensemble of decision trees. The samples are repeatedly split by a random value of a random feature, until outliers are isolated. The configuration of this model is described below and the common and dataset-specific parameters are summarized in Tables 26 and 27, respectively.

The number of estimators controls the number of trees utilized in the ensemble, which is equivalent to the parameter utilized by both XGBoost and LightGBM. In contrast with those models, the optimal value of this parameter remained identical for all datasets.

An important aspect is subsampling, which enables the model to isolate samples from a smaller and clearer set. The size of the subsamples utilized to train the decision trees is specified by the maximum samples parameter, which was optimized to each individual dataset. Additionally, the utilized features can also be subsampled, but the model performed significantly better with a ratio of 1.0, using all the available features.

The isolation of anomalies relies on the contamination ratio of the training data. This ratio must not exceed 50%, otherwise the outliers cannot be detected. Therefore, the number of samples intended to be anomalies must not exceed the number of remaining samples. Overall, the contamination ratio was set to the approximate percentage of malicious flows in each dataset. However, some datasets had to be adapted due to the excessive malicious flows.

Even though the 34-1 and 1-1-full datasets do not fit this requirement, 1-1-large, 1-1-medium and 1-1-small are theoretically suitable because of their 50/50 proportion of benign and malicious flows. However, in practice, the model underperformed with such high contamination in these three datasets. To overcome this obstacle and be able to utilize all the five datasets, the malicious flows of their training sets were randomly subsampled. The model performed better with a 0.05 contamination ratio. This value corresponds to 0.3% and 4% of the malicious flows of the training sets of 34-1 and 1-1-full, respectively. Similarly, it corresponds to 5% for 1-1-large, 1-1-medium and 1-1-small.

Table 26. Summary of common iForest configuration.

Parameter	Value
Nº of Estimators	100
Max Features	1.0

Table 27. Summary of dataset-specific iForest configuration.

Parameter	1-1-full	1-1-large	1-1-medium	1-1-small	20-1	21-1	34-1	42-1	44-1
Contamination	0.05	0.05	0.05	0.05	0.006	0.007	0.05	0.001	0.15
Max Samples	250	250	250	250	250	250	100	100	100

5.4.5 Local Outlier Factor

LOF [136] is an unsupervised model that detects anomalies by measuring the local density deviation. This strategy identifies samples with a significantly lower density than their neighbours, which correspond to local outliers that would otherwise go undetected. The configuration of this model is described below and the common and dataset-specific parameters are summarized in Tables 28 and 29, respectively.

Three algorithms can be utilized to compute the nearest neighbours, namely brute-force, Ball Tree and K-Dimensional Tree. Due to its high computational cost, the first algorithm was excluded. From the remaining two, the model performed better with the K-Dimensional Tree algorithm, the Euclidean metric and a leaf size of 30.

By default, LOF only computes outliers on the initial data it receives. The novelty parameter was set to “True” to enable the model to detect outliers on new data, based on the neighbourhoods computed in its training.

The key parameter of this model is the number of neighbours. This parameter regulates the size of the neighbourhoods, affecting the measurement of the local density deviation. Overall, larger datasets performed better with a higher number of neighbours, with 341 requiring the particularly high value of 520.

Since this model also relies on the contamination ratio of the training data, which must be lower than 50%, the approach employed for iForest was replicated.

Table 28. Summary of common LOF configuration.

Parameter	Value
Algorithm	K-Dimensional Tree
Metric	Euclidean
Leaf Size	30
Novelty	True

Table 29. Summary of dataset-specific LOF configuration.

Parameter	1-1-full	1-1-large	1-1-medium	1-1-small	20-1	21-1	34-1	42-1	44-1
Contamination	0.05	0.05	0.05	0.05	0.006	0.007	0.05	0.001	0.15
Nº of Neighbours	290	290	290	290	35	45	520	60	50

5.4.6 Deep Reinforcement Learning Model

To apply the DRL methodology to the intrusion detection context, it was necessary to create a suitable environment and an adapted training process.

Regarding the environment, the standardized template of the Gym library [125] was followed. First, the environment resets to a random state, which corresponds to a random sample of the dataset. Then, every time the agent observes a state and performs an action, predicting a class, a reward and the next state are provided. Due to the promising results achieved in [30], a simple 1/0 reward was utilized for correct/incorrect actions.

Regarding the training process, it was implemented using an agent and a controller. These apply the experience replay, target network, and exploration concepts, due to the benefits described in Chapter 0. This approach was based on the training of a DDQN. However, in a DDQN, the training is affected by the expected future rewards, calculated via the next state provided by the environment. Since the correctness of future predictions is not relevant to the classification of a network flow, the next state is disregarded.

It is important to note that the utilized agent and controller are specific to the initial training of a model, whereas the *Agents* and *Controllers* of the Learning Agent application are components with responsibilities specific to the NIDS. Therefore, even though the experience replay and target network concepts are also applied to the improvement of the model, the components do not perform any exploration.

Figure 40 displays a diagram with the adapted training process. It was created using the Business Process Model and Notation (BPMN) representation to depict the interactions between the controller, agent and environment. The considered aspects and utilized configurations are detailed below.

The controller applies an incremental episode-based methodology. Several training episodes are performed, each consisting of multiple steps in which the agent interacts with the environment. When the number of performed steps reaches the specified minibatch size, the agent is signalled to perform experience replay. This is repeated until the specified number of steps per episode is reached. Then, the agent is signalled to synchronize the target network, completing the episode.

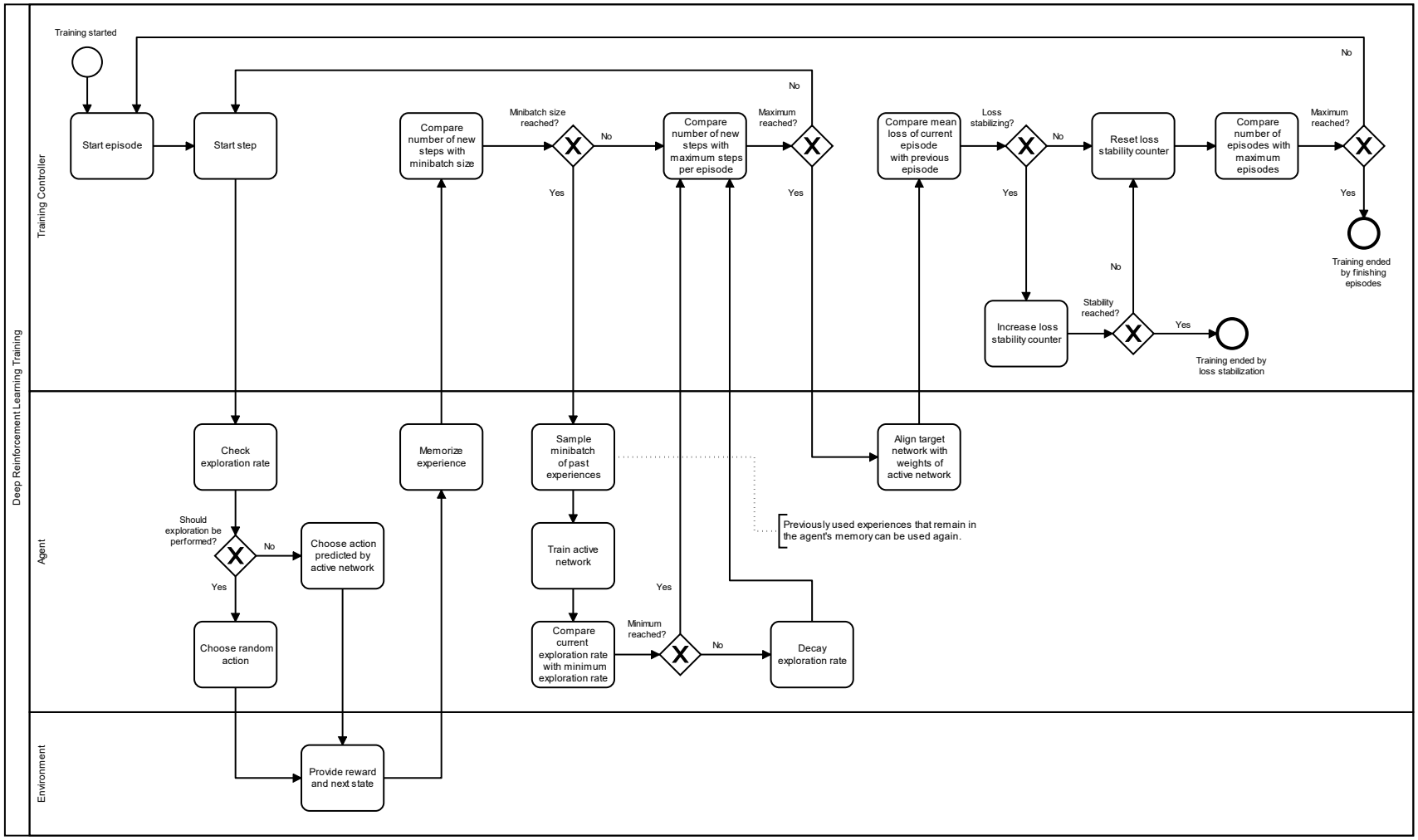

Figure 40. Adapted Deep Reinforcement Learning training process (BPMN diagram).

After each episode, the mean loss of the performed experience replays is compared with the previous ones. This is an early stopping approach that finishes the initial training if the loss has stabilized, to avoid overfitting. The training is considered to be stabilized if, for a certain number of episodes, the variance of their values is within the same range.

Several configurations were considered for each dataset. Since an episode aggregates several losses, the number of episodes and the stability range were set to 3 and 0.05, which gives margin for a slight variance. Utilizing 5% of a training set as the steps per episode achieved the best balance between underfitting and overfitting. Consequently, a total of 20 episodes can be performed. Considering the small size of most of the utilized datasets, the minibatch size was established as half the number of steps, enabling two experience replays per episode. Due to the larger size of the 1-1-full, 1-1-large and 1-1-medium datasets, the percentage of steps per episode was decreased.

The minibatch size affects the experience replay of the agent, since that is the number of randomly sampled past experiences. The memory size was set to 1.5x the minibatch size to enable the agent keep half of the last minibatch in memory. Therefore, the agent can use up to half of the previous samples to mitigate the risk of catastrophic interference. Additionally, since the target network concept is utilized, the impact of the weight updates of the active network is reduced.

Regarding the agent's interactions with the environment, a random exploration is performed. The exploration rate was set at 0.2, which corresponds to 20% random actions and 80% predictions by the active network. With each experience replay, the rate is decayed by 0.01 until the minimum value of 0.05 is reached. This is performed to decrease the random actions as the active network adapts its weights to the training data.

Tables 30 and 31 summarize the common controller and agent parameters, respectively. The dataset-specific parameters are displayed in Table 34.

Table 30. Summary of common DRL controller configuration.

Parameter	Value
Nº of Episodes	20
Stability Range	0.05
Min Stable Episodes	3

Table 31. Summary of common DRL agent configuration.

Parameter	Value
Memory Size	3.75%
Exploration Rate	0.2
Exploration Rate Decay	0.01
Min Exploration Rate	0.05
Epochs	20

The active and target networks employ a four-layered ANN. The input layer node size is the number of utilized features. Next, there are two hidden layers with 20 neurons each and the computationally efficient Rectified Linear Unit (ReLU) [137] activation function. For an input x, it can be calculated according to Equation (9).

$$f(x) = max(0, x) \tag{9}$$

When binary classification is being performed, the output layer is created with 1 output node and the Sigmoid [138] activation function. For an input x, it can be calculated according to Equation (10).

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{10}$$

When multi-class classification is being performed, the output layer node size is the total number of classes of the dataset. It is created with the Softmax [139] activation function. For a class i of the input vector $\vec{x}$, it can be calculated by Equation (11).

$$\sigma(\vec{x})_i = \frac{e^{x_i}}{\sum_{j=1}^{C}(e^{x_j})} \tag{11}$$

where C is the total number of classes, x_i is the input of class i and x_j is the output of class j.

Table 32 describes the structure of the ANN. To account for the differences between the datasets, F and C represent the number of utilized features and the total number of classes, respectively.

Table 32. Employed ANN structure.

Layer		Size	Activation
Dense		F	-
Dense		20	ReLU
Dense		20	ReLU
Dense	Binary output	1	Sigmoid
	Multi-class output	C	Softmax

In addition to establishing a structure, several parameters are required. The Adam optimization algorithm [140] was used with a learning rate of 0.001. Several rates were tested for each dataset, but a better performance was always achieved with this value. The Cross-Entropy loss function was utilized, which corresponds to the *binary_crossentropy* and *categorical_crossentropy* values for binary and multi-class classification, respectively. Table 33 summarizes the configuration.

Table 33. Summary of ANN configuration.

Parameter	Value
Loss Function	Cross-Entropy
Optimizer	Adam
Learning Rate	0.001

Table 34. Summary of dataset-specific controller and agent configuration.

Parameter	1-1-full	1-1-large	1-1-medium	1-1-small	20-1	21-1	34-1	42-1	44-1
Nº of Steps per Episode (controller)	2100	2100	1400	700	112	115	810	155	16
Minibatch Size (controller)	1400	1400	700	350	56	57	405	77	8
Memory Size (agent)	2800	2800	1050	525	84	85	607	115	12

5.5 Results and Discussion

The results obtained for binary and multi-class classification will be presented and discussed in their respective sub-sections.

5.5.1 Binary Classification

From a binary perspective, a comparison of the mean F1-Score of the optimized model configurations is displayed in Figure 41. Figure 42 compares the F1-Scores obtained in the final model tests, which include the DRL model. The values are discussed below.

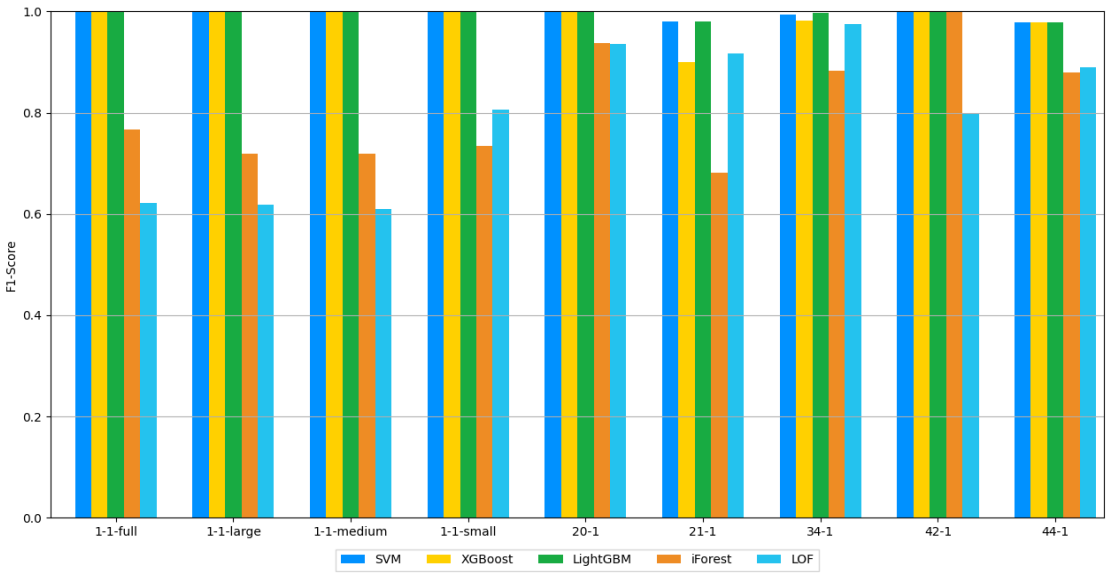

Figure 41. Mean binary cross-validation F1-Score.

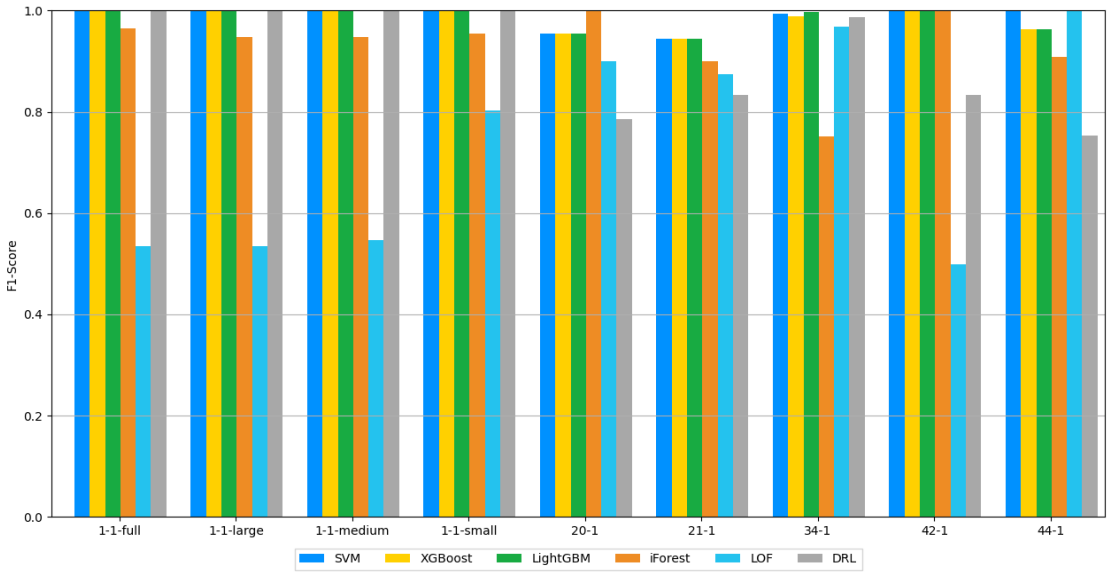

Figure 42. Final binary test F1-Score.

Regarding the performed cross-validation, the supervised models, namely SVM, XGBoost and LightGBM, achieved near perfect F1-Scores when training with a large quantity of balanced data. Nonetheless, their performance worsened on some of the unbalanced and smaller datasets, such as 44-1.

All three models obtained very similar results, except on two datasets. On 34-1, which is highly unbalanced towards malicious flows, the best score was achieved by LightGBM. It reached a value of 99.73%, which is slightly better than the 99.30% and 98.14% of SVM and XGBoost, respectively. However, on 21-1, despite LightGBM and SVM both obtaining approximately 97.99%, XGBoost only reached 89.98%, possibly due to overfitting the small quantity of training data.

The unsupervised models, namely iForest and LOF, obtained quite different validation scores, only reaching similar values on 20-1 and 44-1. In contrast with the supervised models, better performances were achieved on the smaller datasets. Nonetheless, these were significantly lower on most datasets.

LOF obtained better results than iForest on 1-1-small, 21-1 and 34-1. In particular, on 21-1, it even surpassed XGBoost with a score of 91.66%. However, iForest outperformed LOF on the remaining datasets and obtained a near perfect score on 42-1, reaching the results of the supervised models.

Regarding the final model tests, the supervised models achieved a good generalization, preserving the almost perfect scores on most datasets. However, on 20-1 and 21-1, the scores of all three models decreased, which indicates the occurrence of overfitting. The main distinction can be noted on the smallest dataset, 44-1. The scores of both XGBoost and LightGBM were decreased, but SVM was able to preserve a near perfect score.

The test results of iForest significantly increased on the larger datasets, almost reaching the supervised models. Additionally, it also reached a near perfect score on 20-1. This indicates its suitability to detect anomalies on unseen data. On the other hand, LOF worsened on all datasets except 44-1. On this last dataset, it obtained a near perfect score, possibly due to the small number of samples.

Overall, the results indicate that the implemented supervised models are reliable on most datasets but decrease in performance on smaller training sets, whereas the unsupervised models are more advantageous for small datasets when these are highly unbalanced towards benign flows.

Regarding the DRL methodology, near perfect test scores were achieved on the larger datasets, either balanced or unbalanced, which is the case of 34-1. However, it can be noted that the smaller the dataset, the lower the obtained score. This leads to the conclusion that the devised training process requires a large and balanced training set to be effective.

5.5.2 Multi-class Classification

From a multi-class perspective, a comparison of the mean macro-averaged F1-Score of the optimized model configurations is displayed in Figure 43. Figure 44 compares the macro-averaged F1-Scores obtained in the final model tests, which include the DRL model. The values are discussed below.

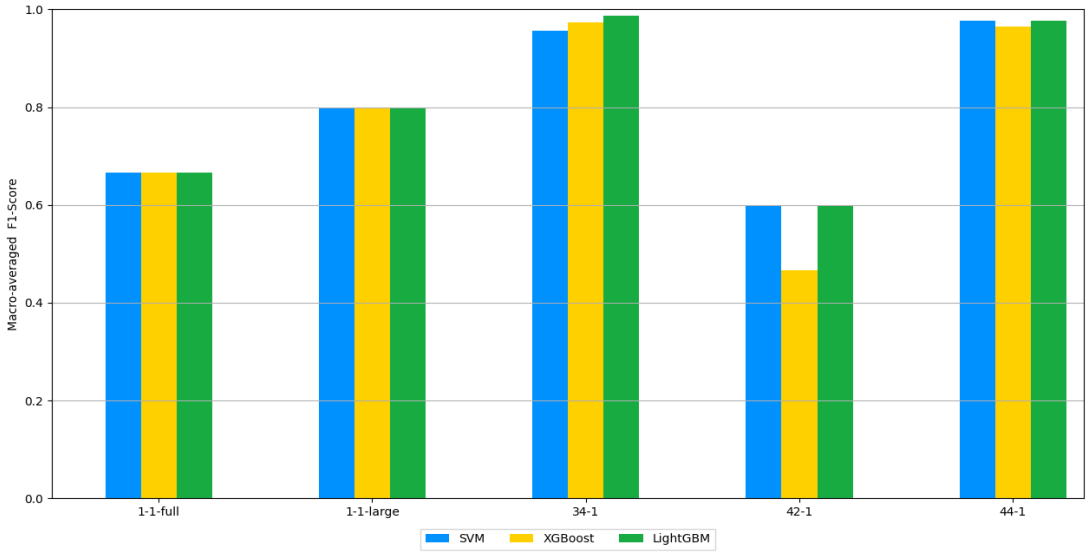

Figure 43. Mean multi-class cross-validation macro-averaged F1-Score.

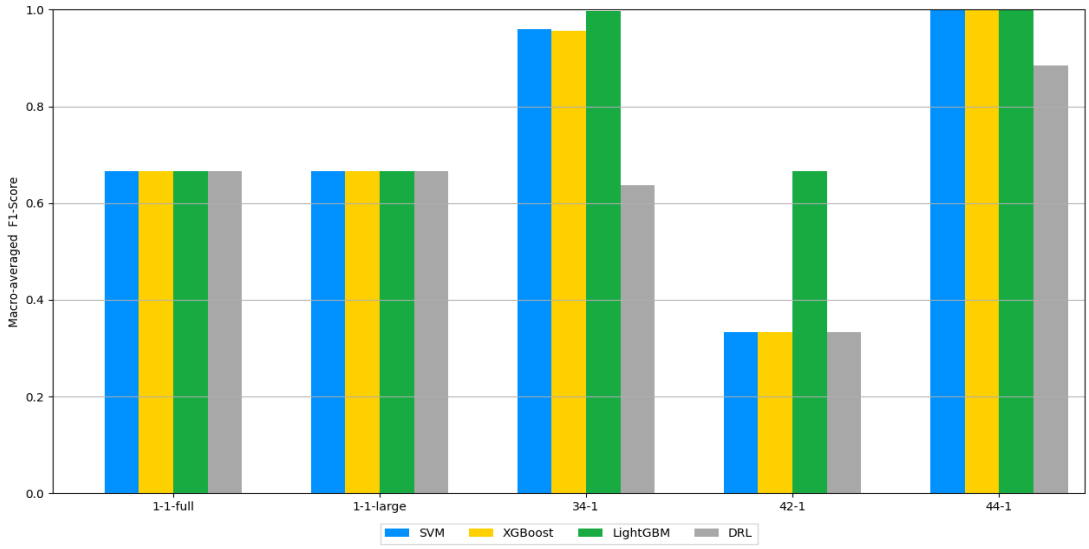

Figure 44. Final multi-class test macro-averaged F1-Score.

Regarding the performed cross-validation, the supervised models achieved very high scores on the 34-1 and 44-1 datasets. On the first, LightGBM achieved a score of 98.77%, followed by XGBoost with 97.30% and SVM with 95.67%. On the latter, both LightGBM and SVM reached approximately 97.66%, whereas XGBoost obtained 96.44%.

On the other hand, very poor results were obtained on the particularly unbalanced datasets. Since 1-1-full, 1-1-large and 42-1 contain minority classes with a very low proportion of samples, the models were not able to learn how to correctly classify them.

Regarding the final model tests, the supervised models reached near perfect scores on the 44-1 dataset. Similarly, on 34-1, the score of LightGBM increased. However, XGBoost obtained a slightly lower score than SVM. This indicates a good generalization of LightGBM, as well as SVM, but a slight overfitting of XGBoost.

All three models reached scores near 66% on 1-1-full and 1-1-large, due to the neglect of the minority class. Furthermore, on 42-1, the scores of SVM and XGBoost decreased to near 33%, failing to detect both minority classes of this dataset. Since the models performed poorly on both validation and test data, it could be a sign of underfitting due to the lack of training samples. In contrast, the score of LightGBM increased to approximately 66%, achieving an apparently good generalization of one of the minority classes.

Overall, the models achieved a relatively good multi-class classification performance on the utilized datasets. The main drawbacks remain the lack of training data and the unbalanced class proportions.

Regarding the DRL methodology, the obtained results were similar to the supervised models on 1-1-full, 1-1-large and 42-1. However, only 63.75% and 88.38% were reached on 34-1 and 44-1, respectively. This indicates that the training process cannot successfully account for an unbalanced dataset with multiple minority classes.

6 Conclusions

This chapter addresses the accomplished objectives and the constrains of the solution, as well as future work that can be developed to improve it.

6.1 Accomplished Objectives

The initially established objectives and the results accomplished by the developed work are summarized in Table 35.

Table 35. Overview of accomplished objectives.

Objective	Result	Completion Rate
Development of a NIDS	A system with three modular applications was developed, implemented and tested	100%
Integration of an adapted DRL methodology	An adapted DRL methodology was successfully integrated in the Learning Agent application	100%
Implementation of supervised models	SVM, LightGBM and XGBoost were trained, validated and tested	100%
Implementation of unsupervised models	iForest and LOF were trained, validated and tested	100%
Conception of an adapted DRL model	The training process of a DDQN was successfully adapted to train a model for intrusion detection	100%
Evaluation and comparison of all models	The performances of all models were evaluated and compared, assessing their applicability to IoT intrusion detection	100%

6.2 Constraints and Future Work

The developed NIDS has one key constraint. To be able to correctly revise benign flows, their features and respective values must be comparable. Therefore, both the system and the watchdogs must utilize the same feature layout and their values must follow the same normalization range. This restricts the deployment to one of the following options:

- All watchdogs store non-normalized flows and the Learning Agent is configured to analyse them directly, without normalization being performed;
- All watchdogs store non-normalized flows and the Learning Agent is configured to perform normalization before the analysis;
- All watchdogs normalize flows utilizing the same normalization range and the Learning Agent is configured to analyse them directly.

In the future, the system could be adapted to create a distinction between the normalization ranges of different watchdogs, enabling an automatic conversion of the values to the range utilized by the Learning Agent.

Another aspect that could be developed is the ability to rectify flow validations. For instance, a new use case could enable administrators to revert an already validated flow to a previous status. Even though it would not undo the training performed with the incorrect reward, the new validation could still aid the improvement of the detection.

Future research could lead to the implementation of several other models and training methodologies to evaluate their performance, compare them with the already developed work and assess their applicability to intrusion detection in IoT systems.

6.3 Final Remarks

The work developed in this project contributed to the research and development work of GECAD. Several promising ML models were evaluated, the DRL methodology was adapted to intrusion detection and a continuously improving NIDS was developed.

Overall, this was a very interesting and challenging project. It provided me the opportunity to acquire a vast amount of knowledge and technical skills related to the ML field, while improving my problem-solving and organizational skills. Furthermore, I was able to put my prior knowledge of cybersecurity and software design into practice to develop an innovative solution for IoT intrusion detection.

References

[1] "GECAD Website." http://www.gecad.isep.ipp.pt/ (accessed Mar. 29, 2021).

[2] "SeCoIIA Project Website." https://secoiia.eu/ (accessed Mar. 29, 2021).

[3] K. Ashton, "That 'Internet of Things' Thing," *RFID J.*, 2009, Accessed: Apr. 03, 2021. Available: https://www.rfidjournal.com/that-internet-of-things-thing.

[4] I. Butun, P. Osterberg, and H. Song, "Security of the Internet of Things: Vulnerabilities, Attacks, and Countermeasures," *IEEE Commun. Surv. Tutorials*, vol. 22, no. 1, pp. 616–644, 2020, doi: 10.1109/COMST.2019.2953364.

[5] E. Sisinni, A. Saifullah, S. Han, U. Jennehag, and M. Gidlund, "Industrial internet of things: Challenges, opportunities, and directions," *IEEE Trans. Ind. Informatics*, vol. 14, no. 11, pp. 4724–4734, 2018, doi: 10.1109/TII.2018.2852491.

[6] N. Chaabouni, M. Mosbah, A. Zemmari, C. Sauvignac, and P. Faruki, "Network Intrusion Detection for IoT Security Based on Learning Techniques," *IEEE Commun. Surv. Tutorials*, vol. 21, no. 3, pp. 2671–2701, 2019, doi: 10.1109/COMST.2019.2896380.

[7] T. Dingsoeyr, D. Falessi, and K. Power, "Agile Development at Scale: The Next Frontier," *IEEE Softw.*, vol. 36, no. 2, pp. 30–38, 2019, doi: 10.1109/MS.2018.2884884.

[8] L. Parody, "How to Manage Modern Software Projects," *Medium*, 2018, Accessed: Apr. 16, 2021. Available: https://medium.com/@lizparody/waterfall-vs-agile-methodology-in-software-development-1e19ef168cf6.

[9] W. Vorhies, "CRISP-DM – a Standard Methodology to Ensure a Good Outcome," *Data Sci. Cent.*, 2016, Accessed: Apr. 16, 2021. Available: https://www.datasciencecentral.com/profiles/blogs/crisp-dm-a-standard-methodology-to-ensure-a-good-outcome.

[10] X. Liu, C. Qian, W. G. Hatcher, H. Xu, W. Liao, and W. Yu, "Secure Internet of Things (IoT)-Based Smart-World Critical Infrastructures: Survey, Case Study and Research Opportunities," *IEEE Access*, vol. 7, pp. 79523–79544, 2019, doi: 10.1109/ACCESS.2019.2920763.

[11] F. Meneghello, M. Calore, D. Zucchetto, M. Polese, and A. Zanella, "IoT: Internet of Threats? A Survey of Practical Security Vulnerabilities in Real IoT Devices," *IEEE Internet Things J.*, vol. 6, no. 5, pp. 8182–8201, 2019, doi: 10.1109/JIOT.2019.2935189.

[12] P. Anand, Y. Singh, A. Selwal, P. K. Singh, R. A. Felseghi, and M. S. Raboaca, “IoVT: Internet of vulnerable things? threat architecture, attack surfaces, and vulnerabilities in internet of things and its applications towards smart grids,” *Energies*, vol. 13, no. 18, pp. 1–23, 2020, doi: 10.3390/en13184813.

[13] S. Pal, M. Hitchens, T. Rabehaja, and S. Mukhopadhyay, “Security requirements for the internet of things: A systematic approach,” *Sensors (Switzerland)*, vol. 20, no. 20, pp. 1–34, 2020, doi: 10.3390/s20205897.

[14] N. Neshenko, E. Bou-Harb, J. Crichigno, G. Kaddoum, and N. Ghani, “Demystifying IoT Security: An Exhaustive Survey on IoT Vulnerabilities and a First Empirical Look on Internet-Scale IoT Exploitations,” *IEEE Commun. Surv. Tutorials*, vol. 21, no. 3, pp. 2702–2733, 2019, doi: 10.1109/COMST.2019.2910750.

[15] K. Tange, M. De Donno, X. Fafoutis, and N. Dragoni, “A Systematic Survey of Industrial Internet of Things Security: Requirements and Fog Computing Opportunities,” *IEEE Commun. Surv. Tutorials*, vol. 22, no. 4, pp. 2489–2520, 2020, doi: 10.1109/COMST.2020.3011208.

[16] R. Colelli, S. Panzieri, and F. Pascucci, “Securing connection between IT and OT: The Fog Intrusion Detection System prospective,” *2019 IEEE Int. Work. Metrol. Ind. 4.0 IoT, MetroInd 4.0 IoT 2019 - Proc.*, pp. 444–448, 2019, doi: 10.1109/METROI4.2019.8792884.

[17] E. Benkhelifa, T. Welsh, and W. Hamouda, “A critical review of practices and challenges in intrusion detection systems for IoT: Toward universal and resilient systems,” *IEEE Commun. Surv. Tutorials*, vol. 20, no. 4, pp. 3496–3509, 2018, doi: 10.1109/COMST.2018.2844742.

[18] A. Khraisat, I. Gondal, P. Vamplew, and J. Kamruzzaman, “Survey of intrusion detection systems: techniques, datasets and challenges,” *Cybersecurity*, vol. 2, no. 1, 2019, doi: 10.1186/s42400-019-0038-7.

[19] J. Sengupta, S. Ruj, and S. Das Bit, “A Comprehensive Survey on Attacks, Security Issues and Blockchain Solutions for IoT and IIoT,” *J. Netw. Comput. Appl.*, vol. 149, no. October 2019, p. 102481, 2020, doi: 10.1016/j.jnca.2019.102481.

[20] F. Hussain, R. Hussain, S. A. Hassan, and E. Hossain, “Machine Learning in IoT Security: Current Solutions and Future Challenges,” *IEEE Commun. Surv. Tutorials*, vol. 22, no. 3, 2020, doi: 10.1109/COMST.2020.2986444.

[21] A. C. Panchal, V. M. Khadse, and P. N. Mahalle, “Security Issues in IIoT: A Comprehensive Survey of Attacks on IIoT and Its Countermeasures,” *Proc. - 2018 IEEE*

Glob. Conf. Wirel. Comput. Networking, GCWCN 2018, pp. 124–130, 2019, doi: 10.1109/GCWCN.2018.8668630.

[22] S. M. Tahsien, H. Karimipour, and P. Spachos, "Machine learning based solutions for security of Internet of Things (IoT): A survey," *J. Netw. Comput. Appl.*, vol. 161, no. April, 2020, doi: 10.1016/j.jnca.2020.102630.

[23] A. Srivastava, S. Gupta, M. Quamara, P. Chaudhary, and V. J. Aski, "Future IoT-enabled threats and vulnerabilities: State of the art, challenges, and future prospects," *Int. J. Commun. Syst.*, vol. 33, no. 12, 2020, doi: 10.1002/dac.4443.

[24] H. Zimmermann, "OSI Reference Model - The ISO Model of Architecture for Open Systems Interconnection," *IEEE Trans. Commun.*, vol. 28, no. 4, pp. 425–432, 1980, doi: 10.1109/TCOM.1980.1094702.

[25] B. Leiner, R. Cole, J. Postel, and D. Mills, "The DARPA internet protocol suite," *IEEE Commun. Mag.*, vol. 23, no. 3, pp. 29–34, 1985, doi: 10.1109/MCOM.1985.1092530.

[26] B. B. Zarpelão, R. S. Miani, C. T. Kawakani, and S. C. de Alvarenga, "A survey of intrusion detection in Internet of Things," *J. Netw. Comput. Appl.*, vol. 84, no. February, pp. 25–37, 2017, doi: 10.1016/j.jnca.2017.02.009.

[27] H. Liu and B. Lang, "Machine learning and deep learning methods for intrusion detection systems: A survey," *Appl. Sci.*, vol. 9, no. 20, 2019, doi: 10.3390/app9204396.

[28] BrainStation, "Machine Learning," *TowardsDataScience*, 2017, Accessed: Apr. 03, 2021. Available: https://towardsdatascience.com/machine-learning-101-supervised-unsupervised-reinforcement-beyond-f18e722069bc.

[29] S. Loukas, "What is Machine Learning," *TowardsDataScience*, Jun. 2020, Accessed: Apr. 03, 2021. Available: https://towardsdatascience.com/what-is-machine-learning-a-short-note-on-supervised-unsupervised-semi-supervised-and-aed1573ae9bb.

[30] M. Lopez-Martin, B. Carro, and A. Sanchez-Esguevillas, "Application of deep reinforcement learning to intrusion detection for supervised problems," *Expert Syst. Appl.*, vol. 141, p. 112963, 2020, doi: 10.1016/j.eswa.2019.112963.

[31] E. Tzorakoleftherakis, "Getting Started with Reinforcement Learning," *Digit. Eng. 247*, 2020, Accessed: Jun. 25, 2021. Available: https://www.digitalengineering247.com/article/getting-started-with-reinforcement-learning/design.

[32] X. Zhu and A. B. Goldberg, "Introduction to semi-supervised learning," *Synth. Lect. Artif. Intell. Mach. Learn.*, vol. 3, no. 1, pp. 1–130, 2009.

[33] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep

Reinforcement Learning: A Brief Survey," *IEEE Signal Process. Mag.*, vol. 34, no. 6, pp. 26–38, 2017, doi: 10.1109/MSP.2017.2743240.

[34] I. Kononenko and M. Kukar, "Machine Learning Basics," *Mach. Learn. Data Min.*, pp. 59–105, 2007, doi: 10.1533/9780857099440.59.

[35] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *J. Mach. Learn. Res.*, vol. 13, pp. 281–305, 2012.

[36] G. C. Cawley and N. L. C. Talbot, "On over-fitting in model selection and subsequent selection bias in performance evaluation," *J. Mach. Learn. Res.*, vol. 11, pp. 2079–2107, 2010.

[37] S. Arlot and A. Celisse, "A Survey of Cross Validation Procedures for Model Selection," *Stat. Surv.*, vol. 4, 2009, doi: 10.1214/09-SS054.

[38] "Cross-validation: evaluating estimator performance," *Scikit-learn*. https://scikit-learn.org/stable/modules/cross_validation.html (accessed May 28, 2021).

[39] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A detailed analysis of the KDD CUP 99 data set," *IEEE Symp. Comput. Intell. Secur. Def. Appl. CISDA 2009*, no. Cisda, pp. 1–6, 2009, doi: 10.1109/CISDA.2009.5356528.

[40] C. Kolias, G. Kambourakis, A. Stavrou, and S. Gritzalis, "Intrusion detection in 802.11 networks: Empirical evaluation of threats and a public dataset," *IEEE Commun. Surv. Tutorials*, vol. 18, no. 1, pp. 184–208, 2016, doi: 10.1109/COMST.2015.2402161.

[41] Y. Meidan *et al.*, "N-BaIoT-Network-based detection of IoT botnet attacks using deep autoencoders," *IEEE Pervasive Comput.*, vol. 17, no. 3, pp. 12–22, 2018, doi: 10.1109/MPRV.2018.03367731.

[42] N. Koroniotis, N. Moustafa, E. Sitnikova, and B. Turnbull, "Towards the development of realistic botnet dataset in the Internet of Things for network forensic analytics: Bot-IoT dataset," *Futur. Gener. Comput. Syst.*, vol. 100, pp. 779–796, 2019, doi: 10.1016/j.future.2019.05.041.

[43] H. Hindy, E. Bayne, M. Bures, R. Atkinson, C. Tachtatzis, and X. Bellekens, "Machine Learning Based IoT Intrusion Detection System: An MQTT Case Study (MQTT-IoT-IDS2020 Dataset)," *Lect. Notes Networks Syst.*, vol. 180, no. June, pp. 73–84, 2021, doi: 10.1007/978-3-030-64758-2_6.

[44] N. A. Stoian, "Machine Learning for anomaly detection in IoT networks : Malware analysis on the IoT-23 data set," 2020, Available: http://essay.utwente.nl/81979/.

[45] E. Chatzoglou, G. Kambourakis, and C. Kolias, "Empirical Evaluation of Attacks against IEEE 802.11 Enterprise Networks: The AWID3 Dataset," *IEEE Access*, vol. 9, pp.

34188–34205, 2021, doi: 10.1109/ACCESS.2021.3061609.

[46] J. Lever, M. Krzywinski, and N. Altman, “Classification evaluation,” *Nat. Methods*, vol. 13, no. 8, pp. 603–605, 2016.

[47] A. Verma and V. Ranga, “Machine Learning Based Intrusion Detection Systems for IoT Applications,” *Wirel. Pers. Commun.*, vol. 111, no. 4, pp. 2287–2310, 2020, doi: 10.1007/s11277-019-06986-8.

[48] A. Thakkar and R. Lohiya, *A Review on Machine Learning and Deep Learning Perspectives of IDS for IoT: Recent Updates, Security Issues, and Challenges*, no. 0123456789. Springer Netherlands, 2020.

[49] J. Meira *et al.*, “Performance evaluation of unsupervised techniques in cyber-attack anomaly detection,” *J. Ambient Intell. Humaniz. Comput.*, vol. 11, no. 11, pp. 4477–4489, 2020, doi: 10.1007/s12652-019-01417-9.

[50] Y. N. Soe, Y. Feng, P. I. Santosa, R. Hartanto, and K. Sakurai, “Machine learning-based IoT-botnet attack detection with sequential architecture,” *Sensors (Switzerland)*, vol. 20, no. 16, pp. 1–15, 2020, doi: 10.3390/s20164372.

[51] N. Oliveira, I. Praça, E. Maia, and O. Sousa, “Intelligent cyber attack detection and classification for network-based intrusion detection systems,” *Appl. Sci.*, vol. 11, no. 4, pp. 1–21, 2021, doi: 10.3390/app11041674.

[52] Y. Meidan *et al.*, “Detection of Unauthorized IoT Devices Using Machine Learning Techniques,” *arXiv*, no. September, 2017.

[53] F. A. Bakhtiar, E. S. Pramukantoro, and H. Nihri, “A lightweight IDS based on j48 algorithm for detecting DoS attacks on IoT middleware,” *2019 IEEE 1st Glob. Conf. Life Sci. Technol. LifeTech 2019*, pp. 41–42, 2019, doi: 10.1109/LifeTech.2019.8884057.

[54] H. Yao, P. Gao, P. Zhang, J. Wang, C. Jiang, and L. Lu, “Hybrid intrusion detection system for edge-based IIoT relying on machine-learning-aided detection,” *IEEE Netw.*, vol. 33, no. 5, pp. 75–81, 2019, doi: 10.1109/MNET.001.1800479.

[55] Y. Zhang, P. Li, and X. Wang, “Intrusion Detection for IoT Based on Improved Genetic Algorithm and Deep Belief Network,” *IEEE Access*, vol. 7, pp. 31711–31722, 2019, doi: 10.1109/ACCESS.2019.2903723.

[56] G. Caminero, M. Lopez-Martin, and B. Carro, “Adversarial environment reinforcement learning algorithm for intrusion detection,” *Comput. Networks*, vol. 159, pp. 96–109, 2019, doi: 10.1016/j.comnet.2019.05.013.

[57] M. Eskandari, Z. H. Janjua, M. Vecchio, and F. Antonelli, “Passban IDS: An Intelligent

Anomaly-Based Intrusion Detection System for IoT Edge Devices," *IEEE Internet Things J.*, vol. 7, no. 8, pp. 6882–6897, 2020, doi: 10.1109/JIOT.2020.2970501.

[58] T. Gu, A. Abhishek, H. Fu, H. Zhang, D. Basu, and P. Mohapatra, "Towards Learning-automation IoT Attack Detection through Reinforcement Learning," *Proc. - 21st IEEE Int. Symp. a World Wireless, Mob. Multimed. Networks, WoWMoM 2020*, pp. 88–97, 2020, doi: 10.1109/WoWMoM49955.2020.00029.

[59] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-Learning," *30th AAAI Conf. Artif. Intell. AAAI 2016*, pp. 2094–2100, 2016.

[60] L. Weng, "Exploration Strategies in Deep Reinforcement Learning," *Lil'Log*, Jun. 2020, Accessed: May 17, 2021. Available: https://lilianweng.github.io/lil-log/2020/06/07/exploration-strategies-in-deep-reinforcement-learning.html.

[61] J. Kirkpatrick *et al.*, "Overcoming catastrophic forgetting in neural networks.," *Proc. Natl. Acad. Sci. U. S. A.*, vol. 114, no. 13, pp. 3521–3526, Mar. 2017, doi: 10.1073/pnas.1611835114.

[62] J. Torres, "Deep Q-Network (DQN)-II. Experience Replay and Target Networks," *TowardsDataScience*, Aug. 2020, Accessed: Apr. 18, 2021. Available: https://towardsdatascience.com/deep-q-network-dqn-ii-b6bf911b6b2c.

[63] E. Evans and E. J. Evans, *Domain-driven Design: Tackling Complexity in the Heart of Software*. Addison-Wesley Professional, 2004.

[64] R. Martin, "Design principles and design patterns," *Object Mentor*, no. c, pp. 1–34, 2000, Available: https://labs.cs.upt.ro/labs/ip2/html/lectures/2/res/Martin-PrinciplesAndPatterns.PDF.

[65] C. Larman, *Applying UML and Patterns: An Introduction to Object-oriented Analysis and Design and the Unified Process*. Prentice Hall Professional, 2002.

[66] R. T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," University of California, 2000.

[67] "Python Website." https://www.python.org/ (accessed May 07, 2021).

[68] "R Website." https://www.r-project.org/ (accessed May 07, 2021).

[69] T. IBM, "Comparison of Python and R," *IBM*, Mar. 2021, Accessed: May 07, 2021. Available: https://www.ibm.com/cloud/blog/python-vs-r.

[70] M. Berga and P. Coelho, "The Data Science language debate," *ImaginaryCloud*, 2021, Accessed: May 07, 2021. Available: https://www.imaginarycloud.com/blog/r-vs-python/.

[71] M. Iansiti and K. R. Lakhani, "The Truth About Blockchain," *Harv. Bus. Rev.*, Feb. 2017,

Available: https://hbr.org/2017/01/the-truth-about-blockchain.

[72] W. Meng, E. W. Tischhauser, Q. Wang, Y. Wang, and J. Han, “When intrusion detection meets blockchain technology: A review,” *IEEE Access*, vol. 6, pp. 10179–10188, 2018, doi: 10.1109/ACCESS.2018.2799854.

[73] V. Acharya, A. E. Yerrapati, and N. Prakash, *Oracle Blockchain Services Quick Start Guide*. Packt, 2019.

[74] “MongoDB Atlas Website.” https://www.mongodb.com/cloud/atlas (accessed May 14, 2021).

[75] M. Heller, “Review: MongoDB takes on the world,” *InfoWorld*, Sep. 2018, Available: https://www.infoworld.com/article/3300619/review-mongodb-takes-on-the-world.html.

[76] “Replication - MongoDB documentation.” https://docs.mongodb.com/manual/replication/ (accessed May 19, 2021).

[77] “Azure Cosmos DB Website.” https://azure.microsoft.com/en-us/services/cosmos-db/ (accessed May 14, 2021).

[78] M. Chudinov, “Comparison of Azure Cosmos DB and MongoDB Atlas,” *Capgemini*, Jun. 2021, Accessed: Jun. 05, 2021. Available: https://www.capgemini.com/se-en/2021/06/azure-cosmos-db-vs-mongodb-atlas/.

[79] “Node.js Website.” https://nodejs.org/en/ (accessed May 09, 2021).

[80]. “NET Website.” https://dotnet.microsoft.com/ (accessed May 09, 2021).

[81] C. Gor, “Comparison of Node.js and ASP.NET for Web Development,” *Educba*, 2019, Accessed: May 09, 2021. Available: https://www.esparkinfo.com/node-js-vs-asp-net.html.

[82] M. Tech, “Node.js vs .NET in 2020,” *Medium*, 2020, Accessed: May 09, 2021. Available: https://medium.com/techmagic/node-js-vs-net-what-to-choose-in-2020-3b53ddfebd28.

[83] “Express.js Framework Website.” https://expressjs.com/ (accessed May 10, 2021).

[84] Mozilla, “Express/Node Introduction,” *MDN Web Docs*, Apr. 2021, Accessed: May 10, 2021. Available: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction.

[85] S. CM, “Benefits of Using Express.js for Backend Development,” *Echomoro*, Nov. 2020, Accessed: May 10, 2021. Available: https://www.techomoro.com/what-are-the-benefits-of-using-express-js-for-backend-development/.

[86] “React Website.” https://reactjs.org/ (accessed May 11, 2021).

[87] “Angular Website.” https://angular.io/ (accessed May 11, 2021).

[88] O. Romanyuk, “Angular vs React: Which One to Choose,” *FreeCodeCamp*, Oct. 2019, Accessed: May 11, 2021. Available: https://www.freecodecamp.org/news/angular-vs-react-what-to-choose-for-your-app-2/.

[89] J. Reis, “Comparison of Angular and React,” *ImaginaryCloud*, Feb. 2020, Accessed: May 11, 2021. Available: https://www.imaginarycloud.com/blog/angular-vs-react/.

[90] “Material-UI Website.” https://material-ui.com/ (accessed May 11, 2021).

[91] C. Realm, “Meet the Material-UI library,” *FreeCodeCamp*, Apr. 2018, Accessed: May 11, 2021. Available: https://www.freecodecamp.org/news/meet-your-material-ui-your-new-favorite-user-interface-library-6349a1c88a8c/.

[92] “Heroku Website.” https://www.heroku.com/ (accessed May 13, 2021).

[93] K. Rusev, “What is Heroku and What is it Used For,” *MentorMate*, Dec. 2020, Accessed: May 13, 2021. Available: https://mentormate.com/blog/what-is-heroku-used-for-cloud-development/.

[94] “General Data Protection Regulation,” *EUR-Lex*, Accessed: Apr. 19, 2021. Available: http://data.europa.eu/eli/reg/2016/679/2016-05-04.

[95] S. Brown, “The C4 model for visualising software architecture,” Accessed: Jun. 10, 2021. Available: https://c4model.com/.

[96] P. B. Kruchten, “The 4+1 View Model of architecture,” *IEEE Softw.*, vol. 12, no. 6, pp. 42–50, 1995, doi: 10.1109/52.469759.

[97] D. Exterman, “Multithreading vs. Multiprocessing - Choosing the Right Approach,” *Incredibuild*, 2020, Accessed: May 02, 2021. Available: https://www.incredibuild.com/blog/multithreading-vs-multiprocessing-choosing-the-right-approach-for-your-development.

[98] “GitHub Website.” https://github.com/ (accessed Jun. 10, 2021).

[99] Rohde and Schwarz, “CI/CD Pipeline and Security,” *DevSecOps Community*, 2021, Accessed: Jun. 02, 2021. Available: https://dev-appsec.rohde-schwarz.com/news/400606.

[100] “Helmet - npm.” https://www.npmjs.com/package/helmet (accessed May 24, 2021).

[101] “Cookie-parser - npm.” https://www.npmjs.com/package/cookie-parser (accessed May 24, 2021).

[102] “CORS - npm.” https://www.npmjs.com/package/cors (accessed May 24, 2021).

[103] “Typedi - npm.” https://www.npmjs.com/package/typedi (accessed May 24, 2021).

[104] “Mongoose - npm.” https://www.npmjs.com/package/mongoose (accessed May 24,

2021).

[105] “Bcryptjs - npm.” https://www.npmjs.com/package/bcryptjs (accessed May 24, 2021).

[106] “SHA-256 Encryption,” *N-able*, Accessed: May 24, 2021. Available: https://www.n-able.com/blog/sha-256-encryption.

[107] “Jwt-simple - npm.” https://www.npmjs.com/package/jwt-simple (accessed May 24, 2021).

[108] “Security of JSON Web Tokens,” *CyberPolygon*, Accessed: May 24, 2021. Available: https://cyberpolygon.com/materials/security-of-json-web-tokens-jwt/.

[109] “Env-cmd - npm.” https://www.npmjs.com/package/env-cmd (accessed May 24, 2021).

[110] “Axios - npm.” https://www.npmjs.com/package/axios (accessed May 29, 2021).

[111] “Multiprocessing Library documentation.” https://docs.python.org/3/library/multiprocessing.html (accessed May 30, 2021).

[112] “Dependency Injector Library documentation.” https://python-dependency-injector.ets-labs.org/ (accessed May 30, 2021).

[113] “Requests Library documentation.” https://docs.python-requests.org/en/master/ (accessed May 30, 2021).

[114] “Jest - npm.” https://www.npmjs.com/package/jest (accessed May 29, 2021).

[115] “Sinon - npm.” https://www.npmjs.com/package/sinon (accessed May 29, 2021).

[116] “Enzyme - npm.” https://www.npmjs.com/package/enzyme (accessed May 29, 2021).

[117] “Postman Platform Website.” https://www.postman.com/ (accessed May 29, 2021).

[118] “Cypress Website.” https://www.cypress.io/ (accessed May 29, 2021).

[119] “NumPy documentation.” https://numpy.org/ (accessed Apr. 23, 2021).

[120] “Pandas documentation.” https://pandas.pydata.org (accessed Apr. 23, 2021).

[121] “Scikit-learn documentation.” https://scikit-learn.org/stable/ (accessed Apr. 23, 2021).

[122] “XGBClassifier Python API documentation.” https://xgboost.readthedocs.io/en/latest/python/python_api.html#xgboost.XGBClassifier (accessed Apr. 24, 2021).

[123] “LGBMClassifier Python API documentation.” https://lightgbm.readthedocs.io/en/latest/pythonapi/lightgbm.LGBMClassifier.html (accessed Apr. 24, 2021).

[124] “TensorFlow documentation.” https://www.tensorflow.org (accessed Apr. 23, 2021).

[125] G. Brockman *et al.*, “OpenAI Gym.” 2016, Available: https://gym.openai.com/.

[126] “Matplotlib documentation.” https://matplotlib.org (accessed Apr. 23, 2021).

[127] M. A. Hearst, S. T. Dumais, E. Osuna, J. Platt, and B. Scholkopf, “Support vector machines,” *IEEE Intell. Syst. their Appl.*, vol. 13, no. 4, pp. 18–28, 1998, doi: 10.1109/5254.708428.

[128] “LinearSVC documentation.” https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html (accessed May 01, 2021).

[129] “Squared Hinge function.” https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/loss-functions (accessed May 01, 2021).

[130] T. Chen and C. Guestrin, “XGBoost: A scalable tree boosting system,” *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Min.*, vol. 13-17-Augu, pp. 785–794, 2016, doi: 10.1145/2939672.2939785.

[131] K. Koech, “Cross-Entropy Loss Function definition,” *TowardsDataScience*, Oct. 2020, Accessed: May 01, 2021. Available: https://towardsdatascience.com/cross-entropy-loss-function-f38c4ec8643e.

[132] “XGBoost documentation.” https://xgboost.readthedocs.io/en/latest/ (accessed Apr. 23, 2021).

[133] G. Ke *et al.*, “LightGBM: A highly efficient gradient boosting decision tree,” *Adv. Neural Inf. Process. Syst.*, vol. 2017-Decem, no. Nips, pp. 3147–3155, 2017.

[134] “LightGBM documentation.” https://lightgbm.readthedocs.io/en/latest/ (accessed Apr. 23, 2021).

[135] F. Tony Liu, K. Ming Ting, and Z.-H. Zhou, “Isolation Forest ICDM08,” *Icdm*, 2008. Available: https://cs.nju.edu.cn/zhouzh/zhouzh.files/publication/icdm08b.pdf.

[136] M. Breunig, H.-P. Kriegel, R. Ng, and J. Sander, “LOF: Identifying Density-Based Local Outliers.,” in *ACM Sigmod Record*, 2000, vol. 29, pp. 93–104, doi: 10.1145/342009.335388.

[137] A. F. Agarap, “Deep Learning using Rectified Linear Units (ReLU),” no. 1, pp. 2–8, 2018, Available: http://arxiv.org/abs/1803.08375.

[138] T. Wood, “Softmax Function Definition,” *DeepAI*, Accessed: Apr. 26, 2021. Available: https://deepai.org/machine-learning-glossary-and-terms/softmax-layer.

[139] T. Wood, “Sigmoid Function Definition,” *DeepAI*, Accessed: Apr. 26, 2021. Available: https://deepai.org/machine-learning-glossary-and-terms/sigmoid-function.

[140] D. P. Kingma and J. L. Ba, “Adam: A method for stochastic optimization,” *3rd Int. Conf. Learn. Represent. ICLR 2015 - Conf. Track Proc.*, pp. 1–15, 2015.

Appendix A Privacy Policy

2021-05-01

We are committed to comply with the General Data Protection Regulation (GDPR) to ensure security and privacy for all the information you provide to the Network-based Intrusion Detection System (NIDS). This Privacy Policy defines the practices applied in the collection and processing of your data. We request that you read it thoroughly to better understand your rights when using this system.

A. Responsible entity and contacts

The Project Company (ProjCo), with VATIN/NIPC PT 123456789 and headquarters in Praça Afonso Henriques, 42, 4400-200, Porto, Portugal, is the entity responsible for the data processing. You can contact us through the email nids@projco.pt, the phone number +351 221 543 876 or by sending a letter addressed to our headquarters.

ProjCo designated a Data Protection Officer that can be reached through the email dpo@projco.pt or the phone number +351 221 789 456.

B. Data collection and processing

The collected data is utilized exclusively for the purposes specified in this section. It is processed with the security and confidentiality assurances demanded by the current legislation. Unless otherwise specified, the collected data is required for us to provide a functional NIDS product. If we can not collect it, we will not be able to provide the product.

i. Data collected by the Dashboard

Category: First-party session cookies.

Purpose: Authenticate and authorize a NIDS user.

Legal basis: Performance of a contract (provide access to the system).

Conservation period: From the moment the user accesses the Dashboard until the session is closed.

i. Data collected by the Watchdogs

Category: Network traffic characteristics.

Purpose: Detect malicious activity and generate alerts.

Legal basis: Performance of a contract (carry out intrusion detection).

Conservation period: From the moment it is captured until it is manually removed or the contract is terminated.

C. Your rights

In compliance with the GDPR, you can request access, rectification, deletion, limitation, opposition and portability of your data by contacting ProjCo. To prevent the disclosure of your data to unauthorized third parties, we may request proof of identification through an up-to-date personal identification document.

If you reckon that your rights have been violated, you have the right to submit a complaint to your country's National Data Protection Commission.

Appendix B SPA Interface Design

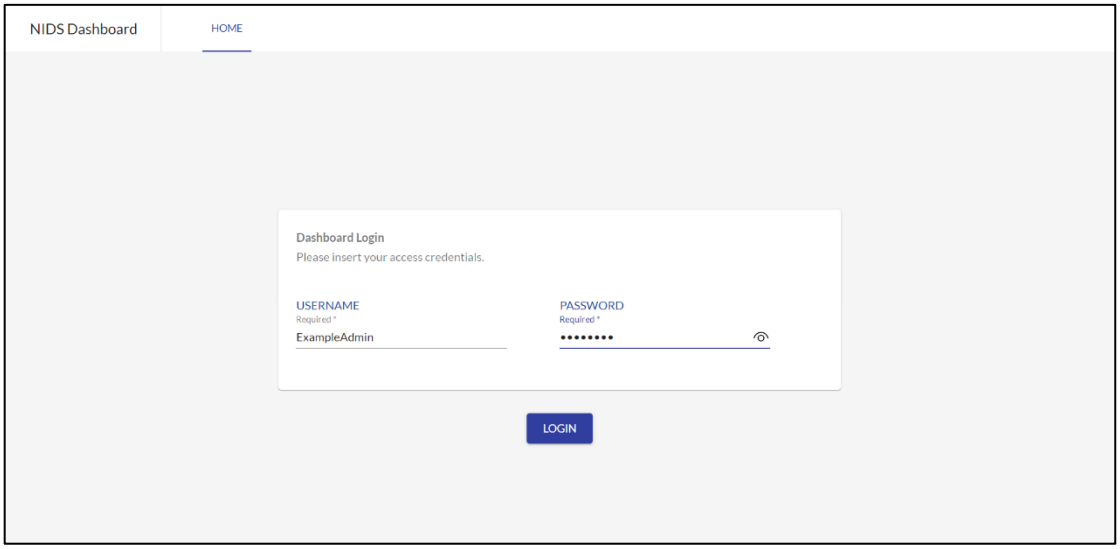

Figure 45. SPA interface design of user login.

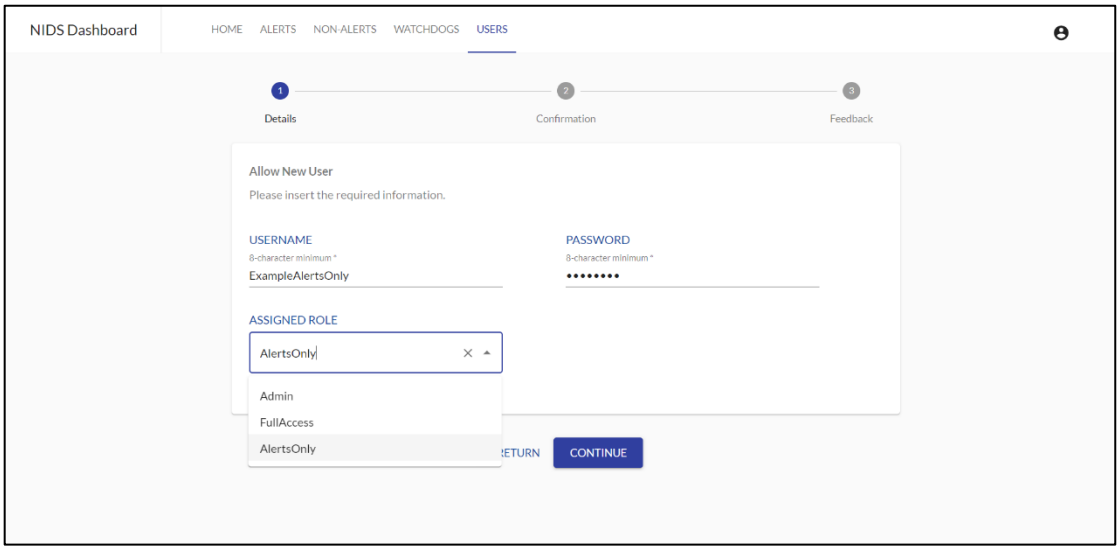

Figure 46. SPA interface design of allowing a new user.

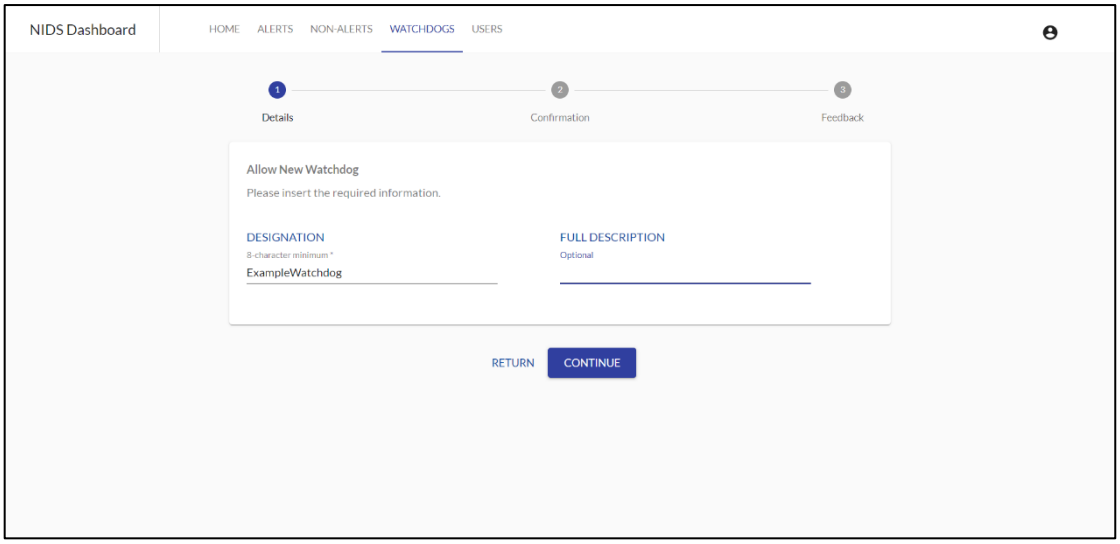

Figure 47. SPA interface design of allowing a new watchdog.

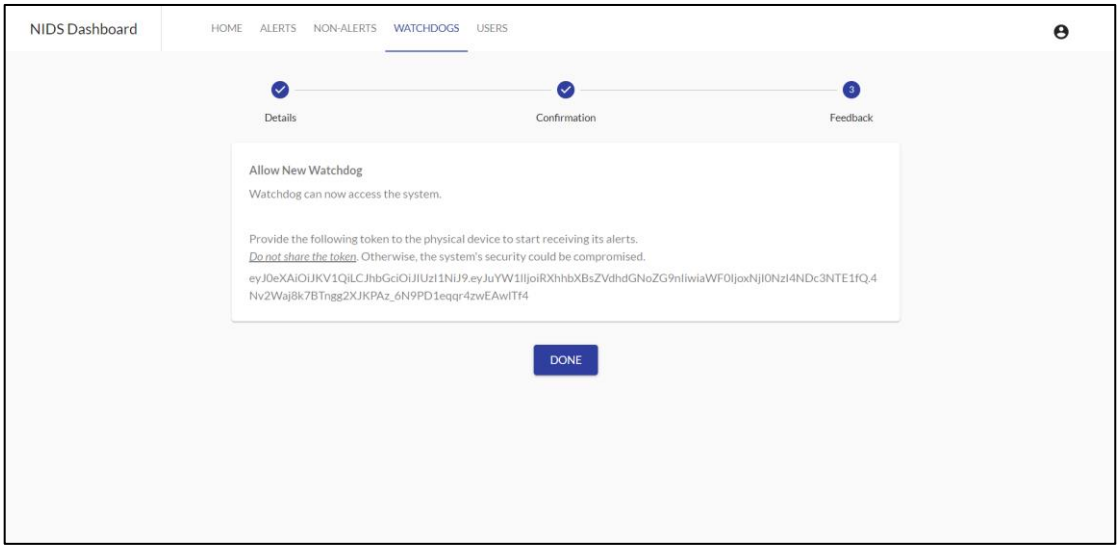

Figure 48. SPA interface design of watchdog successfully allowed.

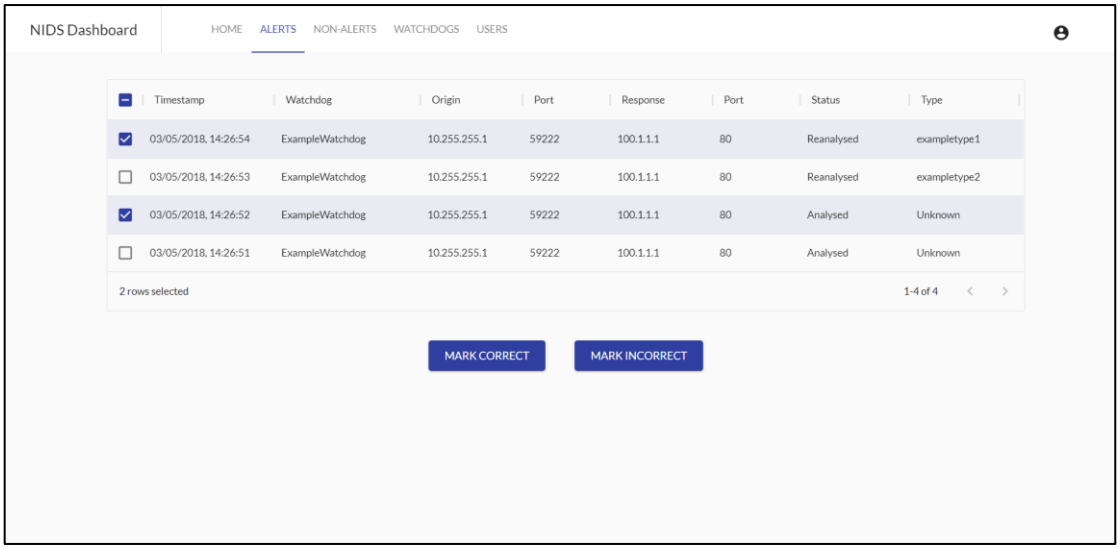

Figure 49. SPA interface design of alert listing and validation.

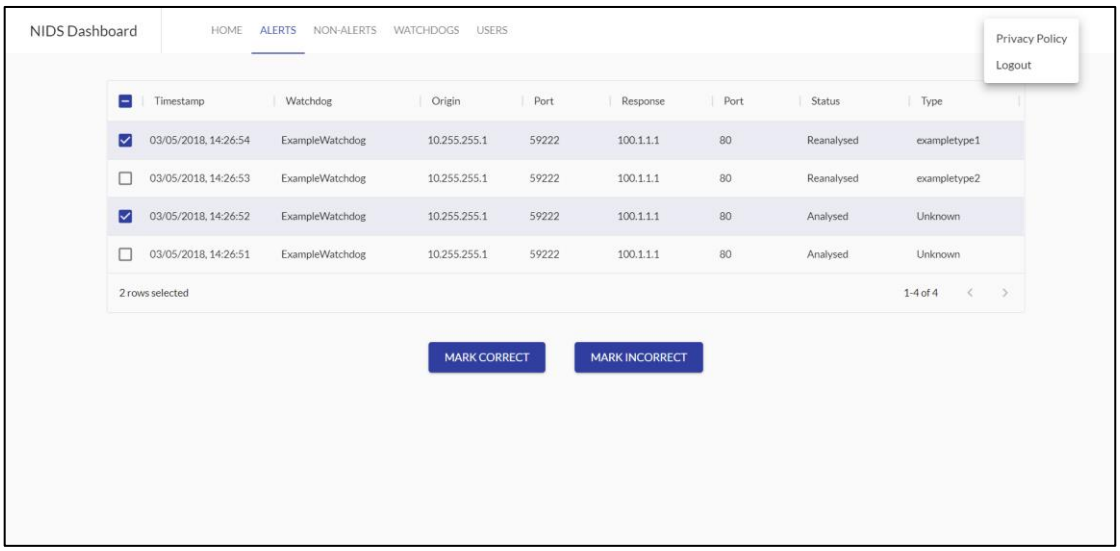

Figure 50. SPA interface design of user top button.