

POSEIDON

Protection and control of Secured Information by
means of a privacy enhanced Dashboard

GRANT AGREEMENT NUMBER: 786713
H2020-DS-2016-2017/ DS-08-2017

Deliverable 4.3

Risk Management Module
&
Personal Data Analyser
Interim implementation

List of Acronyms

Acronym	Definition
CRF	Conditional Random Fields
CNN	Convolutional Neural Network
PDA	Personal Data Analyser
AI	Artificial Intelligence
NLR	Natural Language Reasoning
NLU	Natural Language Understanding
NLP	Natural Language Processing
LDC	Linguistic Data Consortium
PII	Personally Identifiable Information
GDPR	General Data Protection Regulation
EU	European Union
EEA	European Economic Area
RMM	Risk Management Module
API	Application Programming Interface
eID	electronic IDentification
eIDAS	electronic IDentification, Authentication and trust Services
NLTK	Natural Language Toolkit
GATE	General Architecture for Text Engineering
XML	Extensible Markup Language
HTML	Hypertext Markup Language
IOB	Inside Outside Beginning
BILOU	Beginning Inside Last Outside Unit
RCV1	Reuters Corpus Volume 1
MUC	Message Understanding Conference
ACE	Automatic Content Extraction
GMB	Groningen Meaning Bank
TP	True Positive



TN	True Negative
FP	False Positive
FN	False Negative

Disclaimer

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement 786713.

This document has been prepared for the European Commission however it reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Risk Management Module & Personal Data Analyser Interim implementation











Project Title:	POSEIDON
-----------------------	-----------------

Deliverable Number:	D4.3
Grant Agreement number:	786713
Funding Scheme:	HORIZON 2020
Project co-ordinator name:	Francesco Paolo Schiavo – MEF (Italian Ministry of Economy and Finance)
Title of Deliverable:	Risk Management Module & Personal Data Analyser - Interim implementation
WP contributing to the Deliverable:	WP4 - Platform components implementation
Deliverable type	R - Report
Dissemination level	PU - Public
Partner(s)/Author(s):	Paulo Silva, Rui Casaleiro, Fernando Boavida, Marília Curado, Edmundo Monteiro, Paulo Simões, Nuno Antunes (UC)
Internal Reviewers	Veronica Della Posta (Accenture), Concepcion Cortes (Tecnalia)

History:			
Ver.	Comments	Date	Author
0.1	Initial layout and content	2019-02-11	UC
0.2	PDA research (NLP experiments)	2019-03-08	UC
0.3	Comparative study NLP prel. results	2019-05-06	UC
0.4	General content added	2019-06-24	UC
0.5	RMM approach added	2019-07-01	UC
0.6	RMM content updated	2019-07-11	UC
0.7	PoSeID-on architecture overview	2019-07-15	UC
0.8	Final update for internal review	2019-07-22	UC
0.9	Address comments and reviews	2019-07-25	UC
1	Final Version	2019-07-31	UC

Project funded by the European Commission Horizon 2020 - The EU Framework Programme for Research and Innovation.

The POSEIDON Consortium consists of the following partners:

Logo	Name	Country
	MEF – Ministero dell'Economia e delle Finanze	Italy
	ACN - Accenture S.p.A.	Italy
	PNO – PNO Innovation	Belgium
	ELEX – e-Lex Studio Legale	Italy
	TECN – Fundacion Tecnia Research & Innovation	Spain
	SAN – Ayuntamiento de Santander	Spain
	SOFT - Softeam	France
	UC – Universidade de Coimbra	Portugal
	JIBE – SMARTFEEDZ B.V.	The Netherlands
	MITA – Malta Information Technology Agency	Malta

Executive Summary

This deliverable provides details on the interim implementation of the Risk Management and Personal Data Analyser modules of PoSeID-on.

The context and scope of both modules and relation to the PoSeID-on work plan are described as well as a high-level view of the PoSeID-on platform architecture. A brief overview of each module is also presented, for contextualization purposes.

Afterward, the core of this document is presented: the implementation details of the Risk Management Module and Personal Data Analyser, going over the methodology adopted for the module's design and development, each modules architecture and architectural decisions, implementation details and finally configuration and deployment information. Testing and validation are also briefly introduced, as the current stage of development and integration do not allow for in-depth testing.

At their current stage, both modules are communicating using the protocol specified by the integration work package and are both receiving and preliminarily analysing data, using test datasets. Deployment is already possible through the use of containers. Details regarding the accomplished features and roadmap for future work can also be found in the last sections of the document.

The work developed within these modules innovates in the way it identifies privacy exposure risks by using anomaly detection on system logs and operational information, which, as far as we know, is a completely novel area. Moreover, a fully GDPR-compliant monitoring module for personal data transactions also provides privacy analysis by using NLP, Machine Learning and privacy metrics and PII-related information.

The conclusions and future work directions are discussed in the last section of the document.

TABLE OF CONTENTS

<i>Executive Summary</i>	6
<i>Glossary on PoSeID-on Terminology</i>	11
<i>1. Introduction</i>	12
1.1. Scope and Relation to the PoSeID-on Workplan	12
1.2. Document Structure	12
<i>2. PoSeID-on Requirements and Architecture Overview</i>	13
2.1. Overall Architecture	14
2.2. Main Modules	15
2.2.1. Data Subjects, Data Processors and Administrators	15
2.2.2. Web Dashboard	15
2.2.3. Data Processor API	16
2.2.4. Client-side Data Processor API	16
2.2.5. Permissioned Blockchain and Smart Contracts	16
2.2.6. Blockchain API	17
2.2.7. Risk Management Module	17
2.2.8. Personal Data Analyser	17
2.2.9. eID Provider	18
2.2.10. Data Subject's PII Repository	18
2.2.11. Message Bus	19
2.3. Development Roadmap	19
<i>3. Risk Management Module</i>	21
3.1. Module overview and goals	21
3.2. Requirements	21
3.3. Methodology	23
3.4. Detailed Architecture and Design	23
3.5. Risk Detection in PoSeID-on	25
3.5.1. Message Contents	26
3.5.2. Log Anomaly Detection Overview	26
3.5.3. Log Anomaly Detection in PoSeID-on	27
3.5.4. Data flow	30
3.5.5. Datasets	32
3.6. Development	32
3.6.1. Dependencies and Used Frameworks	32
3.6.2. Component Implementation	37
3.7. Configuration and Deployment	41
3.8. Unit Testing and Validation	42
<i>4. Personal Data Analyser</i>	44



4.1. Architecture and System Design	44
4.2. Natural Language Processing (NLP)	45
4.2.1. NLP Tools	46
4.2.2. Labelling Schemes	48
4.2.3. Datasets	49
4.3. Comparative Study NLP Tools	52
4.3.1. NLTK	53
4.3.2. Stanford Core NLP	55
4.3.3. Spacy	58
4.3.4. Summary	61
4.4. Development	62
4.4.1. Dependencies	62
4.4.2. Component Implementation	64
4.5. Configuration and Deployment	68
4.6. Unit Testing and Validation	69
5. Integration approach	69
5.1. Component communication	69
5.2. Interim version features	70
5.3. Final version features	71
6. Innovation Summary	72
7. Conclusion and future work	72
References	73

List of Figures

Figure 1 - PoSeID-on General Architecture	14
Figure 2 - Roadmap for T4.2 and T4.3	19
Figure 3 - Risk Management Module Architecture	24
Figure 4 - Graphical example of log anomaly detection [12]	27
Figure 5 - RMM Data Flow	31
Figure 6 - Personal Data Analyser Architecture	45
Figure 7 – NLTK time spent to train the model	54
Figure 8 – NLTK F1 Scores	54
Figure 9 – NLTK overall results	55
Figure 10 – Stanford CoreNLP time spent to train and number of iterations	56
Figure 11 – Stanford CoreNLP F1 Scores	57
Figure 12 – Stanford CoreNLP overall results	57
Figure 13 – spaCy time spent to train and number of iterations	59
Figure 14 – spaCy F1 Scores	59
Figure 15 – spaCy overall results	60
Figure 16 - spaCy time spent to train and number of iterations (under 500)	60
Figure 17 - spaCy F1 Scores (under 500 iterations).....	61
Figure 18 - NLTK, Stanford CoreNLP and spaCy F1-Score Overall Results Comparison	61

List of Tables

Table 1 - D4.3 Revision and Submission Roadmap	20
Table 2 - Risk Management Module Requirements	22
Table 3 - Message Parameters	26
Table 4 - Summary of Automated Log Parsing Tools [16]	28
Table 5 - Comparison of NLP Tools General Features	47
Table 6 - Comparison of NLP Tools Technical Features	47
Table 7 - Labelling Schemes Examples	48
Table 8 – Corpus' Characteristics Summary	51
Table 9 - Metrics Formulation	52
Table 10 – Default Named Entity Recognition Scores from NLP Tools	53
Table 11 - NLTK v3.4 NER Scores - GMB (Kaggle) Dataset	54
Table 12 - Stanford CoreNLP v3.9.2 NER Scores - GMB (Kaggle) Dataset	56
Table 13 - spaCy v2 NER Scores - GMB (Kaggle) Dataset	58
Table 14 - Interim Version Features (PDA and RMM)	70

Glossary on PoSeID-on Terminology

- GDPR – General Data Protection Regulation, (EU) 2016/679 - is a regulation in EU law on data protection and privacy for all individuals within the European Union (EU) and the European Economic Area (EEA).
- Informed consent – freely given, specific and informed indication of the wishes of a Data Subject, by which he/she agrees to PII being processed.
- Data Subject - a natural person that represents the primary target of GDPR.
- Message bus – a combination of a common data model, a common command set, and a messaging infrastructure to allow different PoSeID-on components to communicate through a shared set of interfaces.
- Permissioned Blockchain - a blockchain where every node and every user must be granted permissions to utilize the system. Permissions are generally assigned by an administrator.
- Personal Data Analyser – a PoSeID-on component that monitors personal data transactions and related blockchain warnings in order to detect and prevent anomalies and ill-intended transactions.
- Personal Identifiable Information (PII) – information related to a Data Subject, that can be used to directly or indirectly identify the person.
- PII metadata – a set of data that describes and gives information about PII data. Depending on circumstances, metadata may be privacy-sensitive and require the same privacy protection mechanisms that are applied to PII.
- PII Type – category of PII, such as 'First name', 'Last name', 'Date of birth', 'Postal address', 'Contract', etc.
- PII Value – an instantiation of a PII Type, such as 'John', 'Smith', 'June 5th, 1959', or even an entire document, such as a signed contract, etc.
- Pseudonym – an alternate name for a Data Subject so that he/she cannot be directly identified. Identification can still be done with the use of additional data that, typically, is kept outside the main system.
- Risk Management Module – a PoSeID-on component that assesses and manages privacy risks by analysing transactions information.

1. Introduction

Despite the widespread use of digital services, end-users remain worried about data privacy, data protection, digital identities, and data ethics. The goal of the PoSeID-on Project (“Protection and control of Secured Information by means of a Privacy Enhanced Dashboard”) is to develop a transparent ecosystem for personal data protection, supporting the pillars of the EU’s new General Data Protection Regulation (GDPR) with regard to digital security.

In this deliverable, we document development details regarding the interim implementation of two crucial modules of the PoSeID-on Project: the Risk Management Module, and the Personal Data Analyser module. Additionally, as considerable research and experimentation were necessary before starting the actual development of these modules, conclusions and results obtained during this exploratory phase are also described and analysed, as they provide both background and rationale for the choices that were taken in the development.

The technical details comprise the development and implementation, dependencies, deployment, and integration of the respective modules. Although deliverable 5.1 provides a fully detailed integration approach, we provide a high-level description of the integration approach adopted in the specific cases of the RMM and PDA modules, for contextualisation and completeness.

The remainder of this section sets the scope of the current deliverable, explains its relation to the PoSeID-on workplan, and describes the document structure.

1.1. Scope and Relation to the PoSeID-on Workplan

This report is one of the initial deliverables of Work Package WP4 (“Platform components implementation”). It has been produced within the scope of Tasks T4.2 (“Risk Management Module implementation”) and T4.3 (“Personal Data Analyzer implementation”), and its goals are the following:

- Building on the outcome of Deliverable D2.2 (“System Requirements and Architecture”) [1], proceed with a detailed specification of two of the modules (RMM and PDA) previously identified in the PoSeID-on’s general architecture.
- Document the stage of development (i.e., interim version) of the Risk Management Module and Personal Data Analyzer and respective implementation details. The final version is to be documented in Deliverable 4.4 (“Risk Management Module & Personal Data Analyzer - Final implementation”).
- Provide a high-level view of the integration approach adopted for the RMM and PDA modules in relation to the general PoSeID-on architecture, which is further discussed in Deliverable D5.1 (“Preliminary PoSeID-on Integrated Platform”) [2] from Work Package 5.

1.2. Document Structure

This deliverable is structured as follows:

A terminology/glossary has been provided at the beginning of the document, presenting short definitions and/or descriptions of selected terms used throughout this deliverable. While some

of these terms are generic (e.g. blockchain, data subject), others are specific to the PoSeID-on project and extensively used in the technical discussions and descriptions throughout this document.

Section 1 identifies the scope/goals of the document and its relation with the overall workplan of PoSeID-on.

Section 2 reviews the PoSeID-on project requirements and general architecture. This allows the readers less familiar with the PoSeID-on project to understand the rationale behind the identified requirements and architectural design decisions, providing the user with a basis for more technical details in the sections that follow.

Section 3 presents the work developed in the scope of the Risk Management Module (RMM). It describes the methodology and approach taken in the detection of privacy risks in the PoSeID-on project, including an overview of the applied state-of-the-art techniques. This is followed by the description and design of the module's architecture, implementation details, configuration and deployment and, finally, testing and validation.

Section 4 presents information regarding the Personal Data Analyser (PDA) module. The architecture and design of the module are described in detail. This is followed by an overview of NLP and a comparative study of NLP tools. Development details are given, followed by configuration, deployment, testing and validation details.

Section 5 presents a summary of the integration approach followed in the interim version of the PDA and RMM modules, as well as the roadmap for the final version.

Section 6 provides an innovation summary, within the scope of technologies addressed in this document.

Section 7 provides the main conclusions and identifies the next steps for the development and integration of the PoSeID-on platform.

2. PoSeID-on Requirements and Architecture Overview

This section describes the PoSeID-on system architecture from a high-level point of view, introducing the core components and actors of the platform. A thorough and in-depth description of the architecture and architectural decision can be found in deliverable D2.2 [1] of the project. The objective of this section is to provide background and contextualize the descriptions of the Personal Data Analyser and the Risk Management Module interim implementation provided in subsequent sections, and, as such, only an overview of the PoSeID-on architecture and main modules will be given.

PoSeID-on aims to provide a Privacy Enhancing Dashboard for personal data protection, a platform that manages the personal data transactions between data subjects and private or public entities processing or storing their data. PoSeID-on also aims to manage direct transactions of PII between data processors, with the consent of data subjects. All relevant information concerning the data subject and his/her personal information shall be made available through a user-friendly and accessible web dashboard. In addition to providing tracking of PII exchanges and consent management, PoSeID-on will also provide the data

subjects with a risk score or reputation associated with the data processors making use of their PII. In order to reduce identity fraud and protect the privacy of users, access to the PoSelD-on Dashboard is to be made available through eID accounts only, in line with the eIDAS regulation. In this context, the RMM module aims to analyse the PoSelD-on platform's behaviour in order to detect possible privacy risks and system problems, while the PDA module aims to analyse the actual content of PII transactions in order to detect personal data which is not covered by the permissions given to data processors.

2.1. Overall Architecture

PoSelD-on envisages a solution leveraging permissioned blockchain technologies and smart contracts. Through the use of smart contracts, data confidentiality, access control and transparency of operations will be guaranteed. Data controllers and data processors will only be able to access PII when consent is given by data subjects. The following figure illustrates PoSelD-on's general architecture, including its main components and their interactions.

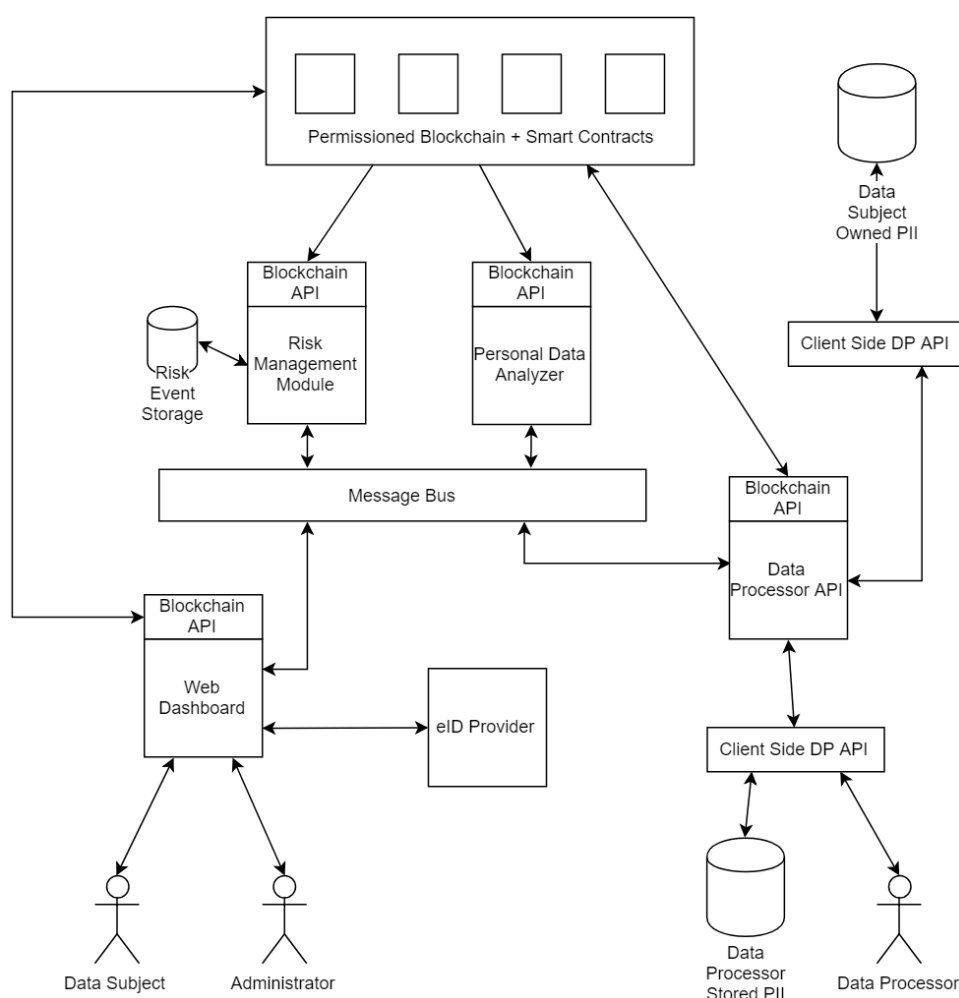


Figure 1 - PoSelD-on General Architecture

As can be seen, both the Risk Management Module and the Personal Data Analyser communicate with other platform components through the message bus and the blockchain API. Details regarding these interfaces and the information being exchanged between

components can be found in deliverable D2.2 [1], and in Sections 3 and 4 of the current deliverable for the Risk Management Module and Personal Data Analyser, respectively.

2.2. Main Modules

This sub-section presents an overview of the main modules of the PoSelD-on platform. For a detailed description of the modules, each deliverable associated with the development work packages of each module can be consulted. As described in D2.2, the main modules are the web dashboard, data processor API, client-side data processor API, permissioned blockchain and smart contracts, blockchain API, risk management module, personal data analyser, eID provider, data subject's PII repository, and message bus. These are described below.

2.2.1. Data Subjects, Data Processors and Administrators

PoSelD-on considers three different types of actors:

- **Data subjects** are natural persons that represent the primary target of GDPR. Data subjects have **Personally Identifiable Information** (PII) that constitute valuable resources for third-parties and represent a privacy risk. Nevertheless, in several situations data subjects need to share this PII with third parties (such as insurance companies, health institutions, tax services, other public services, banks and so forth), either directly (data subject provides the PII) or indirectly (a third party which already has some PII shares this PII with another third party, after permission is given by the data subject – this is the case, for instance, of a tax service getting revenue reports directly from a bank).
- Third parties exchanging PII with the data subject or between themselves are designated as **data processors**. Depending on the granted permissions and on the nature of exchanged PII, several models of exchange may occur. For instance: the data subject provides PII to a data processor for a single operation limited in time and scope; the data subject provides PII to a data processor for multiple operations (of a predefined type) but with a limit in time; the data subject provides PII to a data processor for an unlimited period of time. Depending on the nature of the permissions granted by the data subject, data processors may store PII and share this PII with other data processors (within the limits established by the permissions). The PII storage mechanisms actually employed by each data processor fall outside the scope of PoSelD-on. However, the data processor is supposed to comply with data subject's requests and permissions regarding retention, deletion, and processing of their PII, due to regulatory and legal obligations.
- The **Administrators** are the operators managing the PoSelD-on platform. The platform is operated by a preassigned entity – typically a public or semi-public organization with the mission of providing a PII management framework such as PoSelD-on. While this entity is usually considered as a trusted third-party, the whole platform design tries to minimize the risk of intentional or accidental breaches of privacy originating on the management entity. Administrators manage the PoSelD-on platform on behalf of this entity.

2.2.2. Web Dashboard

The interface provided to data subjects, to access the PoSelD-on platform, is a web-based application that provides access to the various types of operations performed by data subjects,

such as, for instance, granting, modifying and revoking permissions for a specific data processor, checking the history of exchanges of their PII, and receiving alarms due to high privacy exposure risks. This human-oriented dashboard is the primary interface for data subjects, although in some situations other communication channels may be used according to the data subject preferences (e.g., receiving urgent alarms of privacy exposure via email or SMS). Access to the web-based dashboard is based on logging with the data subject's national eID (or similar credentials, depending on the specific PoSeID-on instance).

The web dashboard also provides an interface for PoSeID-on administrators, although with a different set of functionalities and user interfaces.

2.2.3. Data Processor API

The access point for Data Processors, for communicating with PoSeID-on, is the Data Processor API. While the interface for data subjects and administrators (the dashboard) is primarily human-oriented, the interface for Data Processors is focused on interconnecting the Data Processor's already existing information systems with PoSeID-on (so that their business logic can integrate with PoSeID-on functionalities). This application will publish an authenticated API over which Data Processors can send requests for/about PII to PoSeID-on. This application will also send the Data Processor messages through their Client-Side Data Processor API, about changes in PII values and PII deletion requests.

Note: Prior to D2.2 completion, this module was referred to in documents as Gateway API. After several discussions, it was agreed that the name was misleading in what concerns the functionalities it provides and, as such, was renamed Data Processor API.

2.2.4. Client-side Data Processor API

In order to connect to the PoSeID-on platform, data processors can use a client-side Data Processor API, which is made available by Data Processors to interface with their PII store. This API manages the transport of PII and the revocation of permissions. The API will need to be well secured and authenticated.

PoSeID-on itself will also develop one Client-side Data Processor API, to manage communication with the Data Processor API which will be implemented for the PoSeID-on platform to store PII that has no central authority otherwise (PII like personal phone numbers, for example). From the rest of the system's point of view, this is just another Data Processor. This PoSeID-on local Data Processor can also act as a reference implementation for the Data Processor API specification.

Note: Prior to the D2.2, in some early documents, this module was referred to as the "Data Processor API". As a result of the discussion of the renaming of the Gateway API, that led to it being renamed Data Processor API, the previous Data Processor API was renamed Client-side Data Processor API.

2.2.5. Permissioned Blockchain and Smart Contracts

The PoSeID-on permissioned blockchain consists of a special-purpose blockchain implementation that only works within the PoSeID-on system. It will be permissioned, meaning that instead of a proof-of-work or proof-of-stake consensus, a central authority will provide the

permission to participate in the PoSelD-on Blockchain network. This will cause faster transaction speeds and will protect the implementation from the risks public blockchains are facing. The functionality of the system will be controlled by Smart Contracts stored within this blockchain, and they will describe the management of the requests and permissions to grant, deny and check PII access. The blockchain nodes are expected to be hosted by the PoSelD-on administration entity and the participating data processors. It is also possible to allow the inclusion of nodes hosted by other entities – which might make sense in some scenarios.

2.2.6. Blockchain API

The blockchain API abstracts all Blockchain operations into a high-level API suitable for integration into other applications. Since the use of Blockchain carries some important implications on how the clients and the servers behave, actions like account management (Data Processor and Data Subject identity on the Blockchain) or system functionality (Smart Contract functions usage) change drastically. For instance, users shall locally sign every call to a Smart Contract, but other components in the overall PoSelD-on architecture, like the Web Dashboard Module might function normally despite the existence of a Blockchain network behind it. Modules writing information to the Blockchain ledger must have a mechanism to access the network and work against it. This nexus will be the Blockchain API, that will also allow the intervening modules to feed the Risk Management and Data Process Analytics Module.

The Blockchain API is not a web-service, but it will be directly used as a wrapper from the client browser, without intermediaries, providing a direct connection to the Blockchain Module. This API will only be accessible once the user has been authenticated by the eID Provider Module (implementing eIDAS).

2.2.7. Risk Management Module

The risk management module (RMM) is responsible for monitoring PoSelD-on, both from a system-wide perspective and from the point of view of individual data subjects' operations.

RMM is expected to detect and evaluate possible security and privacy risks (such as anomalous behaviour from a specific data processor which suddenly begins collecting much more data than usual from a large set of data subjects, which means the data processor may have been hacked and is being used to syphon PII to external attackers) or risks associated with a specific data subject (such as successive attempts to login with his/her credentials). Risk detection is made combining machine learning algorithms, that analyse multiple sources of information about transactions, user-level behaviour and system-level behaviour. When the data subject provides explicit consent, specific data transactions and PII might also be used for this analysis. High-risk levels may trigger alerts to PoSelD-on administrators and Data Subjects, depending on the RMM settings.

2.2.8. Personal Data Analyser

The Personal Data Analyser (PDA) will be used to monitor personal data transactions and related warnings generated by the blockchain platform, in order to detect and prevent anomalies and misbehaved transactions. A warning is generated each time a transaction is not received and approved by all the interested parties.

This component is used to control personal data in a transaction, with the aim of discovering all previously non-identified personal data, such as personal data for which there is no data subject authorisation. Due to the sensitive nature of the analysed data (i.e., Personally Identifiable Information), the PDA only acts when explicit consent is provided by the user. Furthermore, all data is discarded after each analysis. In case the user does not provide explicit consent, the PDA will simply not operate for data from that specific data subject. While this scenario is not the most desirable, it does not affect other data subjects.

2.2.9. eID Provider

An identity provider which, in line with the European eIDAS regulations and ecosystem¹, authenticates persons and organisations on behalf of the PoSeID-on platform.

2.2.10. Data Subject's PII Repository

Data Processors may receive PII directly from the data subject or from (allowed) transactions with other data processors. In the first case, it is important to provide an easy-to-use mechanism for the data subjects to insert, store and update this PII. While this functionality could be provided by the Dashboard, this would represent a mix of roles that would allow the PoSeID-on administrator to potentially bypass existing privacy restrictions (besides adding unnecessary complexity to the dashboard). Therefore, it is considered that the Data Subject always uses a Data Processor for storing his/her PII, even when this data processor has no other specific purpose (in practice it acts as just a storage point for the data subject's PII). This component will be responsible for storing all the PII considered private to a Data Subject for which no authoritative Data Processor exists. It will interface with the rest of the PoSeID-on system by using the same Data Processor API as all other Data Processors. The only difference is that this component acts as a Data Processor with the Data Subject's credentials. All data stored within this component is encrypted with the Data Subject's public key, instead of a Data Processor-specific key.

Depending on deployment options, storage itself may take place:

- on-premises with the rest of the PoSeID-on components, though due to the adopted encryption mechanisms the PoSeID-on authority actually has no access to the data subject's PII – therefore meeting the privacy requirements previously discussed (whenever the same authority manages the PoSeID-on platform and this data subjects PII repository, as will happen in the planned pilots)
- or in the infrastructure of third-party entities, when this service is (hypothetically) provided by third parties selected by the data subject. The adopted encryption mechanisms still prevent unauthorized access to the data subject's PII in this case. Also, it should be noted that this deployment model is supported by the conceptual architecture but will not be tested and/or adopted by any of the planned PoSeID-on pilots.

A side-benefit of this approach is the fact that, in this way, an example of Data Processor will be developed and made available to the (real) data processors participating in the Pilots, as a showcase of how to use the Data Processor API to access PoSeID-on services. Data processors can then use this example as a starting point to interconnect their own information systems with PoSeID-on.

¹ <https://ec.europa.eu/cefdigital/wiki/display/CEFDIGITAL/eID>

2.2.11. Message Bus

This component provides a messaging infrastructure for PoSeID-on components to communicate with each other in a controlled but decoupled fashion, which will allow for the easy addition and removal of components without affecting the overall system operations and asynchronous communication, facilitating scalability and fault tolerance.

The different components of PoSeID-on can be seen as individual applications. Messages are passed between them, through a queue mechanism, each and every one signed by the sender and encrypted for the recipient. When a component is unreachable (for instance, a Data Processor is offline), the message will be kept until either a predefined timeout (e.g., two weeks) passes or the recipient comes back online.

In this way, PoSeID-on can stay easy to reason about, easy to test, easy to develop and maintain by many distributed parties, and easy to scale up when the need arises.

2.3. Development Roadmap

Before deliverable D.2.2 - “System Requirements and Architecture”, an overall roadmap for the tasks of each Work Package was agreed upon. As the activities started, and with a complete system architecture available from D2.2, a more fine-grained roadmap was deemed necessary for the sake of good planning and execution of tasks. Therefore, several milestones were set for each WP4-related task (T4.2 and T4.3). Below, we provide the details of the milestones defined for each task.

ACTIVITIES	MILESTONES	DESCRIPTION OF THE ACTION
Task T4.2 – RMM		
Exploratory studies & detailed design (M3-M10)	28/02/2018	Detailed RMM Design and implementation plan.
RMM implementation – Interim (M14)	30/06/2019	RMM initial implementation, based on low-level design.
D4.3 – Deliverable writing (M14)	30/06/2019	Report on RMM and PDA interim implementation ready for internal review
D4.3 – Deliverable review & submission (M15)	31/07/2019	Revised report ready for submission
Interim version analysis & design updates (M17)	30/09/2019	Analysis of interim version and design updates for final version
RMM implementation – Final version (M23)	30/04/2020	RMM final implementation
D4.4 – Deliverable writing (M23)	31/03/2020	Report on RMM and PDA final implementation ready for internal review.
D4.4 – Deliverable review & submission (M24)	30/04/2020	Revised report ready for submission
Task T4.3 – PDA		
Exploratory studies & detailed design (M3-M10)	28/02/2018	Detailed PDA Design and implementation plan.
PDA implementation – Interim (M14)	30/06/2019	PDA initial implementation, based on low-level design.
D4.3 – Deliverable writing (M14)	30/06/2019	Report on RMM and PDA interim implementation ready for internal review
D4.3 – Deliverable review & submission (M15)	31/07/2019	Revised report ready for submission
Interim version analysis & design updates (M17)	30/09/2019	Analysis of interim version and design updates for final version
PDA implementation – Final version (M23)	30/04/2020	PDA final implementation
D4.4 – Deliverable writing (M23)	31/03/2020	Report on RMM and PDA final implementation ready for internal review.
D4.4 – Deliverable review & submission (M24)	30/04/2020	Revised report ready for submission

Figure 2 - Roadmap for T4.2 and T4.3

Figure 2 lists the roadmap for Tasks 4.2 and 4.3. The exploratory studies and detailed module design are completed for RMM and PDA. Moreover, implementation is proceeding according to schedule and heading towards the final version of the implementation, which is due on April

30th, 2020. It is worth mentioning that task T4.1 is not discussed in this document since it has its own deliverable, namely Deliverable D4.1 [3].

<i>Date (July)</i>	<i>Description</i>
8 th -9 th	Internal revision
10 th -11 th	GA Discussion
15 th - 19 th	Document Revision (ACN + TCN)
22 nd - 26 th	Accommodate review feedback or changes
29 th	Final document verification
30 th	Submission

Table 1 - D4.3 Revision and Submission Roadmap

Table 1 lists the deadlines that were internally agreed and set for a seamless finalisation of D4.3.

3. Risk Management Module

This section details the Risk Management Module. As previously mentioned, a description of the RMM functionality (without implementation details) can be found in deliverable D2.2 [1] which presents the overall architecture and requirements of the PoSeID-on platform.

This section starts by providing an overview of the module goals and an identification of requirements. Subsequent subsections present and discuss the used methodology, module architecture and design, risk detection approach, module development, preliminary results, configuration and deployment, and unit testing and validation.

3.1. Module overview and goals

The Risk Management Module is responsible for monitoring PoSeID-on, both from a system-wide perspective and from the point of view of operations involving individual data subjects. The RMM is expected to detect and evaluate possible security and privacy risks inherent to the processing of personal data, such as the exposure of PII through the malicious use of the PoSeID-on platform. An example of such risk is a data processor which suddenly begins collecting much more data than usual, requesting permission to access several types of PII from a large number of data subjects, while normally such data processor would collect information regarding one or two PII types from a small set of data subjects. This can indicate a data processor is syphoning information or has been compromised.

Besides detection of possible risks, the RMM must also associate a risk score to data processors, evaluating them according to their previous behaviour on the platform. This allows the dashboard to advise which services can be trusted or should be disabled.

In addition, the RMM must notify the administrators of PoSeID-on and the data subjects involved in possible privacy risks, in order for them to take the necessary counter-measures, such as disabling services, when risk is identified.

In order to do this, RMM resorts to machine learning and well-known frameworks for real-time analysis of the data provided by PoSeID-on to the module.

As of the time of writing, the agreed-upon sources of information are system logs, provided by each module within the platform and, when data subjects provide explicit consent, PII transactions and management operations metadata which is not included in logs for security and privacy reasons.

3.2. Requirements

The identified requirements for the risk management module can be seen in Table 2. These requirements were taken from the PoSeID-on system requirements delivered as annex on deliverable D2.2 and their identification numbers correspond to the ones on that deliverable, for the sake of consistency between documents.

Requirement ID	Title	Description
R102	RMM Data Sources	RMM will receive information from the blockchain through the blockchain API and information from the API Gateway

R103	RMM Risk Detection	RMM will detect risk events based on blockchain transactions and previous stored risk events information
R104	RMM Risk Events	The RMM will create, update and delete PII risk events
R105	RMM Notifications	Depending on the nature of risk events notifications will be delivered to involved data processors, data subjects and PoSeID-on platform administrators
R106	RMM PII in Risk Events	Risk Events may include PII in case of explicit consent by the data subjects and data processors to the RMM
R107	RMM Risk Event Storage	Risk events will be logged in a secure way for later analysis, forensics and processing
R108	RMM Data Processor Reputation	RMM will associate a reputation score with each data processor based on previous track record
R109	RMM DP Reputation	Reputation scores will be created, updated and deleted by the RMM
R110	RMM Risk Notification Delivery	RMM will notify identified parties in a risk event through the web dashboard or other channels specified for this purpose
R111	RMM Risk Prediction	Risk Management Module will predict risks based on reputation, PII transactions and system logs

Table 2 - Risk Management Module Requirements

In addition to these requirements, the following quality attributes were also identified:

Performance

The RMM is expected to output real-time results. By real-time, it is understood as within the length of a user session on the web dashboard. The results do not need to be instantly made available, as the user will not be waiting for the feedback and may return to other dashboard activities, but results of user actions which have been identified as a risk should arrive within the active session.

Scalability

The potential volume of data subjects managed by PoSeID-on is in the order of millions, as explained in the Italian Ministry of Economy and Finance's use case presented in deliverable D2.1 [4]. As such, the module should be built using technologies that can scale horizontally to this magnitude of users.

Availability

A system of this nature, focused on ensuring the privacy of data subjects, must be able to process all requests within a user session and also provide fault tolerance and graceful degradation. Multiple deployments of the module should be possible, either by working in parallel or processing data in a distributed manner.

Privacy and Security

The RMM will be dealing with PII if consent is given by data subjects. Although the RMM does not process PII provided to the PoSelD-on system or associated data processors by data subjects, it processes and stores metadata regarding PII exchanges and system operations, such as reputation about the entities involved in an exchange and the type of data being exchanged. As such, it should comply with the GDPR and provide the same assurances as data controllers. This means that if PII is stored it should be encrypted at rest, communications with other modules should be done through secure and authenticated channels, and revocation of data access should be possible.

3.3. Methodology

In order to define an approach to accomplish the goals set for the module, several points had to be taken into consideration.

First, RMM results must be accurate and made available in near-real-time. Secondly, as risk management in this context is quite novel and we are developing a new system, there are currently no labelled or unlabelled datasets which can be utilized to model the behaviour of a PII management and distribution system. Without training data, the immediate solution was to select mechanisms which were able to model the behaviour of the system without previous knowledge.

Furthermore, the RMM should also be able to adapt to changes as long as they reflect the normal behaviour of the system, as any new platform is expected to see a growth in usage with time and adoption. Also, with the increase in visibility and growth, the number of potentially malicious attacks or usage is also expected to grow, so the detection mechanisms should be able to account for new kinds of attacks or malicious patterns of usage.

Last but not least, the sources of data and the volume of data being sent to and processed by the RMM pose a limit to the approaches that can be taken, especially when considering the near-real-time operation requirement.

With these points in mind, we concluded that developing a module focused on the detection of anomalies in the patterns of data generated by the PoSelD-on system was the best approach to take. Using a continuous unsupervised learning approach addresses the concerns expressed above. Moreover, by combining stream-based anomaly detection and batch analysis of the data, it is possible to increase the accuracy of the risk predictions while allowing analysis reports in near-real-time.

In the following sections, the architecture designed and implemented to accommodate the risk detection approach selected is explained, followed by the description of the anomaly detection mechanisms implemented.

3.4. Detailed Architecture and Design

As previously explained, the Risk Management Module will evaluate and manage privacy and operational risks within the PoSelD-on system, through analysis of operational logs and PII exchanges. The result of this analysis will be an identification of data processors and data subjects who are involved in an anomalous pattern of logs, resulting in possible updates to

The diagram illustrates the architecture of the Risk Management Module (RMM), organized into three layers: Speed Layer, Batch Layer, and Serving Layer. The RMM is connected to external components: Dashboard, Data Processor API, and Blockchain, all of which interact with a central Message Bus. The Message Bus, in turn, interacts with the RMM Message Broker.

External Components:

- Dashboard**: Provides input to the Message Bus.
- Data Processor API**: Provides input to the Message Bus and receives output from the Message Bus.
- Blockchain**: Provides input to the Message Bus and receives output from the Message Bus.

Message Bus: A central component that facilitates communication between the external components and the RMM Message Broker.

Risk Management Module (RMM) Components:

- RMM Message Broker**: A component that acts as a central hub for the RMM, receiving input from the Message Bus and distributing it to the Speed Layer and Batch Layer.
- Speed Layer**:
 - Stream Processor**: A component that receives input from the RMM Message Broker and outputs to the Rule set Temporary Dataset.
 - Rule set Temporary Dataset**: A storage component that receives input from the Stream Processor and outputs to the Stream Analyst.
 - Stream Analyst**: A component that receives input from the Rule set Temporary Dataset and outputs to the Risk Manager.
- Batch Layer**:
 - Batch Processor**: A component that receives input from the RMM Message Broker and outputs to the Master Dataset.
 - Master Dataset**: A storage component that receives input from the Batch Processor and outputs to the Batch Analyst.
 - Batch Analyst**: A component that receives input from the Master Dataset and outputs to the ML Model Builder.
 - ML Model Builder**: A component that receives input from the Batch Analyst and outputs to the Monitoring Module.
- Serving Layer**:
 - Risk Manager**: A component that receives input from the Stream Analyst and outputs to the Notification Dispatcher.
 - Notification Dispatcher**: A component that receives input from the Risk Manager and outputs to the Monitoring Module.
 - Reputation Manager**: A component that receives input from the Monitoring Module and outputs to the Monitoring Module.
 - Monitoring Module**: A component that receives input from the Notification Dispatcher and the Reputation Manager, and outputs to the Monitoring Module.

The diagram uses standard UML notation: components are represented by rectangles with a component icon, storage components by cylinders with a storage icon, and dashed boxes represent layers. Arrows indicate the direction of data flow.

24

This architecture is divided into three layers: a batch layer, a speed layer, and a serving layer. Messages arriving at the RMM are pre-processed into structured parameter lists and then sent to both the batch and the speed layers.

The batch layer manages a master dataset which is used for historic risk analysis, taking advantage of having a large collection of data, whose time span will depend on how long the RMM is allowed to retain data, provided the data subject has given explicit consent for this purpose. This historic risk analysis takes significant time and is not expected to work in near real-time. Instead, this layer provides warnings in case of risk detection in an "offline" manner. Having a batch layer also provides the advantage of being able to use machine learning models which do not have a stream-based counterpart. This layer is also in charge of training and updating machine learning models for use in near real-time analysis by the speed layer, for algorithms that require an offline training step, as is the case with most supervised learning algorithms.

The speed layer deploys the models created by the batch layer, in addition to the clustering models, and analyses the stream of data in real-time, using the data that arrives between batch analysis instances. This layer reasons over the data and, in case an anomaly is detected, dispatches a recommendation action request to the serving layer.

The serving layer is in charge of receiving the results from both the batch and speed layers, and of notifying the respective entities about risk exposure through the Dashboards interface. It is also in charge of receiving feedback from administrators regarding such risk notifications, by providing the identification of the log window where the anomaly was identified, and allowing them to confirm or deny that a real risk is present within it. According to the administrator's feedback and the history of anomalous logs involving each data processor, the associated risk reputation of data processors is also updated.

Additional architectural information, such as the details regarding each component of the module, can be found in deliverable D2.2.

3.5. Risk Detection in PoSeID-on

In the scope of the PoSeID-on project, the taken risk detection approach is a combination of domain specific feature extraction and the generic well-known approach for log analysis anomaly detection [9] [10] [11].

In short, information regarding PII handling operations and logs collected from each component operating within PoSeID-on are sent to the Risk Management Module as free form text plus a set of parameters, included in the "*params*" field of the message protocol defined in D5.1.

As messages arrive and are collected within the RMM, a model is created, representing the normal behaviour of the system (or normal pattern of PII operations and logs). Successive messages arriving at the Risk Management Module are grouped into sequences and analysed in order to understand if the sequence under analysis falls into the regular pattern expected for system behaviour or not. If it does, the expected behaviour is updated by calculating a new model based on the old model and new values. If not, the sequence is considered an anomaly and a notification is dispatched to inform about this occurrence. Later on, the sequences of events that triggered the anomaly event can be checked by a security analyst or system administrator for further action and correction. In the following sections, a more

detailed description of the steps involved in anomaly detection based on log mining is given, as well as a detailed explanation of the PoSeID-on specific implementation of these steps.

3.5.1. Message Contents

Messages arriving at the RMM contain a set of parameters, including some or all of the following set of parameters:

Parameter	Description
Module	The module issuing the message
Operation	The operation that is being performed, where applicable. Some logs may be associated with specific operations, such as PII Permission Requests, PII Permission Grants, PII Permission Revocation, PII Transacted/Accessed.
Sender ID	The ID of the issuer of a certain operation, where applicable. If it is a system operation it can be the ID of the module producing the message. If it is a PII related operations it can be a Data Subject ID or a Data Processor ID (only sent when consent is given).
Sender IP	The IP originating the message.
Receiver ID	The ID of the target of a certain operation, where applicable. If it is a system operation it can be the ID of the module receiving the request. If it is a PII related operations it can be a Data Subject ID or a Data Processor ID (only sent when consent is given).
GELF Log	The default fields sent to GELF log manager such as the host, short message, full message (raw log) and syslog level.

Table 3 - Message Parameters

The operation description parameter allows us to extract a feature vector consisting of the count of each operation type instance happening per window of collected messages. The raw log from GELF will be used to build a set of log templates which will also be used for log event counting per window. The fields identifying the entities involved in a PII operation, such as the data processors and data subjects involved, when supplied if consent is given, will be used to identify the parts at risk or creating risk, when an anomalous pattern is detected in a message window.

3.5.2. Log Anomaly Detection Overview

In order to build a feature vector to feed the prediction models, we apply the generic framework used in log-based anomaly detection, which consists of four main steps [12] :

Log Collection: Systems constantly produce a stream of logs describing system states and runtime information. These logs contain a set of parameters depending on the used logging framework, and a raw text log message describing the event which resulted in the respective log. These logs can be stored for later use or fed directly into a stream analyser for real-time anomaly detection.

Log Parsing: Since logs consist of free form text, it is necessary to parse them and extract a group of structured event templates. The event templates consist of the most common combination of log parts/segments, which define the constant part of a log. Once a set of event templates is selected, each log can then be considered an

occurrence of one of the selected events, with added specific parameters, meaning that constitute the variable part of the log.

Feature Extraction: Once event templates are selected, we can then slice our log collection into several sequences of log events, which are then encoded into numerical feature vectors, which generally consist of a count of occurrences of a certain event within the sequence of log events being analysed. These vectors can then be passed as input to machine learning models. There are several ways to slice the logs, such as using fixed windows, sliding windows or other types of windows, depending on the domain and desired outcome. **Fixed and Sliding Windows** are based on time and timestamps. Each log has an associated timestamp corresponding to the time the log was issued. A **fixed window** is determined by a fixed time span or duration for each window. All logs occurring within a predefined duration (e.g., an hour or a day) are regarded as a sequence and analysed together. These windows are consecutive, the end of one window corresponding to the beginning of the next one. **Sliding windows** are also grouped by time, but in this case, we have also a sliding step, which is typically smaller than the window size. This step determines the movement of the window over time, as the beginning of a window corresponds to the beginning of the previous one, plus the step size or duration. In practice, this means that we can have windows with overlapping logs (i.e. if we have a 1-hour window with a 10 minutes step, we will have consecutive windows where the only new logs being added on each slide are the logs received in those new 10 minutes). Logs occurring within each window are here too regarded as a sequence.

Anomaly Detection: The last step consists of feeding the feature vectors created in the previous phase into machine learning models for training and construction of a model for anomaly prediction. The constructed model can then be used to determine if a new incoming log sequence is considered an anomaly or not.

A graphical example of this process can be seen in the following figure.

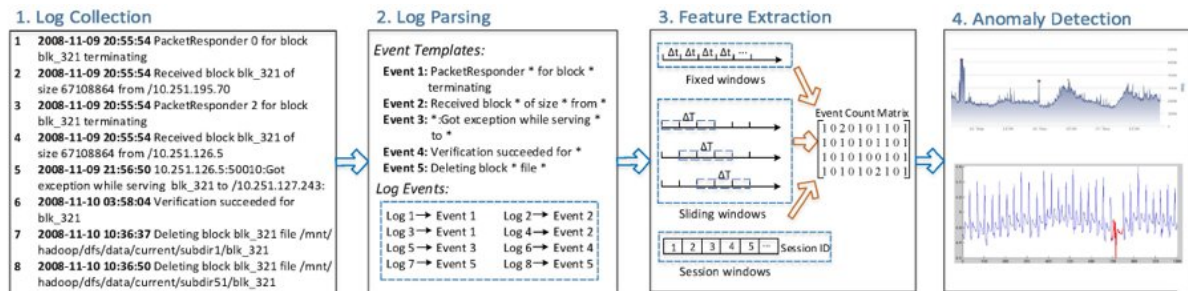


Figure 4 - Graphical example of log anomaly detection [12]

3.5.3. Log Anomaly Detection in PoSeID-on

The log anomaly detection in the RMM currently has only been implemented in the stream-layer and, as such, the following section describes the steps taken for that respective layer. Batch log anomaly detection will follow a similar structure, with the difference that the batch layer allows for more flexibility when performing feature extraction, by converting the incoming data to a vectorized format that anomaly detection libraries accept.

Log collection

Logs in PoSeID-on are delivered through the message bus directly to the Risk Management Module, using the established message protocol that, in turn, uses Protobuf [13] for the message format and Libsodium for encryption [14]. Since a message will be sent for each log using this message protocol, we can not only send a raw description of the log but also an extra set of parameters that can be useful for analysis. These parameters include the default parameters of Graylog Extended Log Format (GELF) [15], and any extra parameter which is also added to be stored in Graylog for later auditing. Extra fields can be sent to the RMM, which can enrich the analysis, such as PII regarding the data subjects and data processors involved in the operations logged using this message format, while keeping this information out of the general log storage, thus accounting for privacy issues. It is also important to keep in mind that the RMM collection of this extra PII is optional and subject to consent.

Log parsing

Log parsing, being the first step of the log anomaly detection pipeline, has an important impact on the result of the analysis. The quality of the structuring of the raw logs and the creation of the event templates directly affects the event count vectors which are fed to the anomaly detection algorithms, such as K-means or Principal Component Analysis. Moreover, good log parsing enables efficient search, filtering and grouping of logs.

Current solutions for automated log parsing have built-in parsing support for common log structures, such as Apache and Nginx logs. Since the PoSeID-on platform will have a collection of logs being generated by several third-party frameworks and tools, and also custom-built software, with unique logs, there is the necessity of manually configuring custom parsing with regex scripts, grok patterns or a parsing wizard. This does not scale particularly well and an automated solution is much more flexible. Also, since PoSeID-on is a novel platform working in a novel domain, adaptability to change is a must. As such, there was the need for analysing the current automated log parsing solutions in order to apply one to the Risk Management Module log parsing step.

Michael Lyu et al. have done an in-depth research [16] on 13 representative log parsers proposed in the literature, and have compiled their results as seen on the following table:

Log Parser	Year	Technique	Mode	Efficiency	Coverage	Preprocessing	Open Source	Industrial Use
SLCT	2003	Frequent pattern mining	Offline	High	✗	✗	✓	✗
AEL	2008	Heuristics	Offline	High	✓	✓	✗	✓
IPLoM	2012	Iterative partitioning	Offline	High	✓	✗	✗	✗
LKE	2009	Clustering	Offline	Low	✓	✓	✗	✓
LFA	2010	Frequent pattern mining	Offline	High	✓	✗	✗	✗
LogSig	2011	Clustering	Offline	Medium	✓	✗	✗	✗
SHISO	2013	Clustering	Online	High	✓	✗	✗	✗
LogCluster	2015	Frequent pattern mining	Offline	High	✗	✗	✓	✓
LenMa	2016	Clustering	Online	Medium	✓	✗	✓	✗
LogMine	2016	Clustering	Offline	Medium	✓	✓	✗	✓
Spell	2016	Longest common subsequence	Online	High	✓	✗	✗	✗
Drain	2017	Parsing tree	Online	High	✓	✓	✓	✗
MoLFI	2018	Evolutionary algorithms	Offline	Low	✓	✓	✓	✗

Table 4 - Summary of Automated Log Parsing Tools [16]

From this list, we are interested in the following key characteristics:

- **Mode:** Log parsers can be categorized into two main modes, **offline** and **online**.

Offline log parsers are a type of batch processing and require all log data to be available before parsing. On the other hand, **Online** log parsers allow processing of log messages one by one, as they arrive at the module, in a streaming manner. As such, **online** parsers are what we are looking for.

- **Efficiency:** This characteristic is of the utmost importance, considering the potentially large volume of logs being generated per second. As all of the subsequent analysis tasks depend on log parsing, it is necessary that the parser is efficient in order to perform real-time anomaly detection. As such, **High** efficiency is the value we should focus on.
- **Coverage:** Indicates the capability of a log parser to successfully parse all input log messages. If a checkmark is present, then the log parser is able to handle not only frequent patterns but also rare events, which is not always the case with log parsers. Ignoring rare events might result in missing important events, which are consequentially missed on the anomaly detection.
- **Open-source:** The availability of the source-code makes it much easier to adopt the parser, reusing it or even extending its implementation. As such, we only look for parsers available as open-source.

Considering **online** and **highly efficient** automated log parsing tools only, we have SHISO [17], Spell [18] and Drain [19]. Among these three, all offer optimal **coverage** but only Drain is open source. Coincidentally, Drain also achieves the best overall accuracy, has also been successfully implemented in a production system in collaboration with Huawei, and has been extended with Spark [20] for parallelization, exploiting Drain's data partitioning. This makes Drain an optimal candidate for log parsing in RMM. Drain log parsing approach uses a fixed depth parsing tree, which encodes specially designed rules for parsing. As input, Drain receives a batch of raw text logs and a simple description of the raw text formats, given during configuration, in order to extract components (i.e. the following string: "<Date> <Time> <Pid> <Level> <Component>: <Content>"). Optionally regular expressions can also be given in order to remove well-known parameters from the raw text, such as numbers, IP addresses or session ID's. For more details regarding the Drain Log Parser algorithm, the original paper can be consulted [19].

As Drain is only available in python, a java implementation of the algorithm was developed in order to facilitate the integration with the Risk Management Module pipeline and data analysis frameworks of Apache Spark. This implementation was also further extended in order to work with the Spark Streaming pipeline, in order to maintain log template consistency across batches.

The output of this log parsing step are the raw logs parsed into a dataset of structured fields, with an extra field for parameters. In addition, the list of log templates can also be extracted from the tree, as well as the ID's of the logs that fall into that log template.

Feature Extraction

Feature extraction in the RMM consists of taking the output of the parsing step, the event templates extracted from the parsing tree, and the logs associated with that template, and



creating a normalized vector corresponding to the count of occurrences of each template within that window. In addition to this, the count of each type of PII operation occurring in that window is also added to the vector and normalized.

For windowing, the current implementation of the stream layer has a fixed window of 5 secs, corresponding to the interval during which messages are fetched from the message bus. This is just for testing purposes, as such short windows usually do not offer enough insight into the system behaviour. We foresee the use of hourly sliding windows with a step of 3 to 5 minutes in order to be able to output results within a user session, or real-time. For the batch layer, options for window size and type are still under study. For both the stream and batch layers, the possibility of windowing by data processor is being considered in order to model data processor behaviour. Tuning and testing of these values are to be done for the final version of the RMM.

The current state of implementation does not yet allow for evaluating the feature extraction step, as the last steps of the pipeline, model training and anomaly detection, have not been implemented yet. Separation of the log occurrences from the PII operations count is also a possibility, creating two separate pipelines with different concerns, namely PII operation focused on anomaly detection and system logs anomaly detection. Whether separation of concerns will improve or not results will have to be evaluated, as well as normalization techniques.

Anomaly Detection

The anomaly detection step of the pipeline is currently under implementation, and MLlib [21] streaming K-means will be used for stream analysis. Models will be re-trained every hour based on the received logs. As well as for the batch layer, models will be generated before each batch analysis, for which daily runs are the first testing choice.

3.5.4. Data flow

The following figure demonstrates the flow of data, or log messages, within the module, to better understand the implementation and the anomaly detection pipeline.

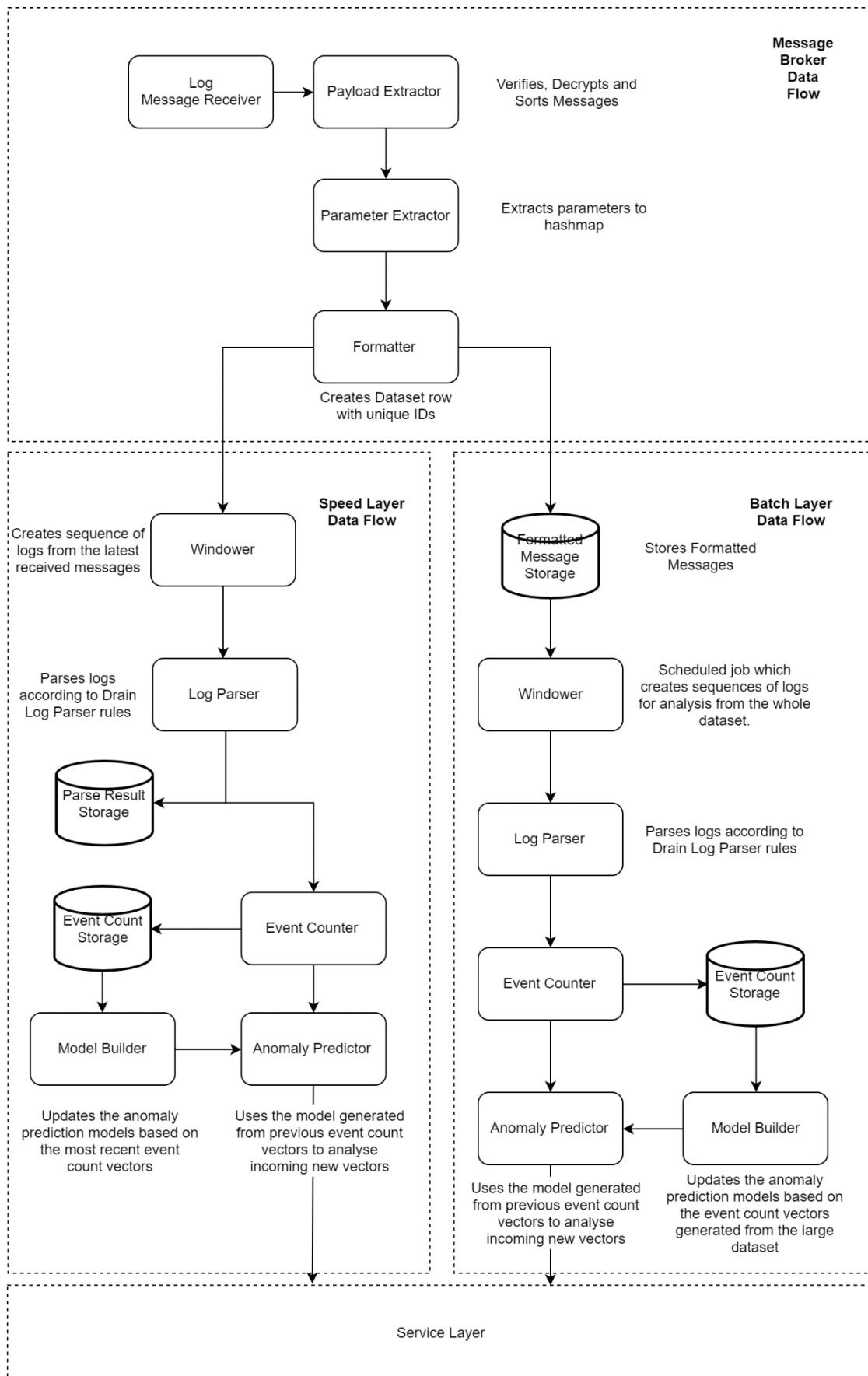


Figure 5 - RMM Data Flow

The service layer receives the results of the anomaly predictors, namely the anomalous sequences of logs and the involved data processors and data subject ID's, and stores them. The current version of the RMM does not yet receive and use these results, but they will be used to notify the involved parties of possible risks and to update data processors reputation.

3.5.5. Datasets

One of the challenges of the Risk Management Module is that there is no public dataset which focuses on the domain area of PoSeID-on that is, i.e. PII management. As such, we cannot test the anomaly detection approach with 100% certainty that it will work for detecting privacy risks until a real dataset is created and manually labelled (manually analysed and labelled on whether a sequence of logs corresponds to a PII privacy risk). Nevertheless, the log anomaly detection approach is quite generic, so it is possible to test for anomaly detection using labelled datasets, which present a similar structure to the ones that will be available in PoSeID-on. We have collected from the technical partners' examples of messages structures and log structures for the main PII operations for this purpose. For more specific feature extraction, such as counting the number of PII specific operations per window, the generation of a dataset with this type of parameters will have to be done or if possible, logs available after integration of the interim version modules will be used for testing.

Currently, the datasets available for testing are few but the LogPAI research group [22] has made available a collection of datasets from which some are labelled. One of them presents a similar structure to the one which will be used in PoSeID-on logs, namely:

- HDFS [23]
 - Hadoop Distributed File System log set is generated in a private cloud environment using benchmark workloads, and manually labelled through handcrafted rules to identify the anomalies. The logs are sliced into traces according to block ids. Then each trace associated with a specific block id is assigned a ground-truth label: normal/anomaly
- Time Span: 38.7 Hours
- Number of Messages: 11,175,629

3.6. Development

This section presents the frameworks and dependencies of the Risk Management Module, followed by the description of each implemented Java class, as well as foreseen but still to be implemented classes.

3.6.1. Dependencies and Used Frameworks

Dependencies and frameworks descriptions are organized by type, following the POM configuration file structure: first the plugins, afterwards the repositories, and afterwards libraries. Most of these dependencies are available in the general Maven repositories or custom repositories, that can be configured and imported directly using the POM of the RMM

Maven Project. The exceptions to this are the C implementation of Libsodium and the Spark-RabbitMQ connector, which will be detailed in the following subsections.

3.6.1.1. Plugins

Maven Compiler

Used to compile the sources of the project. The Java version used in the Risk Management Module must be Java 8, as the Apache Spark libraries do not support newer versions of Java yet.

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-compiler-plugin</artifactId>
  <configuration>
    <source>8</source>
    <target>8</target>
  </configuration>
  <version>3.8.0</version>
</plugin>
```

Maven Assembly

Used to create the uber-jar containing the RMM application and all necessary dependencies.

```
<plugin>
  <artifactId>maven-assembly-plugin</artifactId>
  <configuration>
    <archive>
      <manifest>
        <mainClass>eu.poseidon_h2020.rmm.RiskManagementModule</mainClass>
      </manifest>
    </archive>
    <descriptorRefs>
      <descriptorRef>jar-with-dependencies</descriptorRef>
    </descriptorRefs>
  </configuration>
  <executions>
    <execution>
      <id>make-assembly</id> <!-- this is used for inheritance merges -->
      <phase>package</phase> <!-- bind to the packaging phase -->
      <goals>
        <goal>single</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

3.6.1.2. Repositories

Lazysodium Java Repository

Necessary for importing the Lazysodium library.

```
<repository>
  <id>lazysodium-java</id>
  <url>https://dl.bintray.com/ter1/lazysodium-maven</url>
</repository>
```

3.6.1.3. Libraries

Unit Testing

Both dependencies refer to Junit necessary libraries for unit testing.

JUnit Jupiter Engine

```
<dependency>
  <groupId>org.junit.jupiter</groupId>
  <artifactId>junit-jupiter-engine</artifactId>
  <version>5.4.2</version>
  <scope>test</scope>
</dependency>
```

JUnit Jupiter API

```
<dependency>
  <groupId>org.junit.jupiter</groupId>
  <artifactId>junit-jupiter-api</artifactId>
  <version>5.4.2</version>
  <scope>test</scope>
</dependency>
```

Logging

All libraries under this section are used for logging purposes. The framework used for logging is Log4J2, and both Core and API dependencies are necessary for the framework to function.

Log4J2 Core

```
<dependency>
  <groupId>org.apache.logging.log4j</groupId>
  <artifactId>log4j-core</artifactId>
  <version>2.11.2</version>
</dependency>
```

Log4J2 API

```
<dependency>
  <groupId>org.apache.logging.log4j</groupId>
  <artifactId>log4j-api</artifactId>
  <version>2.11.2</version>
</dependency>
```

LogStash GELF

LogStash GELF is an appender for connecting the logger framework to the Graylog log manager, the PoSeID-on choice for log storage and management purposes.

```
<dependency>
  <groupId>biz.paluch.logging</groupId>
  <artifactId>logstash-gelf</artifactId>
  <version>1.13.0</version>
</dependency>
```

Messaging

The message protocol used in PoSeID-on makes use of the serialization mechanisms of Google Protobuf and is described using the associated **.proto** files. Currently, the **.proto** files are included in the resources folder of the project, but this will be changed to be imported directly from the PoSeID-on message protocol repository for consistency between modules.

Google Protobuf

```
<dependency>
  <groupId>com.google.protobuf</groupId>
  <artifactId>protobuf-java</artifactId>
  <version>3.8.0</version>
</dependency>
```

Lazysodium is a complete Java (JNA) wrapper over the Libsodium library that provides developers with a smooth and effortless cryptography experience. It is used to perform cryptographic operations on the PoSeID-on messages.

Lazysodium Java

```
<dependency>
  <groupId>com.goter1.lazycode</groupId>
  <artifactId>lazysodium-java</artifactId>
  <version>3.6.0</version>
</dependency>
```

Libsodium (Added as a resource)

Cryptographic library included as a resource which is used by the Lazysodium Java wrapper for Libsodium.

RabbitMQ AMQP Client

Dependency necessary to connect and operate with the RabbitMQ message bus. The version used is quite outdated due to the fact that the Spark Streaming to RabbitMQ connector used (and only available public implementation) has not been updated in over two years. More details regarding this will be given in the Spark-RabbitMQ connector dependency description.

```
<dependency> <!-- necessary for stratio rabbitmq connector-->
  <groupId>com.rabbitmq</groupId>
  <artifactId>amqp-client</artifactId>
  <version>3.3.4</version>
</dependency>
```

Data Processing

The following dependencies are necessary for the tasks of message pre-processing, processing and anomaly detection.

Spark SQL

Spark SQL is Apache Spark's module for working with structured data.

```
<dependency>
  <groupId>org.apache.spark</groupId>
  <artifactId>spark-sql_2.11</artifactId>
  <version>2.4.0</version>
</dependency>
```

Spark Streaming

Spark Streaming brings Apache Spark's language-integrated API to stream processing, letting you write streaming jobs the same way you write batch jobs.

```
<dependency>
  <groupId>org.apache.spark</groupId>
  <artifactId>spark-streaming_2.11</artifactId>
  <version>2.4.0</version>
  <scope>compile</scope>
</dependency>
```

RabbitMQ Spark Streaming Receiver (Locally installed JAR)

This dependency was a weighted decision. Rabbit MQ Spark Streaming Receiver library is a connector between the Spark Streaming framework and the RabbitMQ message bus. This library has not been updated in almost two years and is outdated in dependencies, not working out of the box with the latest Spark versions. Currently, there are no other implementations.

The choice was to either create a connector of our own for PoSeID-on or to fork the repository containing RabbitMQ Spark Streaming Receiver and try to update enough dependencies for it to work without breaking other necessary dependencies for the RMM.

Since implementing a new connector would take a significant amount of time, we tried to update the outdated library, with success. The compromise is having an outdated version of RabbitMQ AMQP Client dependency since this library has suffered significant changes in the last two years and updating the RabbitMQ Spark Streaming Receiver to accommodate newer versions would be of almost equal effort as creating a new connector altogether. Another compromise is having to install our custom jar of the receiver into our local maven repository in order to import the dependency and compile the RMM.

```
<dependency>
  <groupId>com.stratio.receiver</groupId>
  <artifactId>spark-rabbitmq</artifactId>
  <version>0.6.0-SNAPSHOT</version>
</dependency>
```

AKKA Actor

Necessary dependency for the RabbitMQ Streaming Receiver to work.

```
<dependency> <!-- necessary for stratio rabbitmq connector-->
  <groupId>com.typesafe.akka</groupId>
  <artifactId>akka-actor_2.11</artifactId>
  <version>2.5.3</version>
</dependency>
```

Spark MLlib

Machine learning library used in the anomaly detection pipeline for its clustering algorithms, implemented specifically to work with Spark and Spark Streaming.

```
<dependency>
  <groupId>org.apache.spark</groupId>
  <artifactId>spark-mllib_2.11</artifactId>
  <version>2.4.0</version>
  <scope>runtime</scope>
</dependency>
```

Miscellaneous

Project Lombok

Project Lombok is a java library that automatically plugs into an editor and build tools, which allows the reduction of boilerplate code by using annotations instead.

```
<dependency>
  <groupId>org.projectlombok</groupId>
  <artifactId>lombok</artifactId>
  <version>1.18.6</version>
  <scope>provided</scope>
</dependency>
```

3.6.2. Component Implementation

This section provides a description with the necessary level of detail to better understand the actual code and its methods. However, the actual code will not be listed in this section. Instead, the code will be available in the PoSelD-on repositories. It should be noted that this implementation is an interim version and work is still in progress. Therefore, some modules could have incomplete features and performance is not optimal. An overview of each implemented class will be given, including the functionalities expected to be implemented in current placeholder classes (classes without relevant code). These placeholder classes will be identified with “[Placeholder]” after the class name. Most implemented methods names are self-explanatory and, as such, only relevant details or steps will be explained in detail. For a better understanding of the implementation, it is recommended to take Figure 5 as reference, which describes the flow of data and operations in the RMM. Described classes are grouped by first level package only, in order to facilitate reading.

3.6.2.1. Base Package

BaseClass.java

Base class extended by all other classes, which setups the logging framework.

SparkClass.java

Base class extended by classes that leverage the Spark Core and Spark Streaming frameworks. It is used simply to reduce redundant code in extending classes. Stores the spark session and streaming context.

3.6.2.2. Batch Layer Package

Currently, all classes in this package are a placeholder, as the implementation of the batch layer has not started yet.

BatchAnalyser.java [Placeholder]

Will hold the code pertaining to the data processing done in the batch layer.

ModelBuilder.java [Placeholder]

Will hold the code pertaining to the build of the anomaly detection models.

MasterStorage.java [Placeholder]

Will hold the code pertaining to persistent storage management. It will manage connection and operations related to received message data storage.

BatchLayer.java [Placeholder]

Interface which describes the methods that must be implemented in the Batch Processor.

BatchProcessor.java [Placeholder]

Will hold the code pertaining to the logic and interaction of components in the batch layer.

3.6.2.3. Data Ingestion Package

Classes in this package focus on the reception and pre-processing of data received by the RMM.

MessageBroker.java

Implements the connection between the RMM and the Message Bus. Connection for sending messages (used to respond to RPC requests) is implemented in *createConnectionToMessageBus()*. Connection for receiving messages, in a streaming manner is set up leveraging the RabbitMQ Spark Streaming Receiver through the setup method *setSparkStreamingRabbitMQConsumer()*. In order to work, this class fetches from environment variables the following variables, used to connect to RabbitMQ (names are self-explanatory):

```
private static String QUEUE_NAME, QUEUE_HOST, QUEUE_USERNAME, QUEUE_PASSWORD;  
private static int QUEUE_PORT;
```

This class starts by creating a connection for writing to the Message Bus. Then it creates a Streaming Receiver which connects to the Message Bus for reading and creates a stream of message, usable by the Spark framework. Afterwards, received messages are decrypted and formatted into datasets, in order to pass a copy to both the Stream and Batch layers for further processing.

3.6.2.4. Drain Package

Drain.java

Java implementation of the original Drain algorithm. Leverages Spark SQL in order to perform the dataset operations which in the original Python implementation are done using the **pandas** library. For details on the Drain log parsing algorithm, the original paper can be consulted [19].

DrainHDFSDemo.java

Demo class for Drain, using HDFS logs, which implements the necessary configuration steps for the Drain algorithm to work. These include setting the input path for the log file, the output path, and the file name, as well as the log format and the regular expressions to be used during the log parsing.

DrainPoseidon.java

Wrapper class for the DrainStreaming class used in the RMM log parsing step. As the DrainHDFSDemo class, it holds the necessary configuration of the Drain algorithm, currently hardcoded but that can be implemented as environment variables later in the development. It also holds the data structures used to store the parsing tree and the log cluster list in order to build the feature vectors which are feed to the clustering algorithms.

DrainStreaming.java

Adapted implementation of the Drain algorithm to work with the RMM log analysis pipeline. The input to this algorithm is a sequence of log messages in JavaRDD<Row> format, the rootNode of the parsing tree, and the log clusters list. The output is a Dataset<Row> of the parsed log message sequence. The updated parsing tree and log cluster list are also updated with the new results.

3.6.2.5. Proto Package

Classes in this package are automatically generated by the Protobuf compiler, which must be run over the **.proto** files. For more details on these files, deliverable D5.1 (which focuses on the integration aspects of PoSelD-on) can be consulted.

3.6.2.6. Service Layer Package

Currently, all classes in this package are placeholders, as the implementation of the service layer of the RMM has not started yet.

MonitoringModule.java [Placeholder]

Will hold the code pertaining to the monitoring module, which will provide methods for results retrieval and visualization.

NotificationHandler.java [Placeholder]

Will hold the code pertaining to the notification of data subjects, data processors and PoSelD-on administrators, regarding privacy risks or corrections to previous alarms.

ReputationManager.java [Placeholder]

Will hold the code pertaining to operations regarding updates and retrievals of Data Processor's reputation.

ServingStorage.java [Placeholder]

Will hold the code pertaining to the persistent storage management of the service layer. It will manage connection and operations related to the storage of data processor reputation scores and anomaly detection results.

ServiceLayer.java [Placeholder]

Interface describing the methods that must be implemented by the RiskManager class.

RiskManager.java [Placeholder]

Will hold the code pertaining to the logic and interaction between components of the service layer.

3.6.2.7. Speed Layer Package

This package implements the stream layer of the RMM, which focuses on stream analysis and output results to the Risk Manager class of the serving layer.

`StreamAnalyser.java` [Placeholder]

Will hold the code respective to the data processing done in the stream layer. Currently, this code is in the `StreamProcessor` class but it will be moved to this class to allow the `StreamProcessor` class to serve as a manager of all components in the stream layer.

`StreamStorage.java` [Placeholder]

Currently, the data is stored in memory, in the RMM. This allowed for faster development and testing, but this is just a temporary solution, as it is much less performant. This class will hold the code pertaining to storage connection management and operations in the speed layer.

`SpeedLayer.java`

Interface describing the methods that must be implemented by the `StreamProcessor` class.

`StreamProcessor.java`

The implementation regarding data processing in the stream layer is currently done in this class but in the future this class will serve as a manager between components in the stream layer in order to allow separation of concerns between classes. Currently this class receives the formatted and structured log messages passed by the `MessageBroker` class and performs a series of operations on that data, which can be seen in Figure 5. First logs received are windowed according to `WINDOW_LENGTH` in seconds, which currently is 2 seconds, which corresponds to the batch interval, or the interval in which messages are retrieved from the Message Bus. Afterwards, these messages are parsed using the `DrainStreaming` algorithm. The output of that step is used to build an event count dataset, from which a vector is extracted for use in the clustering algorithms. At this stage of development clustering and anomaly detection is still not implemented.

3.6.2.8. Utils Package

General utility classes are contained within this package. These are the following:

`MessageGenerator.java`

Class used to generate messages according to the PoSeID-on message format from a dataset of logs. Each entry of the dataset is converted into a message and sent to the `MessageBus`. This allows us to test the anomaly detection pipeline with actual messages.

`MessageProtocol.java`

Implements all the operations related to the creation, verification, encryption and decryption of messages, according to the PoSeID-on message protocols.

`Pair.java`

Wrapper class used in the java implementation of the Drain log parsing algorithm in order to be able to return two resulting objects from a method.

RMMUtils.java

Implements miscellaneous methods for actions such as data type conversions, fetching environment variables and printing variables descriptions.

UncaughtExceptionHandler.java

Sets up a logger of Uncaught Exceptions using the Log4J2 logging framework.

3.6.3. Main Class

RiskManagementModule.java

This class is the main class of the Risk Management Module and it is the class that ties every component together. Firstly, it sets up Spark to run either in local cluster mode or connected to a Kubernetes cluster, according to the supplied environment variables. Details regarding environment variables used in RMM will be given in the configuration section.

Once a Spark Session is successfully generated, a Spark Streaming Context is generated from it, and both the Session and the Context are passed to new instances of the StreamProcessor and MessageBroker. An instance of the MessageGenerator is also created and ran in order to provide testing input for the RMM. After all classes are instantiated, both the MessageGenerator and the MessageBroker are run in order to start processing messages in the RMM.

3.7. Configuration and Deployment

In order to follow the 12-factor standards and the choice of Kubernetes, which is described in the integration deliverable D5.1, all configuration of the software will happen through environment variables, with the exception of regular expressions and log structure used by Drain log parser, since this is domain specific and will change very rarely between deployments. Currently environment variables are passed to the docker build tool but this will be changed to be supplied through the runtime environment in the future, since secrets passed this way can be accessible by inspecting the built containers. Deployment will be done through an OCI container image.

Configuration about how the application should be started is done through the Dockerfile which is part of the OCI container image. Deployment settings will be supplied using Kubernetes configuration files in the yaml format.

The Dockerfile also provided with the code takes care of all the steps necessary for building the OCI container image, including the installation of the custom JAR file containing the RabbitMQ Streaming Receiver which is also provided in the libs folder of the project.

A complete list of environment variables will be provided for the final version. The currently used environment variables are the following:

MAVEN_OPTS

Options for the maven compiler should be at least:

“-Xmx2g -XX:ReservedCodeCacheSize=512m”

KEYPAIR_SEED

Temporary seed to generate the RMM certificate. This is just for testing and debugging consistency as certificates will be loaded from files in the upcoming integrated versions of the RMM.

MESSAGE_QUEUE_PASSWORD

Password for the RabbitMQ queue connection.

MESSAGE_QUEUE_USERNAME

Username for the RabbitMQ queue connection.

MESSAGE_QUEUE_HOST

Host address for the RabbitMQ queue connection.

MESSAGE_QUEUE_PORT

Port for the RabbitMQ queue connection.

MESSAGE_QUEUE_NAME

Name of the RabbitMQ queue the RMM will connect to.

LOCAL_THREADS

Number of threads to be assigned to the local Spark cluster. Must be at least 2, one for the Spark Driver and another for a Spark Worker. If no value is given, the RMM assumes it should connect to a Kubernetes cluster and will make use of the Kubernetes environment variables.

KUBERNETES_HOST

Host address for the Kubernetes cluster connection. This is only used if the LOCAL_THREADS variable is not set.

KUBERNETES_PORT

Port for the Kubernetes cluster connection. This is only used if the LOCAL_THREADS variable is not set.

INPUT_PATH

Path to the log file used for testing purposes by the MessageGenerator.

FILE_NAME

Name of the log file used for testing purposes by the MessageGenerator.

3.8. Unit Testing and Validation

In the current phase of development, unit and integration testing and validation are very limited, as the focus is on having a working prototype as fast as possible for integration. Nevertheless, a few unit tests have been developed which do not, by all means, provide complete coverage of all operations. Only critical aspects of certain libraries were tested in order to guarantee the proper function of the tested classes. The implemented tests are the following:

MessageProtocolTest.java

Tests developed in order to verify the message protocol implementation. These comprise:

- Verification of signed payloads using correct and forged signatures
- Successful decryption of messages with the correct keys and failure otherwise
- Verification of signed and unsigned certificates

RegexTest.java

Tests developed in order to experiment with Java Regex. These tests were merely to aid in the implementation of the Drain log parsing algorithm in java.

SparkTest.java

Tests developed in order to experiment with Spark SQL. These tests were merely to aid in the implementation of the Drain log parsing algorithm in java.

After integration, proper unit and integration testing will be performed as part of Work Package 6 which focuses on the testing and evaluation of the PoSelD-on platform as a whole. For more information on the roadmap for pilot evaluation and testing, deliverable D6.1 [24] can be consulted.

4. Personal Data Analyser

This section starts by introducing the PDA's architecture and design. Subsequently, a description and comparison of NLP tools, data labelling schemes, and datasets is provided. The section also includes the results of a comparative study of the referred tools. Finally, development, deployment and validation details are provided.

4.1. Architecture and System Design

The Personal Data Analyser (PDA) is a software module developed to control personal data in a transaction. By resorting to state-of-the-art Natural Language Processing (NLP) tools and Artificial Intelligence (AI), it aims at discovering all previously non-identified personal data, such as personal data for which there is no data subject authorization.

The PDA interacts directly with the Message Bus – a shared message queueing service used within the PoSeID-on platform. As shown in Figure 6, through the Message Bus, the PDA establishes communication with three other modules: Dashboard, Data Processor API and the Risk Management Module (RMM). This allows the PDA to receive analysis requests and return analysis results whenever there is explicit consent from the data subjects. In addition, there is a one-way communication from the RMM module where information about Data Processors' reputation and incident history is provided.

As each message arriving at the PDA is serialized and dully encrypted, it needs to be handled by the Request Processor, which performs deserialization and decryption procedures. According to the received message, the contents are forwarded to the correspondent components: Metadata Extractor, Structured Data Processor or Permissions Analyser.

The information gathered at the previous components is forwarded to the NLR / NLU Processing Unit, which uses NLP mechanisms (described in the following sections) to process the information. The final stage is the assessment of privacy risk, using privacy metrics and the results derived from the NLP process. An answer is generated and sent back to the respective requesters, through the Message Bus.

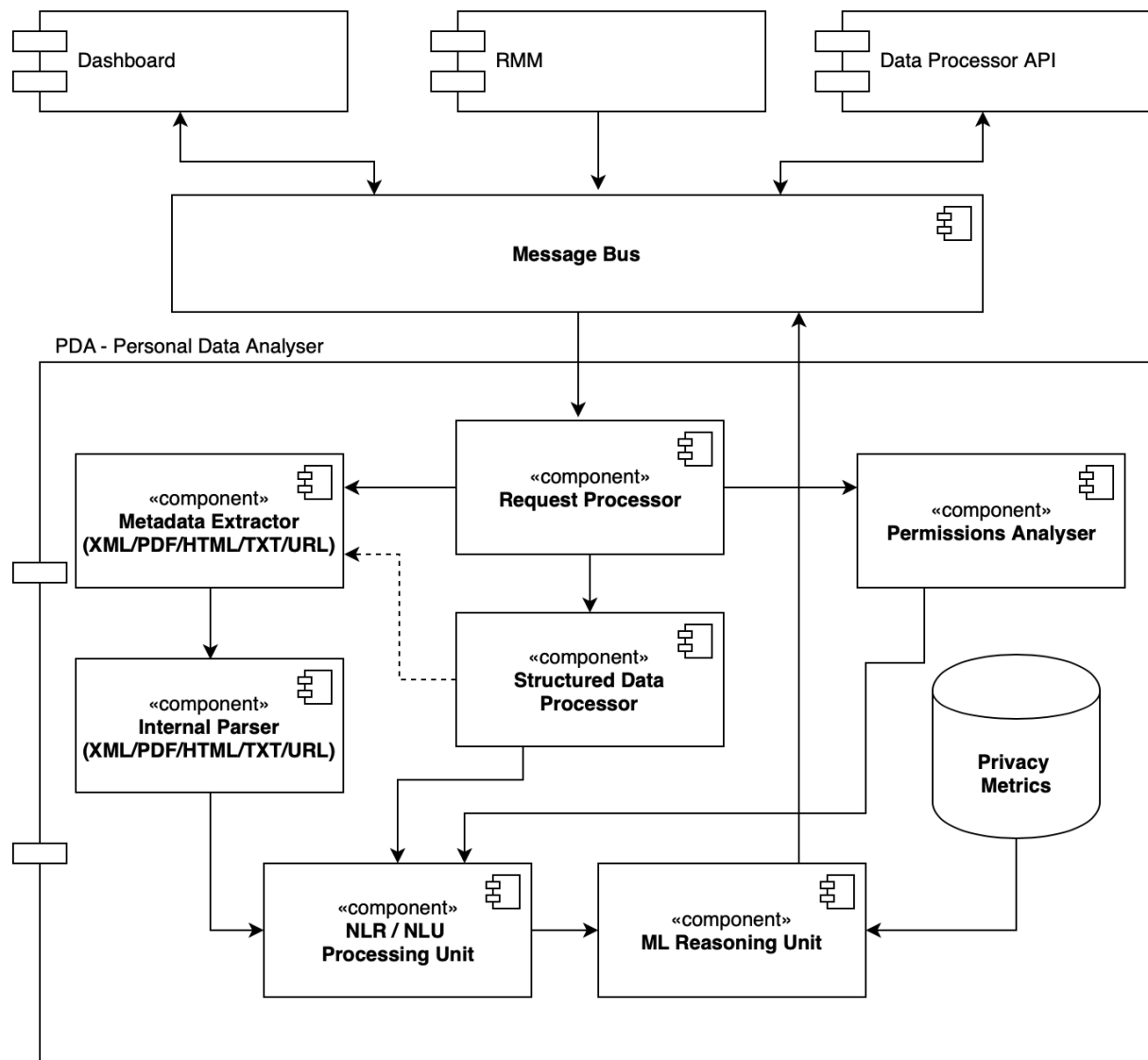


Figure 6 - Personal Data Analyser Architecture

Additional architectural information can be found in deliverable D2.2 – System Requirements and Architecture. The following sections provide technical information regarding the development of the PDA internal components.

4.2. Natural Language Processing (NLP)

Before the development phase, there was a preliminary stage that addressed the identification of the state-of-the-art, a set of experiments with different NLP tools, assessment of characteristics, and the gathering of different types of available datasets. The assessment of capabilities relied on the evaluation of performance, accuracy, model training options, labelling schemes, language support, and types of machine learning capabilities.

4.2.1. NLP Tools

Natural Language Processing tools are software libraries and applications that are used for extracting information from text in digital format and derive meaning from it. The analysis can be performed, for instance, of over semantics, syntax, or speech, with each one having its particular challenges. For example, most tools support tokenization, Part of Speech (POS) Tagging or lemmatization. Other features such as Named Entity Recognition (NER), translation, recognizing textual entailment or NLU are not as available in terms of quantity and quality. Nevertheless, in the following paragraphs, we provide some examples of options that are available and their main characteristics.

The Natural Language Toolkit (NLTK) [25] is one of the most well-known NLP tools. It is community-driven and open-source Python software, which allows the manipulation of different corpora, categorizing text or analysing linguistic structure (e.g., tokenization, POS tagging, or NER). The Stanford CoreNLP [26] tool stands out as a reference tool in the field of NLP. The tool is open-source, developed in Java and, among other features, it is capable of performing sentiment analysis, dependency parsing, or NER, for instance. In 2016, ExplosionAI introduced SpaCy [27] as the fastest NLP library in the world. The fact is that it is not only fast, but also performs well against similar tools and supports similar features [27].

There are other NLP tools such as: TextBlob [28] or Polyglot [29] developed in Python which got inspiration from NLTK; the General Architecture for Text Engineering (GATE) [30] developed in Java, which comprises several components for specific purposes such as HTML, XML or email processing; or Google's SyntaxNet [31], which uses the open-source TensorFlow [32] machine learning platform to provide an NLU toolkit.

Given the reputation from NLTK and Stanford CoreNLP, both in academic and research environment, we decided to compare the two with a recent and promising tool such as spaCy, which is also designed to operate in production environments. As such, Table 5 shows the high-level features that these three tools have to offer. The most relevant features for the PDA development are the possibility of updating the machine learning models, the types of Named Entities supported, and the available languages. For these relevant requirements, spaCy is the tool that performs the best, by allowing the retraining of models, having the number of named entities in their models, and by supporting the languages to be used in the PoSeId-on's pilots.

NLP Tools	Natural Language Toolkit - NLTK v3.4 (Python)	Stanford CoreNLP v3.9.2 (Java)	SpaCy v2 (Python)	Relevance for PDA requirements
Methods	Linear CRF's	Linear / Neural (CNN) not officially supported	Linear / Neural (CNN)	Low relevance
Can a model be re- trained/updated?	No	No	Yes	High relevance
Default Entities	ORG, PERSON, LOCATION, DATE, TIME, MONEY, PERCENT, GPE, FACILITY	ORG, PERSON, LOCATION, DATE, TIME, MONEY, PERCENT, NUMBER, ORDINAL, DURATION, SET, MISC	ORG, PERSON, LOC, DATE, TIME, MONEY, PERCENT, GPE, FAC, NORP, PRODUCT, EVENT, WORK_OF_ART, LAW, LANGUAGE, QUANTITY, ORDINAL, CARDINAL	High relevance

Model Training Scheme	IO, IOB	IO, IOB	BILOU	Low relevance
Default Languages Supported for POS Tagging and NER	English	English, German, Spanish, Chinese	English, German, Spanish, Portuguese, French, Italian, Dutch, (and Multi-language)	High relevance
Documentation	Book, articles, manuals, GitHub, Website	Articles, GitHub, FAQ's, Website	Articles, GitHub, Manuals, Website, Specific Tutorials	Moderate relevance

Table 5 - Comparison of NLP Tools General Features

On a more fine-grained analysis, Table 6 shows a comparison between specific NLP functionalities provided by each tool. Such features are common NLP tasks that can be used by the PDA for the interim version, and others can further complement it with additional development for the final version. It is possible to observe that Stanford CoreNLP natively supports all the listed features. Nevertheless, NLTK and spaCy can equally support them by using additional functionalities. For instance, spaCy can perform textual entailment by using Keras [33] or predict gender identification by using previously labelled dictionaries.

NLP Tools	Natural Language Toolkit - NLTK v3.4 (Python)	Stanford CoreNLP v3.9.2 (Java)	SpaCy v2 (Python)	Relevance for PDA requirements
Part of Speech Tagging	Yes	Yes	Yes	Low relevance
Lemmatizing	Yes	Yes	Yes	High relevance
Named Entity Recognition	Yes	Yes	Yes	High relevance
Textual Entailment	Yes	Yes	-	Low relevance
Dependency Parsing	-	Yes	Yes	Low relevance
Sentiment Analysis	Yes	Yes	-	Low relevance
NER Regexp	Yes	Yes	Yes	High relevance
Gender Identification	Yes	Yes	-	Moderate relevance

Table 6 - Comparison of NLP Tools Technical Features

To understand over which kind of data such tools operate, the following sections will introduce the labelling schemes used to train the NLP machine learning models and the types of datasets that are available to train such models.

4.2.2. Labelling Schemes

A labelling scheme is a notation (or encoding) used on training datasets. It is a crucial step for training the language models used in NLP tasks. It is intended to *teach* the software how to classify text. It is usually used for Part of Speech Tagging and Named Entity Recognition, in which the latter stands as an essential functionality of the Personal Data Analyser. The most common schemes are IO (Inside, Outside), IOB (Inside, Outside, Begin) and BILOU (Begin, Inside, Last, Outside, Unit). Intuitively, with more detail, better classification performance should be expected.

In fact, reference [34] shows that the BILOU scheme results are in improved performance. Given the characteristics of each labelling scheme, BILOU is the one offering one of the highest levels of detail. Of course, this applies to the group of languages we intend to use. For other languages, different schemes like IOB2 might perform better [35].

Scheme	IO	IOB	BILOU
“Simon is going to San Francisco”	Simon I-PER is O going O to O San I-LOC Francisco I-LOC	Simon B-PER is O going O to O San B-LOC Francisco I-LOC	Simon U-PER is O going O to O San B-LOC Francisco L-LOC
“Tim Cook presented Apple’s new iPhone 8 last Thursday.”	Tim I-PER Cook I-PER presented O Apple I-ORG new O iPhone I-PRODUCT 8 I-PRODUCT last O Thursday I-TIME	Tim B-PER Cook I-PER presented O Apple B-ORG new O iPhone B-PRODUCT 8 I-PRODUCT last O Thursday B-TIME	Tim B-PER Cook L-PER presented O Apple U-ORG new O iPhone B-PRODUCT 8 L-PRODUCT last O Thursday U-TIME
“Martin Luther King Jr. was born in Atlanta”	Martin I-PER Luther I-PER King I-PER Jr. I-PER was O born O in O Atlanta I-LOC	Martin B-PER Luther I-PER King I-PER Jr. I-PER was O born O in O Atlanta B-LOC	Martin B-PER Luther I-PER King I-PER Jr. L-PER was O born O in O Atlanta U-LOC

Table 7 - Labelling Schemes Examples

Table 7 shows the differences between the operation of IO, IOB and BILOU schemes. It is possible to see that the IO scheme holds basic information about an entity, and it is very limited, as it cannot represent two adjacent entities. IOB records not only the existence but

also the beginning of a multi-word entity. Finally, BILOU records the existence of an entity by its beginning word, its interior word and its last word, as well as the information about unitary entities.

4.2.3. Datasets

Some datasets are particularly useful for NLP-related tasks, and their usual denomination is corpus (i.e., a text collection). The corpora are labelled (i.e., tagged) on a variety of topics such as cultural, financial, political, scientific and others. The labelling is handmade and contains a tag for each and every word in the context of each sentence. For instance, “The little yellow cat” is represented as “The/DT little/JJ yellow/JJ cat/NN”, where DT stands for Determiner, JJ for Adjective, and NN for a Noun. Below, there are examples of such datasets and respective characteristics. Some datasets have more descriptive characteristics than others due to the publicly available information about each of them.

- **OntoNotes 5**
 - Release Date: 2013
 - Data Sources: telephone conversations (Tele), newswire (News), newsgroups, broadcast news (BN), broadcast conversation (BC), weblogs
 - Contents: News 625k, BN 200k, BC 200k, Web 300k, Tele 120k
 - Penn Treebank for syntax and the Penn PropBank for predicate-argument structure
 - Access: Registration and organization approval is necessary
 - Source: <https://catalog.ldc.upenn.edu/LDC2013T19>
 - NER
- **Reuters Corpus, Volume 1 (RCV1)**
 - Release Date: 2000
 - Data Sources: Reuters, English Language News stories
 - Contents: News 810k
 - Needs a signed form:
https://trec.nist.gov/data/reuters/ind_appl_reuters_v4.html
 - Source: <https://trec.nist.gov/data/reuters/reuters.html>
 - Publicly accessible, with licence
- **CoNLL 2003 (Conference on Computational Natural Language Learning)**
 - Manual annotation of a RCV1 subset
 - Publicly accessible
 - Source: <https://github.com/Franck-Dernoncourt/NeuroNER/tree/master/neuroner/data/conll2003/en>
- **The MUC 3 and MUC 4**
 - Publicly accessible
 - With NER
 - Contents: News reports
 - Topics: Terrorist activities in Latin America



- https://www-nlpir.nist.gov/related_projects/muc/muc_data/muc_data_index.html
- ACE 2002 (Automatic Content Extraction)
 - Contents: transcripts, newswire, and newspaper
 - Fee payment
 - With NER
- MUC 6 and 7 (Message Understanding Conference)
 - Contents: News reports
 - Topics: Negotiation of Labor Disputes and Corporate Management Succession; Airplane crashes, and Rocket/Missile Launches
 - Fee payment (Accessible to LDC members)
- The New York Times Annotated Corpus
 - Fee payment (Accessible to LDC members)
 - Contents: newswire – over 1.8 million articles
 - Topics: people, organisations and locations
- GMB (Groningen Meaning Bank)
 - Publicly accessible
 - With NER
 - <https://gmb.let.rug.nl/data.php>
 - PMB (Parallel Meaning Bank) is a derivation of GMB with more languages (<https://pmb.let.rug.nl/data.php>)
 -
- GMB-derived (developed by the community)
 - With NER
 - Publicly accessible
 - <https://www.kaggle.com/abhinavwalia95/entity-annotated-corpus#ner.csv>
- Enron Email Dataset
 - With NER
 - Publicly accessible
- English Gigaword
 - With NER
 - News
 - Fee payment
 - <https://catalog.ldc.upenn.edu/LDC2003T05>
- Brown

- No NER
- Publicly accessible

Despite the availability of several datasets, PoSeID-on is context-specific. Therefore, it is preferable to produce datasets containing PII-related information. This task is extremely difficult since PII is a sensitive type of data and it is not openly accessible.

To circumvent this issue, it is necessary to generate synthetic data that resembles real data as closely as possible. In this way, it is possible to train and teach the Personal Data Analyser to (correctly) identify PII (e.g., names, addresses, locations, events, etc.). There are other methods for identifying specific PII attributes, which are discussed further on.

Although most of the listed datasets feature entity annotation, many do not provide open access. As such, preference is given to datasets that are freely accessible and contain labelled entities.

We did not find a direct specification regarding gold-standard status of the datasets. The available information describes the datasets as mostly machine-annotated, requiring human intervention for correction.

Corpus	Named Entities	Availability	Size	Notes
OntoNotes 5 (Spacy)	YES - 18	Fee payment	1.445.000 tokens	No price shown; request submitted from UC organization; no answer so far
RCV1	NO	Publicly available, with licence agreement	810.000 news stories	Takes time to sign paperwork and wait for approval
CoNLL 2003 (NLTK, Stanford)	Yes - 5	Publicly available, with RCV1	301.418 tokens	Linked to RCV1 - time
MUC 3 / 4	YES	Publicly available	Information not available	Freely available, no information found regarding its characteristics
MUC 6 / 7 (Stanford)	YES	Fee payment	Information not available	-
ACE 2002	YES	Fee payment	Information not available	-
New York Times Corpus	YES	Fee payment	1,8 million articles	-
GMB	YES - 8	Publicly available	1,374,629 tokens	Available to download
GMB-derived	YES - 8	Publicly available	1.354.149 tokens	Available to download from Kaggle (updated by the community)
Enron Email	YES	Publicly available	200.399 messages	Available to download
English Gigawork	YES	Fee payment	1.756.504 tokens	-
Brown (NLTK)	NO	Publicly available	1.000.000 tokens	Available to download but no named entities

Table 8 – Corpus' Characteristics Summary

Listed in Table 8 is a summary of the previously described datasets. It is possible to observe that some fields are not completed due to the impossibility to find such information (e.g.,

number of entities available, or the overall size of the corpus). The first column (Corpus) identifies which corpora are used by the previously identified NLP tools. It is shown that spaCy uses trained OntoNotes5 models, which is one of the largest and most tagged corpora. Below, we present the details of experiments performed with the previously identified NLP tools and a publicly available dataset.

4.3. Comparative Study NLP Tools

In order to analyse and evaluate the characteristics of NLP tools, it is necessary to identify the number of true positives (TP), true negatives (TN), false positives (FP) and false negatives (FN). Therefore, the following metrics are applied:

- **Accuracy:** the ratio between the number of correct choices to the total number of choices. This metric should be applied when there is an even class distribution. Therefore, there might be cases where it is not provided, as it does not provide the desired insight.
- **Precision:** the ratio between the number of correct positive choices to the total number of positive choices.
- **Recall:** the ratio between the number of positive choices to the real number of choices. This means not only true positives but also false negatives (i.e., false negatives are positives).
- **F1 Score:** weighted average between precision and recall. Contrary to accuracy, this metric becomes more helpful when dealing with uneven class distributions, which is the expected case in the Personal Data Analyser. It is used in related work to evaluate the models built for different purposes.

The previously described metrics are formulated as shown in Table 9:

<i>Metric</i>	<i>Formula</i>
Accuracy	$(TP+TN)/(TP+FP+FN+TN)$
Precision	$TP/(TP+FP)$
Recall	$TP/(TP+FN)$
F1-Score	$2*(Recall * Precision) / (Recall + Precision)$

Table 9 - Metrics Formulation

Based on the information retrieved from the publishers of the tools, the scores provided with default models are very similar. This can be explained by the types of datasets used for model training, as well as the fine-tuning of the training process. In Table 10 the scores show that they have an average F1-Score of 0,85.

Tools	Natural Language Toolkit - NLTK v3.4 (Python)	Stanford CoreNLP v3.9.2 (Java)	SpaCy v2 (Python) - Neural		
	Linear	Linear	Small Model [36]	Medium Model [37]	Large Model [38]
Precision	0,86	-	0,84	0,85	0,85
Recall	0,83	-	0,85	0,86	0,86
F1 Score	0,86	0,86	0,85	0,85	0,85

Table 10 – Default Named Entity Recognition Scores from NLP Tools

There are also scientific publications which indicate better than average results. This is explained by the specificity of the models. Instead of generalizing the models, the authors use datasets that are context-specific, and thus perform very well on specific scenarios.

It is important to note that SpaCy provides a model that supports several languages. The supported languages are English, German, Spanish, French, Italian, Portuguese and Russian. The entities included in the model are Person, Location, Organization, MISC.

Finally, the scores are as follows:

- F1 - 78,55;
- Precision - 78,48;
- Recall - 78,62.

Additional local experiments have been performed. Since the GMB (Groningen Meaning Bank) dataset is openly distributed, has more than one million tokens, and eight classes of names entities, it was chosen for the experiments. From the total size, the dataset was partitioned into smaller sub-sets. Each subset was divided into two portions: 70% for training, and 30% for validation. It was possible to train and evaluate models with the three tools under consideration. The results obtained by each tool are presented below.

4.3.1. NLTK

The complete details such as code, datasets, and results of the NLTK experiments can be found at the GitHub's repository (e.g., GMB(Kaggle)09-04-2019/_SIZE_pc). In the following section, a summary of the experiments performed with NLTK is presented.

Dataset size (%)	Training Sentences	Validation Sentences	Precision	Recall	F1-Score
2,5%	839	359	0,4370	0,521	0,475
5%	1678	719	0,527	0,625	0,572
10%	3356	1438	0,555	0,647	0,597
20%	6713	2877	0,602	0,714	0,653
30%	10071	4316	0,589	0,696	0,638
40%	13428	5755	0,595	0,71	0,647
50%	16786	7193	0,606	0,719	0,658
60%	20143	8632	0,605	0,717	0,657

70%	23500	10071	0,609	0,719	0,659
80%	26857	11510	0,611	0,716	0,659
90%	30214	12949	0,614	0,72	0,663
100%	33572	14387	0,618	0,726	0,668

Table 11 - NLTK v3.4 NER Scores - GMB (Kaggle) Dataset

Table 11 shows the information on each training set and respective results used in the NLTK experiments. For each training set we present the number of sentences used to train the models, and with how many validation sentences was the model evaluated. Then we provide the respective Precision, Recall and, F1-Scores.

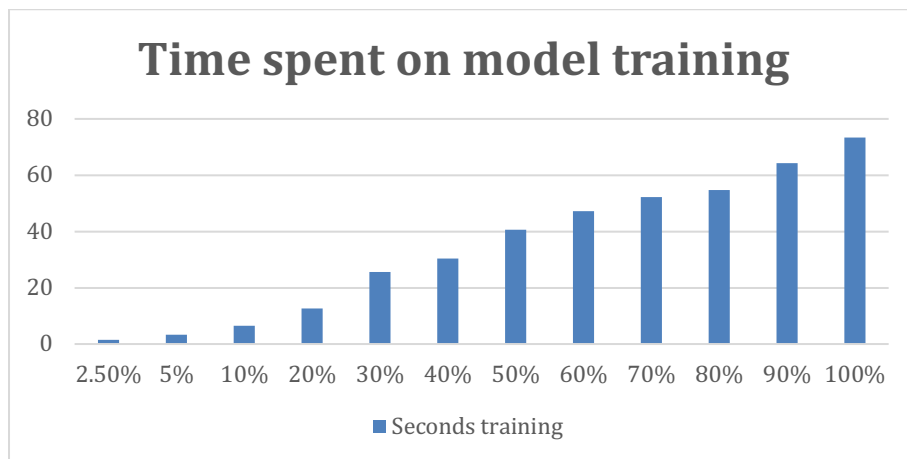


Figure 7 – NLTK time spent to train the model

In Figure 7, it can be seen a close to linear growth of the time necessary to train models. Starting with the smallest portion of the dataset (2,5% of the original size), the models takes around 2 seconds to train. On the other end, using the full-size dataset (100% of the original size), it takes almost 75 seconds to train the models. It is not possible to record the number of iterations as NLTK does not provide such information.

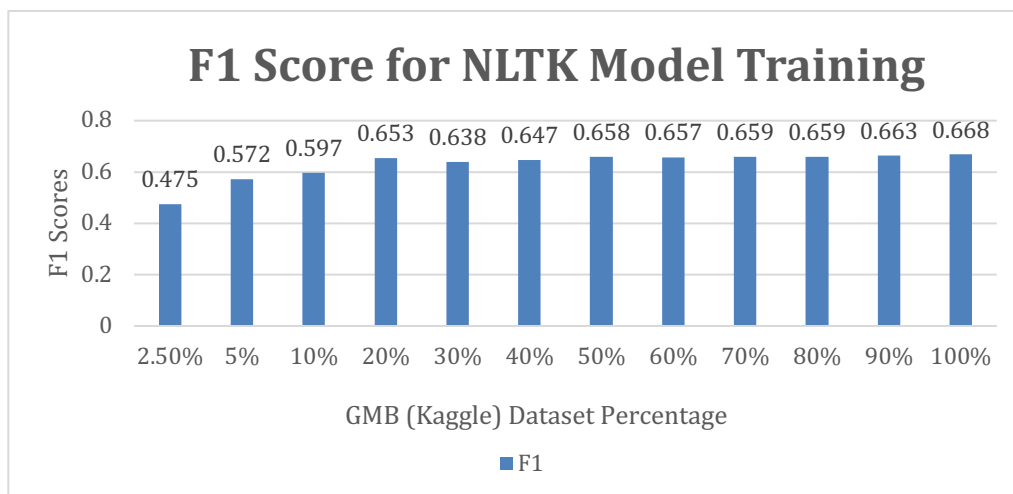


Figure 8 – NLTK F1 Scores

While for the model training time it was observed a close to linear growth, in Figure 8 it is noticeable a different trend in the F1 Score of the models. With an F1-Score of 0,47 with the smallest dataset (2,5%), there is a gradual growth (along with the dataset increase) until reaching an F1-Score of 0,65 with the 20% dataset. From that point on, there is only a small improvement with the increase of the dataset size. Reaching the best F1-Score (0,668) with the fully-sized dataset (100%).

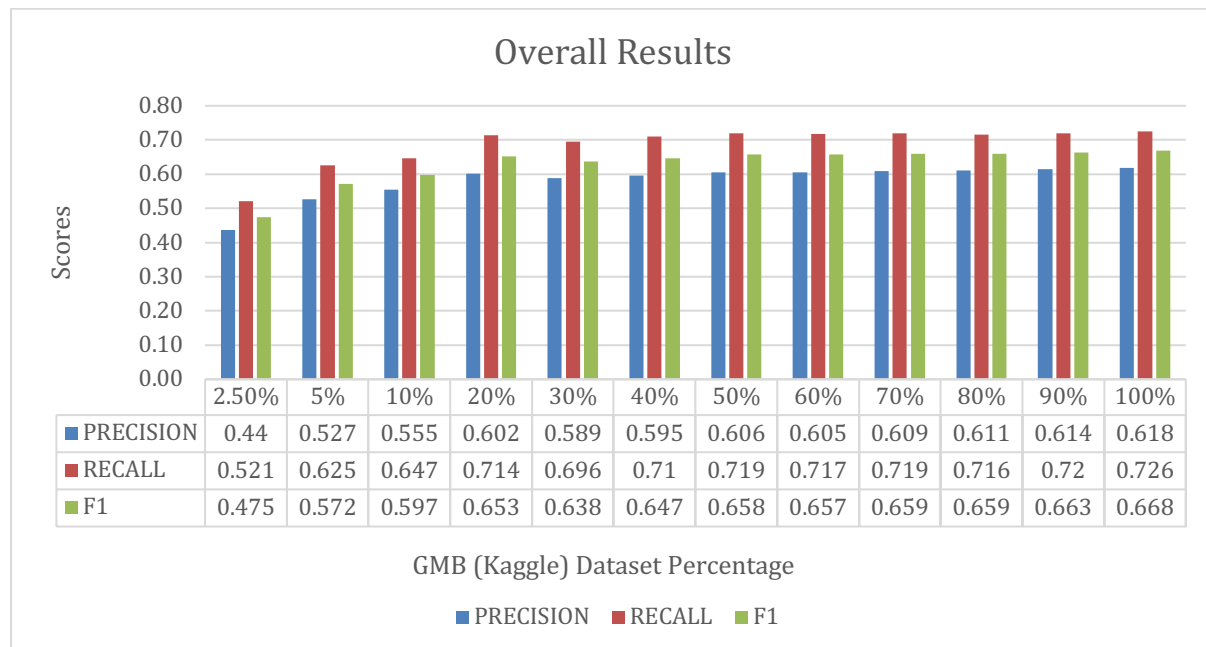


Figure 9 – NLTK overall results

Figure 9 depicts the overall results which include the precision and recall values obtained in the experiments. All show the same tendencies as the F1 Scores. The results obtained from the experiments performed with NLTK allow us to draw a few conclusions: (i) model training is very fast; (ii) the overall F1 Score is limited, with the maximum being 0,61 with the full-size dataset; (iii) with a relatively small dataset (20%) the results are similar to the ones obtained with a much larger dataset (100%).

4.3.2. Stanford Core NLP

The complete details such as code, datasets and results of the Stanford CoreNLP experiments can be found at the Git's repository (e.g., GMB(Kaggle)09-04-2019/_SIZE_pc). In the following section, a summary of the experiments performed with Stanford CoreNLP is presented.

Dataset size (%)	Training Sentences	Validation Sentences	Precision	Recall	F1-Score
2,5%	839	359	0,6819	0,6332	0,6566
5%	1678	719	0,7522	0,7195	0,7355
10%	3356	1438	0,7688	0,7453	0,7568
20%	6713	2877	0,8265	0,8047	0,8155

30%	10071	4316	0,8208	0,8062	0,8134
40%	13428	5755	0,8317	0,8175	0,8245
50%	16786	7193	0,8389	0,8274	0,8331
60%	20143	8632	0,8453	0,8290	0,8371
70%	23500	10071	0,8450	0,8268	0,8358
80%	26857	11510	0,8454	0,8248	0,8350
90%	30214	12949	0,8465	0,8299	0,8381
100%	33572	14387	0,8516	0,8360	0,8438

Table 12 - Stanford CoreNLP v3.9.2 NER Scores - GMB (Kaggle) Dataset

Table 12 shows the information on each training set and respective results used in the Stanford CoreNLP experiments. For each training set we present the number of sentences used to train the models, and with how many validation sentences was the model evaluated. Then we provide the respective Precision, Recall, and F1-Scores.

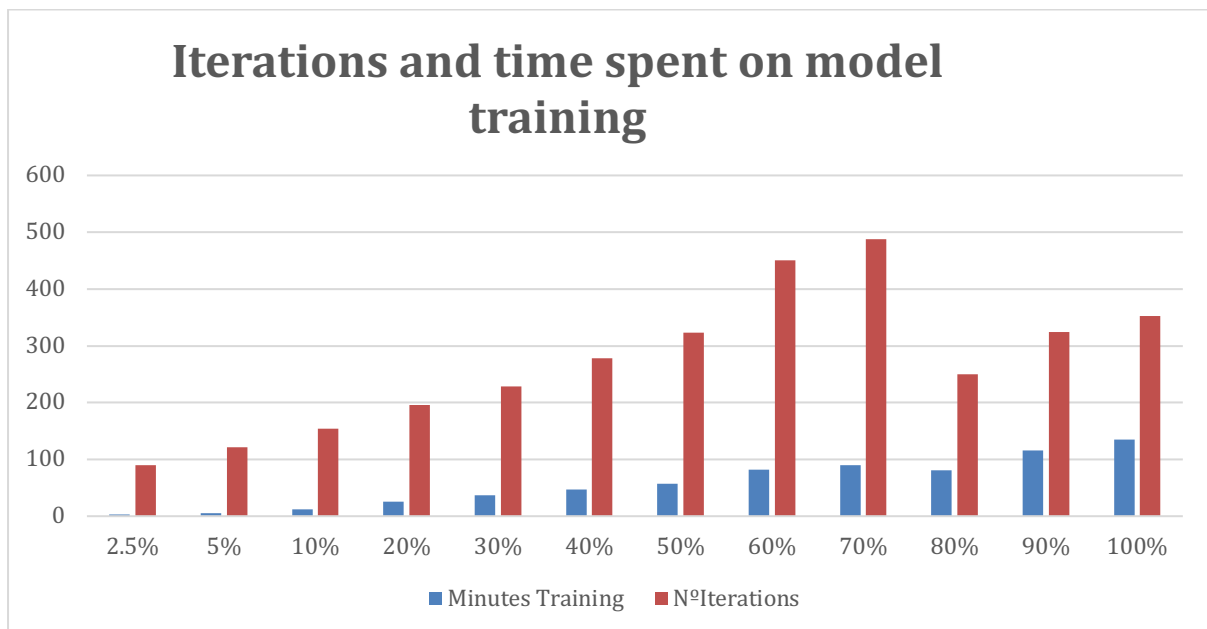


Figure 10 – Stanford CoreNLP time spent to train and number of iterations

In Figure 10, we observe a growth in training time, which contrasts with the previous tool, as it takes much longer to train similar models. In the fully-sized dataset, the training took around 120 minutes. The smaller partitions of the dataset train in less than 10 minutes.

The behaviour of the number of iterations necessary to train is quite unexpected. It was possible to observe a steady increase in the time spent on model training up to 70% of the dataset size. From that point on, we observe a decrease in the number of iterations necessary to train the models. The explanation for this behaviour lies in the limitation we have imposed on the number of iterations. Since it can be very demanding in terms of time and memory, we limited the number of iterations for Stanford CoreNLP (and spaCy) to a maximum of 500.

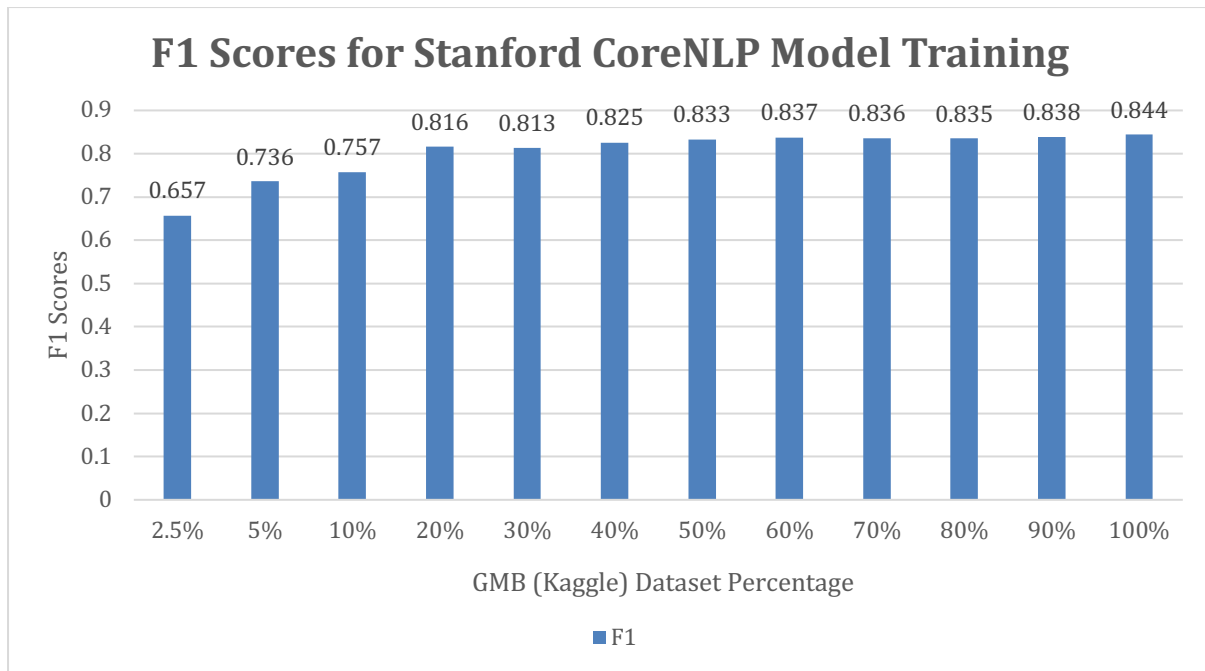


Figure 11 – Stanford CoreNLP F1 Scores

The F1-Scores obtained from the experiments show an improvement in comparison with NLTK. In Figure 11, it is possible to note the increasing tendency up to the 20% size dataset and stabilizing (with very small variations) after that point. From the 20% size to the fully-sized dataset, there is only a slight improvement of almost 0,03 in the score.

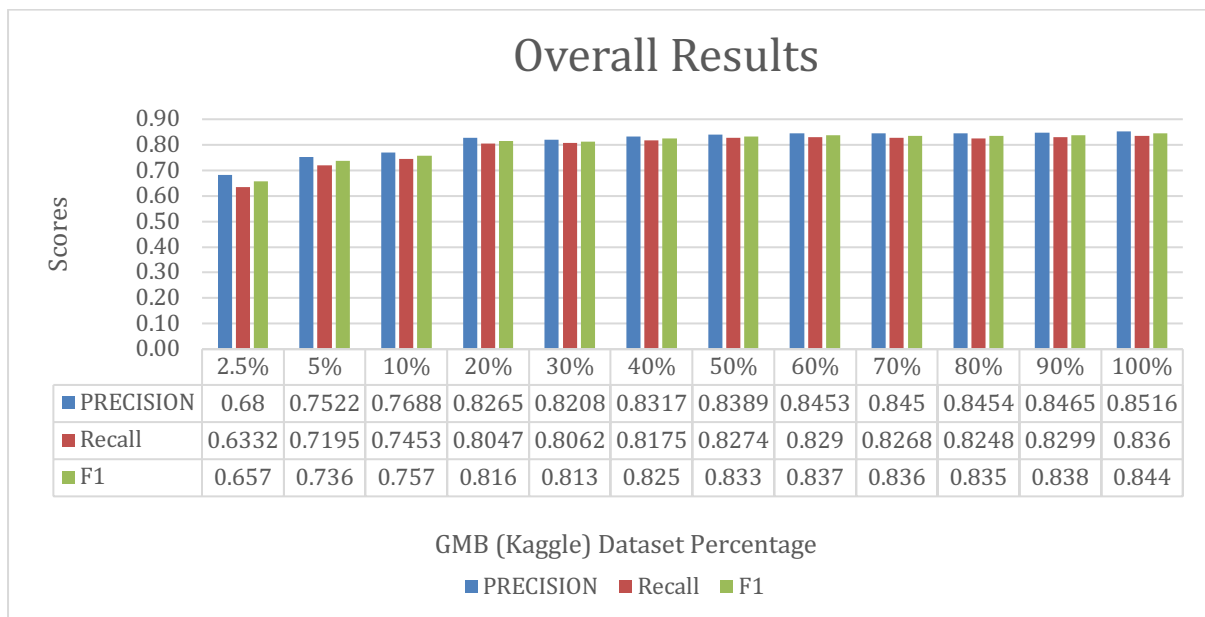


Figure 12 – Stanford CoreNLP overall results

Figure 12 depicts the overall results which include the precision and recall values obtained in the experiments. All of them show the same tendencies as the F1 Scores. The results obtained from the experiments performed with Stanford CoreNLP allow us to draw a few conclusions: (i) model training is still fast, even though the longer takes over 120 minutes; (ii) the overall F1 Score is getting good values of about 0.8, with the maximum being 0,84 with the full-size

dataset; (iii) with a relatively small dataset (20%) the results are similar to the ones obtained with a much larger dataset (100%).

4.3.3. Spacy

The complete details such as code, datasets and results of the spaCy experiments can be found at the Git's repository (e.g., GMB(Kaggle)09-04-2019/_SIZE_pc). In the following section, a summary of the experiments performed with spaCy is presented.

<i>Dataset size (%)</i>	<i>Training Sentences</i>	<i>Validation Sentences</i>	<i>Precision</i>	<i>Recall</i>	<i>F1-Score</i>
2,5%	839	359	0,6455	0,6440	0,6448
5%	1678	719	0,7226	0,7333	0,7428
10%	3356	1438	0,7810	0,7723	0,7766
20%	6713	2877	0,8045	0,8170	0,8107
30%	10071	4316	0,8095	0,8240	0,8160
40%	13428	5755	0,8320	0,8349	0,8335
50%	16786	7193	0,8461	0,8499	0,8480
60%	20143	8632	0,8454	0,8460	0,8457
70%	23500	10071	0,8530	0,8596	0,8566
80%	26857	11510	0,8527	0,8591	0,8559
90%	30214	12949	0,8530	0,8603	0,8570
100%	33572	14387	0,8600	0,8700	0,8653

Table 13 - spaCy v2 NER Scores - GMB (Kaggle) Dataset

Table 13 shows the information on each training set and respective results used in the spaCy experiments. For each training set we present the number of sentences used to train the models, and with how many validation sentences was the model evaluated. Then we provide the respective Precision, Recall, and F1-Scores.

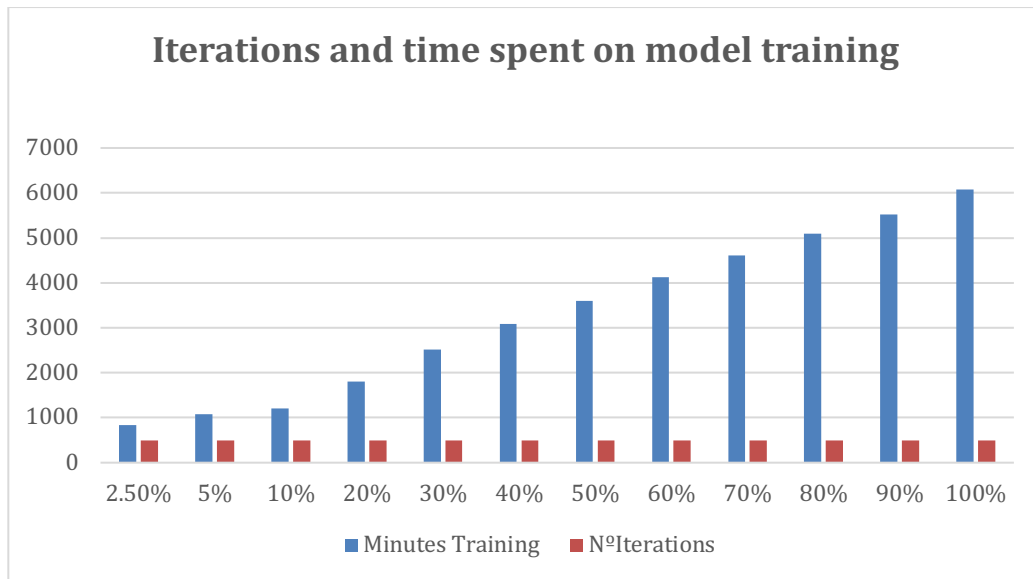


Figure 13 – spaCy time spent to train and number of iterations

In Figure 13, we observe a much higher growth in training time in relation to the previous tools, as it takes up to 6000 minutes to train similar models. Even the smaller partitions of the dataset train in about 1000 minutes. As explained before, we have limited the number of training iterations to a maximum of 500. In fact, all the training sessions ran up to the maximum number of iterations.

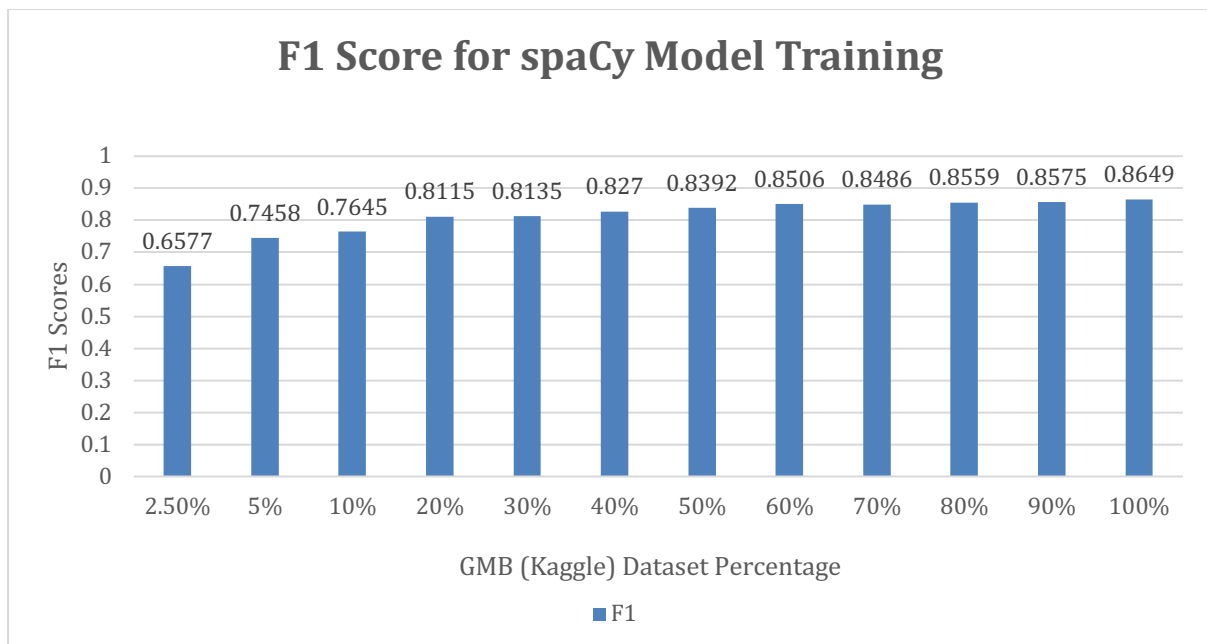


Figure 14 – spaCy F1 Scores

The F1-Scores obtained from the experiments show little improvement in comparison with Stanford CoreNLP, and a large improvement regarding NLTK. As it happened with the two previous tools, in Figure 14 we note the same increasing tendency up to the 20% size dataset and stabilizing (again with very small variations) after that point. From the 20% size to the fully-sized dataset, there is an improvement of only 0,05 in the score.

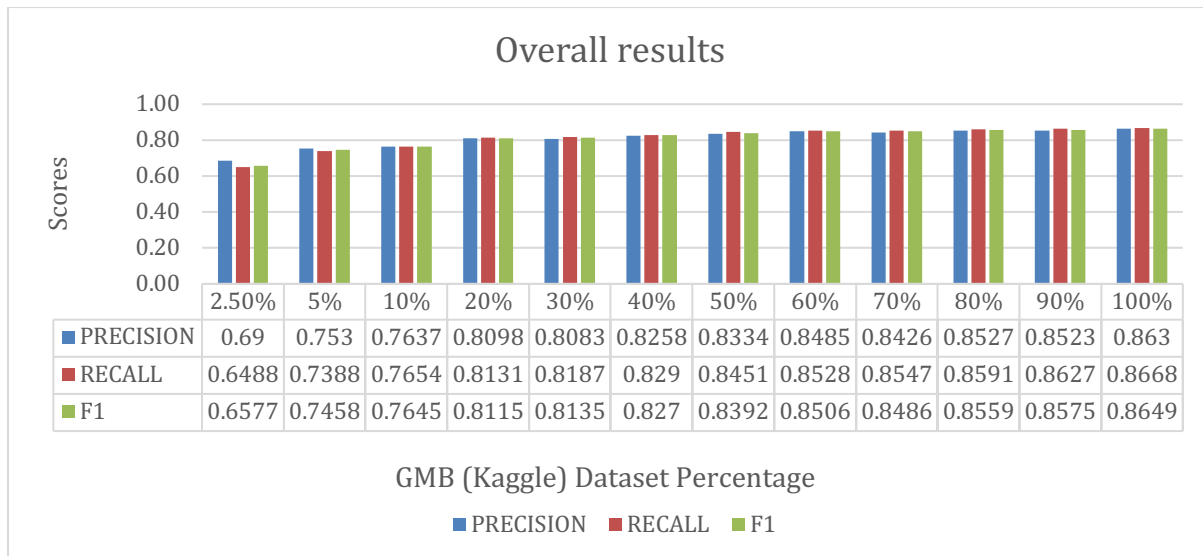


Figure 15 – spaCy overall results

Figure 15 depicts the overall results, which include the precision and recall values obtained in the experiments. All of them show the same tendencies as the F1 Scores. The results obtained from the experiments performed with spaCy allow us to draw the following conclusions: (i) model training under such conditions is taking too much time; (ii) the overall F1 Score is getting good values of about 0.8, with the maximum being 0.86 with the full-size dataset; (iii) with a relatively small dataset (20%) the results are similar to the ones obtained with a much larger dataset (100%).

Due to the fact that the training was taking too long, we decided to run a different set of experiments with spaCy. However, this time, we did not set the number of iterations to 500. For this experiment, we randomly defined the number of iterations.

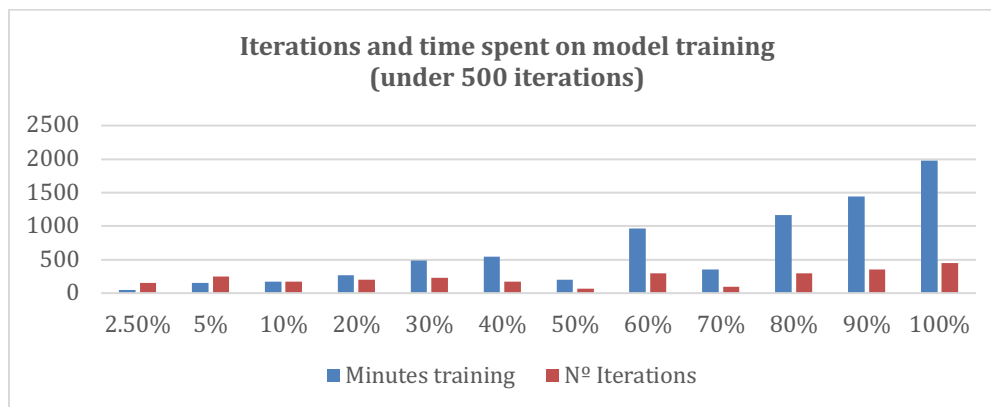


Figure 16 - spaCy time spent to train and number of iterations (under 500)

As we can see in Figure 16, by setting the iteration number to under 500, the longest training session took 2000 minutes. Approximately three times faster than with 500 iterations. In this case, as the numbers were randomly chosen, the distribution of training times is not linear anymore.

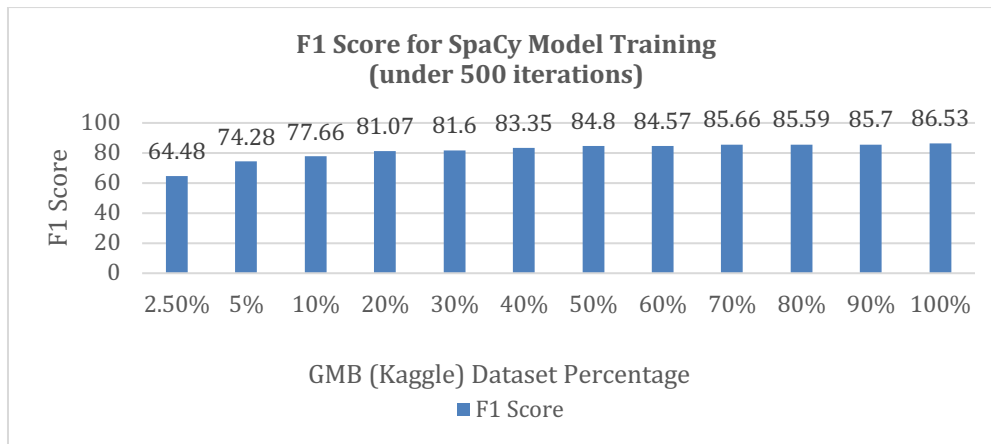


Figure 17 - spaCy F1 Scores (under 500 iterations)

The F1-Scores obtained from this experiment show almost the same results as the ones setting the iteration number to 500. We can see in Figure 17 that the F1-Scores are very similar to the ones shown in Figure 14. This means that we can still achieve quality results while spending less time training. According to spaCy's documentation, developers should experiment with different parameters and fine-tune the model in order to provide the best results.

4.3.4. Summary

After having analysed all the obtained results, it was possible to draw a few overall conclusions regarding the three analysed tools. As Figure 18 shows, in our experiments, the worst performance was that of NLTK. On the other hand, although Stanford CoreNLP and spaCy achieved similar results, the best performance was seen in spaCy with a small margin. The results indicate that without any comprehensive tuning of the model training settings, spaCy provides the best results, while it still leaving room for possible improvement if specific configuration is applied (e.g., shorter training times by setting fewer iterations).

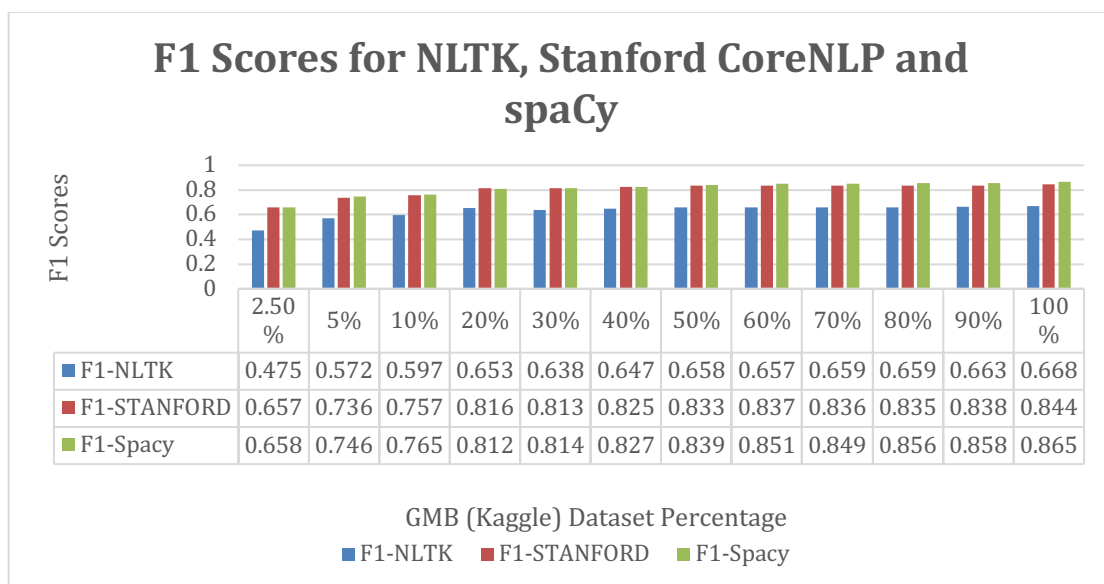


Figure 18 - NLTK, Stanford CoreNLP and spaCy F1-Score Overall Results Comparison

On top of these results (and others like training time or iterations) are the characteristics of the NLP tools. We have seen that the features that are important for the PDA module, like the possibility of re-training models, having a high number of default entities or supporting a high number of languages, are present in spaCy. Moreover, spaCy's development is in Python, which matches with the development language of the Personal Data Analyser module.

4.4. Development

This section starts by indicating and describing the intended usage of all the external libraries and packages (i.e., dependencies) currently used by the Personal Data Analyser. Then, it is followed by the implementation details and respective technical aspects to consider.

4.4.1. Dependencies

PyNaCl is a Python binding to Libsodium. There are many other Python bindings. However, the choice for PyNaCl is based on its usability, security, and speed. It supports digital signatures, secret-key encryption, public-key encryption, hashing and message authentication, password-based key derivation, and password hashing. The usage of this library allows the PDA to comply with the specification defined for the communications within the PoSeID-on platform (through the Message Bus) and assure that all messages are signed with a module-specific ED25519 key and encrypted using the respective Curve25519.

```
1. PyNaCl
2. V1.3.0
3.
4. from nacl.public import PublicKey, PrivateKey, SealedBox
5. import nacl.utils
6. import nacl.encoding
7. import nacl.signing
```

CommonRegex is an open-source library that provides off-the-shelf regular expressions that find specific types of personally identifiable information such as dates, times, phone numbers, links, emails, ips, ipv6s, prices, hex_colors, credit_cards, btc_addresses (bitcoin addresses), street_addresses. These fields have a predetermined format and syntax. Nevertheless, they are used by the PDA to find whichever fields fit such formats.

```
1. commonregex
2. V1.5.4
3.
4. from commonregex import CommonRegex
```

PyPDF2 is a Python library that supports the handling of PDF files and the extraction of the respective information. It is used to load all the contents of a PDF file and to extract relevant metadata from the file container such as the number of pages (relevant for internal parsing), author, owner, date of creation, and other fields.

```
1. PyPDF2
2. V1.26.0
3.
4. from PyPDF2 import PdfFileReader
```

Protobuf 3 is a protocol buffer which serializes all the data flowing within the PoSeID-on platform for inter-module communication. As such, it was necessary to have the protocol format agreed by all partners (details are available in deliverable D5.1 – Integration). Then, it was necessary to compile the files containing the protocol format and to generate the Python code which acts as a protocol API. The necessary code developed to access and interact with the API is described in the next section (4.4.2 - Component Implementation).

```
1. Protobuf
2. V3.7.1
3.
```

Pika is a Python library of the AMQP 0-9-1 protocol that allows the interaction of the PDA with the RabbitMQ service running (i.e., the Message Bus). RabbitMQ is a queueing protocol that allows its connected services to publish and subscribe to streams of messages, which are similar to message queues; to store streams of records in a fault-tolerant and persistent way, and to process streams of records as they occur.

```
1. pika
2. V1.0.1
3.
4. import pika
```

NLTK (Natural Language Tool Kit) is a Python library for Natural Language Processing. It supports several operations related to text processing such as tokenization, POS tagging, lemmatization or named entity recognition.

In addition, this library also includes a bind to the Stanford CoreNLP library. In this way, it is possible to use the functionalities of Stanford CoreNLP in a Python environment, instead of a Java environment. Nevertheless, it is still necessary to load the jar files and classifiers for POS tagging and NER. Respectively, they are *english-bidirectional-distsim.tagger* and *english.all.3class.distsim.crf.ser.gz*. Finally, TextCat categorizes text using n-grams (i.e., bi-grams in this case) and it allows the functionality of language identification.

```
1. nltk
2. V3.4.1
3.
4. from nltk.tag import StanfordNERTagger
5. from nltk.tag import StanfordPOSTagger
6. from nltk.classify.textcat import TextCat
```

spaCy is a Python tool for Natural Language Processing. Similar to NLTK, spaCy supports most operations related with text processing. However, internally the mechanism it is different. While NLTK or Standard CoreNLP rely on linear methodologies like CRF's, spaCy is using CNN's and works vectors for its text processor methods. This means that it not only provides good performance scores (e.g., F1 Scores in named entity recognition), but also uses state of the art NLP mechanisms. The PDA module relies on spaCy with its large model `en-core-web-lg 2.1.0` to perform the necessary operations.

```
1. spacy
2. V2.1.4
```



Other miscellaneous libraries (listed next) are used for different actions. For instance, accessing environment variables, iterate through data structures, perform logging, or reading from CSV files.

```
import os, logging, csv, sys, itertools, timeit
```

The choice for the package installer was *pip*, v19.1. Using Python Version 3. The command to run pip could be mapped to pip3 in order to install the packages on Python V3. All the previous libraries are installed using pip upon setting up of the container image. Then, upon container deployment everything is ready and available with no need for further installation.

4.4.2. Component Implementation

The implementation process of the Personal Data Analyser follows the previously described architecture (Figure 6). Given the mostly modular structure of the PDA, it is possible to develop independent functions and algorithms for each one of its components.

In the current section, we provide a description with the level of detail necessary to better understand the actual code and its methods. However, the actual code will not be listed in this section. Instead, the code is available at the PoSeID-on's code repository (Bitbucket). It should be noted that this is an interim version and work is still in progress.

The PDA receives incoming communication from three modules: Dashboard, RMM and Data Processor API. Therefore, one of the first components of the PDA architecture to be executed is the "Request Processor". Following, there is a description of the PDA components main functions and main purpose.

1) Request Processor

This component is continuously listening to the message bus and waiting for requests. In order to do so, it is necessary to follow a series of steps, described next.

a) Settings

The first step is to dynamically load setting from local environment variables: *MESSAGE_BUS_QUEUE_ADDRESS*, *MESSAGE_BUS_QUEUE_PORT*, *MESSAGE_BUS_QUEUE_NAME*, *MESSAGE_BUS_USERNAME*, *MESSAGE_BUS_PASSWORD*. Such variables allow the PDA listen to its own queue and handle requests.

b) Cryptography

In order to handle the encryption and decryption, the functions *create_sealed_box(key)*, *encrypt_message(message)* and *decrypt_message(unseal_box,encrypted_message)* rely on the methods from the PyNaCl library and perform the necessary operations.

c) To Read from message bus

Depending on the methods from the pika library (described in the previous section), the PDA is able to read and listen to messages from the message bus. The function *listen_to_bus(address, port, queue)* contains a callback function which continuously listens to the message bus and gets all the incoming messages. Once a message is received, the function which loads the contents is called.

d) To load Received Requests

Every message arriving to the queue is a request that needs to be properly handled. As such, the function *read_incoming_message(message, pda_secret_key, verification_key)* starts by deserialising the message using a Protobuf specific function: *ParseFromString(message)*. As described before, all the protobuf methods were previously compiled from the proto specification files. Then, a sequence of steps is performed to extract and validate information:

- i) Verify the PDA is in fact the recipient of the message: compare received recipient key with previously owned key;
- ii) Unseal the received sealed box with the body of the message recurring to the function and deserialise the contents;
- iii) Verify if the body of the message is signed by who claims to be the sender: compare the received sender key with the body signature;
- iv) Verify if the sender's signature is valid;
- v) Finally, pre-defined dictionaries are loaded with the respective message information, either it is a direct message or an *rpc_payload*.

e) To create Response Message

Every incoming request requires a response. Therefore, the functions *create_outgoing_direct_message(destination_pub_sign_key, *args)* and *create_outgoing_rpc_message(destination_pub_sign_key, *args)* load all the respective fields of a message (*direct_message* or *rpc_message*). All the steps defined in the communication protocol are executed and an encrypted and serialised response becomes ready to be sent. It is similar to the reverse procedure described in d).

f) To write to message bus

In order to answer all incoming requests, the PDA needs to properly generate response messages and send them to the message bus. For that reason, the function *send_to_bus(address, port, queue, message)* is in charge of sending back a response to whichever other module that has made a request. Again, this function relies on methods from the *pika* library to be able to connect to the message bus.

2) Metadata Extractor

To extract relevant metadata (mainly from PDF files) the PDA uses the class *PdfFileReader* from the PyPDF2 library. First it is necessary to set all the fields that might be filled using the function *metadata_types()*. The next step is to retrieve fields such as author, title, creator, date, or subject. The extracted information is sent to the internal parser. In the architecture there is a description of URL and HTML metadata extraction. However, it will likely be dismissed for now due to the lack of internet connectivity from within the modules.

3) Permissions Analyser

The permissions refer to which Data Processor can access specific attributes of Personal Identifiable Information (PII). This component has the task of examining the permissions associated with each incoming request. Since the permissions are expected to have a description of its finality, they can also be forwarded to the NLR/NLU component for

identification and validation of particular purpose and attribute associated. At this stage of development (interim version), the permission's analyser is still under development and not ready to be released. It will be delivered for the final version.

4) Structured Data Processor

The designed task of this component was to retrieve all the necessary fields contained in the organised data structure sent by the Request Processor component. Such fields comprise key/value pares that can have different data types such as string, numbers or files. However, as development advanced, the functionalities of this component were merged with the request processor component due to the type data extraction already performed upon the handling requests. Therefore, as of the interim version, the Request Processor and Structured Data Processor are merged.

5) Internal Parser

The Internal Parser component extracts the contents from the various file types supported by the PDA. In conjunction with the previously extracted metadata it builds internal data structures with all the information. Firstly, it is necessary to identify the type of files being analysed using the function *get_file_type(file_input)* which returns the respective name and extension. According to the file type being processed, the function *file_selector(path, filetype)* calls the respective parsers. In case of PDF's, the files are open, checked if they are encrypted with *check_encryption(pdf)* and the contents are extracted recurring to *PdfFileReader* from the PyPDF2 library. In the case of TXT or CSV files, the method is similar as it recurs to Python native methods to open and load the contents to internal data structures. Such information is then ready to be sent to the NLR / NLU Processing Unit for analysis.

6) NLR / NLU Processing Unit

The Natural Language Reasoning and Natural Language Understanding Processing Unit is one of the PDA's core components. The search, identification and classification of text happens at this stage. At this in time, the PDA's interim version of the implements a hybrid approach for NLP processing. This means that it combines the usage of NLTK, Standard CoreNLP, spaCy and regular expressions in order to identify as many entities as it can possibly find. Of course, some entities might not be accurate or may even be repeated. Nevertheless, this is a situation we will tackle after the interim version. As well as to use our own ML models.

At this stage, the actual data to be analysed is already extracted and ready to be forwarded to the NLP pipeline. As such, the NLP pipeline can be described by the following steps:

a) Settings and initialisation

The first step is to import the necessary libraries such as NLTK, spaCy and other utilities. Then, it is necessary to indicate the path to the location of Stanford CoreNLP model and tagger. Finally, we define the data structures to be used: *doc_contents*, *page_number*, *lang*, *filetype*, *sentences* and *tokens*.

b) NLTK

With the Natural Language Toolkit (NLTK) it is necessary to perform tokenization and part of speech tagging before starting the named entity recognition process. For that,

we execute *word_tokenize(doc_contents)* and *pos_tag(sent)*. Beyond this point, we have a collection of tagged tokens ready to be forwarded to the named entity recognition, which is a *chunker*. We execute our custom made *nltk_ner(tags)* function and save to data structures the named entities that were found (e.g., PERSON or DATE).

c) Stanford CoreNLP

In this case, the process is similar in terms of the need for part of speech tagging. Our custom made *stanford_ner()* first performs POS tagging on the data and then runs *StanfordNERTagger(STN_NER_TAGGER)*. Upon the end of execution, we filter the entities we are looking for (e.g., PERSON or LOCATION) and save the results to data structures.

d) SpaCy

On the other hand, spaCy does not require previous part of speech tagging of the text in separate functions. Instead, it is only necessary to load the NER model (*en_core_web_lg*) and run *nlp(doc_contents)*. The resultant data structure contains the identified named entities as well as other relevant information that it automatically extracts. It is a very straight forward process. It only necessary to filter the entities we are looking for with spaCy (e.g., TIME or MONEY) and save the results to data internal data structures.

e) Regular Expressions

Last stage of the NLP pipeline in the interim version is the regular expressions component. In this case, the data to be analysed is fed to the function *regular_expressions(file_input)* which in turns uses the *CommonRegex(input_string)* function from the *CommonRegex* library. Once more, the results are saved to local data structures.

When the previously described pipeline is finished, all the discovered information is ready to be sent back to the requester or displayed in the console (the latter, in the case of this interim version). Since one of the PDA's proposed features for the version is "Preliminary PII Analysis: Through NLP tools, identify named entities (to be defined)", the current pipeline fulfils the objective.

7) ML Reasoning Unit

The machine learning reasoning unit is where the evaluation of all the previously gathered information takes place. Machine learning algorithms (e.g., supervised learning or decision trees) and pre-defined rules are used to assess the relationships between the newly discovered entities, infer privacy violations and so forth. This component is not yet available for the interim version due to the necessary research and experiments performed before. For the final version of the PDA, this component will provide the results back to the requester service. Privacy metrics and sensitivity degree of PII fields are also considered at this stage.

8) Privacy Metrics

This component is a repository of privacy metrics that will be used by the PDA after the interim version. This repository is intended to allow calculations to be made on the

analysed data. Such metrics can be used, for instance, to assess the number and type of sensitive attributes in each transaction, the uniqueness of records, or other kinds of measures, which allows the PDA to provide accurate estimates about privacy risks.

It is now possible to conclude that all the objectives defined for the PDA interim version were achieved. The next section provides the necessary documentation on how to configure and deploy the container on the staging environment.

4.5. Configuration and Deployment

Similar to the RMM module, to follow the 12-factor standards and the choice of Kubernetes derived from Work Package 5 (Integration), the configuration of the PDA module happens by setting environment variables. Currently, environment variables are passed to the docker build tool. However, this is going to change in the final version, as the settings should be supplied through the runtime environment in order to prevent secrets from being accessible by inspecting the respective containers. Deployment is performed through an OCI container image.

A Dockerfile and a Pipfile execute the necessary commands and set up the required packages for the module loading and initialisation. With these two files, the image is seamlessly built and the PDA module becomes available. In order to automate the process and use intended image registries and code repositories, a make file pulls the PoSeID-on's Python base image and, after building on top of it, pushes the resulting images to the current image repository (i.e., Tecnalia's artifactory).

A complete list of environment variables will be provided for the final version. The currently used environment variables are the following:

_MESSAGE_BUS_PASSWORD

Password for the RabbitMQ queue connection.

_MESSAGE_BUS_USERNAME

Username for the RabbitMQ queue connection.

_MESSAGE_BUS_QUEUE_ADDRESS

Host address for the RabbitMQ queue connection.

_MESSAGE_BUS_QUEUE_PORT

Port for the RabbitMQ queue connection.

_MESSAGE_BUS_QUEUE_NAME

Name of the RabbitMQ queue the RMM will connect to.

Additional detail is provided in the PDA code repository. All the necessary python files for each component, as well as the Dockerfile, Pipfile, and Makefile used for the configuration and deployment of the module, are presented.

4.6. Unit Testing and Validation

Due to the priority given towards having a working prototype ready for integration, the current stage of development was subject to limited unit testing and validation. Regardless of that fact, the main components were informally tested and experimented, which, nevertheless, does not provide complete and consistent coverage of all operations. Only critical aspects of certain libraries were tested in order to guarantee the proper functioning of the components. The final version of the module will undergo all the necessary tests and full validation procedures fully completed.

Similar to the RMM, the PDA will also have unit and integration testing that will be performed as part of Work Package 6, which focuses on the testing and evaluation of the PoSeID-on platform as a whole. For more information on the roadmap for pilot evaluation and testing, deliverable D6.1 [24] can be consulted.

5. Integration approach

As it is shown in Figure 1, there are several modules in need of communication and integration. Such modules are mainly developed in Python or Java, and rely on the PoSeID-on's proposed communication scheme and integrated environment to communicate with another one. Communication-wise, the message flow is supported by the message bus. In terms of integration and deployment, there is a staging environment for the interim version. As deliverable D5.1 fully details the integration approach, next we only provide an overview of the integration approach for the D4.3 modules.

5.1. Component communication

The Message Bus is a middleware which leverages communications between the PoSeID-on system components, in a decoupled fashion, allowing for the easy addition and removal of system components and their communication. In a generic sense, a message bus is a combination of a data model for structuring system messages and communications, a common command set which specifies the available operations on messages and between connected components and a messaging infrastructure in charge of providing the set of functionalities associated with the command set. Given the above, it is not expected that the message bus is developed by PoSeID-on, as there are several good solutions providing message bus functionality already available.

During the integration discussion in Work Package 5, one of the possibilities that were analysed and later chosen to provide the message bus functionality was RabbitMQ [39]. It provides functionalities which are well suited to PoSeID-on. For instance:

- Publish and subscribe to streams of messages, which are similar to message queues;
- Store streams of records in a fault-tolerant and persistent way;
- Process streams of records as they occur;
- High availability.

RabbitMQ requires only three or more servers to provide high availability. It is also a lossless mechanism and supports Remote Procedure Call (RPC), used for the messages. The node performance is, on average, 20,000 msg/sec.

All the messages flowing through the Message Bus are serialized and dully encrypted. The serialization uses Google's Protocol Buffer [13] due to its simplicity, speed, and efficiency. To guarantee the security and privacy of the messages, PoSelD-on employs a custom message protocol which uses the Libsodium software library [14] for encryption, decryption, and signing. Therefore, all messages are signed with a module-specific ED25519 [40] key and encrypted using the respective Curve25519 [41] key. This requirement is applied to the messages of all modules integrated with PoSelD-on.

Besides communication requirements, it is necessary to configure and deploy the modules in order to complete the integration. For that purpose, the interim version of the PoSelD-on platform has a staging environment which acts as a development environment for integration and testing purposes. It is composed by a machine hosted at the datacentre of the Department of Informatics Engineering of the University of Coimbra. The staging environment has a functional setup comprising *Minikube*, *Docker*, and *add-ons* such as *ingress* and *heapster*, which together compose the set of the minimum software.

5.2. Interim version features

Module	Feature	Description
PDA	Request Handling	Process and Extract information for analysis requests
PDA	Supported Data Types	Process Structured Data as well as PDF files, TXT, CSV
PDA	Preliminary PII Analysis	Through NLP tools, identify named entities
RMM	Message Pre-processing	Process the stream of messages received into a relevant dataset for analysis
RMM	Identified Risks Visualization	View of the data instances flagged as risks and the regular data pattern
RMM	Preliminary Risk Analysis	Identification of risk events based on heuristics and clustering algorithms applied to the data stream

Table 14 - Interim Version Features (PDA and RMM)

To allow for an easier integration, a feature-set was defined for the interim version of the two modules, listed in Table 14. At the current stage of development, the interim version of the PDA is able to receive requests, process and extract the respective information. It is also capable of processing structured information like direct messages or RPC messages, and different file types such as PDF, TXT and CSV. Finally, it is also able to search and identify named entities within the processed data. Regarding the RMM, at its current stage it is also able to receive requests, process and extract the information from messages and run the parsing and feature extraction steps on the received data. Due to the implementation of a custom version of the log parsing algorithm, development was delayed by two weeks, and in the end the expected features fell short, as we cannot yet run the risk analysis and subsequent results visualization. This is a minor step, however, which will be implemented in the upcoming weeks, and results up until the anomaly detection step are already being shown and working successfully.

5.3. Final version features

For the final version of the modules, the remaining requirements must be met for each module. As such, for the Risk Management Module, the following features will be added:

- Anomaly detection
 - Finishing the stream layer anomaly detection pipeline
 - Implementing the batch layer anomaly detection
- Persistent storage for:
 - Received and parsed logs
 - Feature vectors
 - Analysis results
 - Data Processor reputation metrics

For the Personal Data Analyser, it is necessary to complete the following features:

- PII Discovery
 - Usage of improved NLP models
 - Multi-language capabilities
- Privacy Analysis
 - Analyse PII
 - Privacy Metrics
 - PII type sensitiveness
 - Analyse PII correlations
 - Likelihood of exposure by having access to specific PII types
 - Request DP reputation to RMM
 - Upon initialization, request DP reputation list to RMM
 - Receive from RMM module the DP reputation list updates
 - Creation of rules / models
 - Setting privacy thresholds for warning generation

For both modules it is also necessary to guarantee the following:

- Communication
 - RPC handling for result delivery
 - RPC handling for DP metrics delivery
 - Notification of Data Subjects and Processors in case of risk/privacy detection
- Security
 - Configuration of the modules dependencies and frameworks must be implemented in a way that ensures the proper levels of security in production
- Unit testing
 - Automated tests for critical functionalities of the module will be implemented
- Deployment and configuration
 - Configuration of the module for scalable deployment using multiple instances
 - Documentation for the deployment configurations of the module

From the listing presented before, it is possible to conclude that several requirements involving security, testing, validation or configuration are transversal to both modules. Other requirements are more specific and related to particularities of risk and privacy exposures.

6. Innovation Summary

To summarize the contributions of both modules, the innovative aspects of each are presented in this section.

The Risk Management Module novelty revolves around the identification of privacy exposure risks by using anomaly detection on system logs and operational information, which, as far as we know, is a completely novel area. In addition, we will score data processors based on their interaction with data subjects' PII and interaction with the PoSelD-on system, according to the anomaly detection results. Finally, the implementation of an open-source module for log anomaly detection, with a complete pipeline ready for production, is also a major contribution to both the log anomaly detection field and the privacy field, with the concrete application of results for the purpose of privacy enhancement.

The Personal Data Analyzer is a fully GDPR-compliant monitoring module for personal data transactions (PII). To achieve this goal, the PDA uses NLP (NLU / NLR) and Machine Learning to perform identification and classification of PII. Moreover, it provides privacy analysis and scores based on the types of PII exchanged, their sensitiveness, the reputation of the involved Data Processors, and privacy metrics. Such guarantees are easily verified, since not only it is a fully GDPR-compliant tool, but it is also open source and, therefore, its code can be verified by anyone. Its innovation also relies on the kind of guarantees it provides, while processing of personal information: it never stores any information and only processes PII with the explicit consent provided by data owners. Such combination of features results in a very innovative and transparent kind of application, that is capable of processing sensitive data while assuring privacy guarantees.

7. Conclusion and future work

This document presented details of the interim implementation of two central modules in the PoSelD-on system, namely the Risk Management Module and the Personal Data analyser module. Implementation details include low-level component description, functionalities and dependencies. Moreover, the NLP experimental research was presented and analysed in depth. According to the adopted document structure, after presenting the module's implementation details, it is provided the necessary documentation regarding configuration and deployment of the modules, as well as the integration approach that was followed.

The proposed features were fulfilled for the PDA and partially fulfilled for the RMM. The development process continues towards the final version of the modules. Therefore, the next steps for the PDA are to propose custom-build AI models capable of identifying previously non-identified PII and perform compliance verifications in order to verify compliance and provide automated privacy assurances to data owners. For the RMM, the development of the features which had some delay will continue as well as the development of the batch counterpart of the anomaly detection and the display and processing the results to be used in the Dashboard.

References

- [1] R. Casaleiro, P. Silva and all, "Deliverable 2.2 System Requirements and Architecture," PoSeID-on H2020, 2018.
- [2] J. v. Rooij and all, "Preliminary PoSeID-on Integrated Platform," PoSeID-on H2020, 2019.
- [3] J. v. Rooij and all, "Deliverable D4.1 Web-based Dashboard Interim Implementation," PoSeID-on H2020, 2019.
- [4] L. Nicoletti and J. E. Cuenca, "D2.1 Use cases analysis and user scenarios," PoSeID-on H2020, 2018.
- [5] N. Marz, "Lambda architecture," [Online]. Available: <http://lambda-architecture.net>.
- [6] Y. Yamato, H. Kumazaki and Y. Fukumoto, "Proposal of Lambda Architecture Adoption for Real Time Predictive Maintenance," in *2016 Fourth International Symposium on Computing and Networking (CANDAR)*, 2016.
- [7] L. Rosa, J. Proença, J. Henriques, V. Graveto, T. Cruz, P. Simoes, F. Caldeira and E. Monteiro, "An evolved security architecture for distributed Industrial Automation and Control Systems," in *In Proc. of 16th European Conference on Cyber Warfare and Security (ECCWS 2017)*, 2017.
- [8] P. Casas, F. Soro, J. Vanerio, G. Settanni and A. D'Alconzo, "Network security and anomaly detection with Big-DAMA, a big data analytics framework," in *Proceedings of the 2017 IEEE 6th International Conference on Cloud Networking, CloudNet 2017*, 2017.
- [9] M. Astekin, H. Zengin and H. Sozer, "Evaluation of Distributed Machine Learning Algorithms for Anomaly Detection from Large-Scale System Logs: A Case Study," in *2018 IEEE International Conference on Big Data (Big Data)*, 2018.
- [10] Q. Lin, H. Zhang, J.-G. Lou, Y. Zhang and X. Chen, "Log clustering based problem identification for online service systems," in *Proceedings of the 38th International Conference on Software Engineering Companion - ICSE '16*, 2016.
- [11] Z. Liu, T. Qin, X. Guan, H. Jiang and C. Wang, "An Integrated Method for Anomaly Detection From Massive System Logs," *IEEE Access*, vol. 6, pp. 30602-30611, 2018.
- [12] S. He, J. Zhu, P. He and M. R. Lyu, "Experience Report: System Log Analysis for Anomaly Detection," in *2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)*, 2016.
- [13] Google, "Protocol buffers," 07 2019. [Online]. Available: <http://code.google.com/apis/protocolbuffers/>.
- [14] "Libsodium - a modern, easy-to-use software library for encryption, decryption, signatures, password hassing and more...," 07 2019. [Online]. Available: <https://download.libsodium.org/doc/>.
- [15] "GELF - Structured events from anywhere. Compressed and chunked," Graylog, [Online]. Available: <https://docs.graylog.org/en/3.0/pages/gelf.html>. [Accessed July 2019].



- [16] Z. J, H. S, L. J, H. P, X. Q and e. al., "Tools and benchmarks for automated log parsing," *Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice*, pp. 121-130, 2019.
- [17] M. Mizutani, "Incremental mining of system log format," *SCC*, p. 595–602, 2013.
- [18] D. M and L. F, "Spell: Streaming Parsing of System Event Logs," in *2016 IEEE 16th International Conference on Data Mining (ICDM)*, 2016.
- [19] H. P., Z. J., Z. Z. and L. M., "Drain: An Online Log Parsing Approach with Fixed Depth Tree," in *2017 IEEE International Conference on Web Services (ICWS)*, 2017.
- [20] "Apache Spark: Lightning-fast unified analytics engine," The Apache Software Foundation, [Online]. Available: <https://spark.apache.org/>. [Accessed July 2019].
- [21] "Spark MLlib: Apache Spark's scalable machine learning library.," The Apache Software Foundation, [Online]. Available: <https://spark.apache.org/mllib/>. [Accessed July 2019].
- [22] "LogPAI," [Online]. Available: <http://www.logpai.com/>. [Accessed July 2019].
- [23] "HDFS Logs Labeled Dataset," LogPAI, [Online]. Available: <https://github.com/logpai/loghub/tree/master/HDFS>. [Accessed July 2019].
- [24] A. Bagnato, L. Goncalves, L. Nicoletti and all, "Deliverable D6.1 Pilot Plan," PoSeID-on H2020, 2019.
- [25] S. Bird, E. Klein and E. Loper, *Natural Language Processing with Python*, O'Reilly Media., 2009.
- [26] C. D. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. Bethard and D. McClosky, "The Stanford CoreNLP Natural Language Processing Toolkit," in *Association for Computational Linguistics (ACL) System Demonstrations*, 2014.
- [27] ExplosionAI, "spaCy - Industrial-Strength Natural Language Processing," [Online]. Available: <https://spacy.io>. [Accessed 06 2019].
- [28] S. Loria, "TextBlob Documentation," [Online]. Available: <https://buildmedia.readthedocs.org/media/pdf/textblob/latest/textblob.pdf>. [Accessed 06 2019].
- [29] R. Al-Rfou, V. Kulkarni, B. Perozzi and S. Skiena, "Polyglot-NER: Massive multilingual named entity recognition," in *Proceedings of the 2015 SIAM International Conference on Data Mining*, 2015.
- [30] C. Hamish, D. Maynard, K. Bontcheva, V. Tablan and Y. Wilks, "Experience using GATE for NLP R&D," in *Proceedings of the COLING-2000 Workshop on Using Toolsets and Architectures To Build NLP Systems*, 2000.
- [31] Google, "SyntaxNet: Neural Models of Syntax," [Online]. Available: <https://github.com/tensorflow/models/tree/master/research/syntaxnet>. [Accessed 06 2019].
- [32] TensorFlow, "An end-to-end open source machine learning platform," [Online]. Available: <https://www.tensorflow.org>. [Accessed 06 2019].
- [33] Keras, "Keras: The Python Deep Learning library," [Online]. Available: <https://keras.io>. [Accessed 06 2019].
- [34] L. Ratnoff and D. Roth, "Design challenges and misconceptions in named entity recognition," in *Thirteen Conference on Computational Natural Language Learning*, Boulder, Colorado, 2009.



- [35] M. K. Malik and S. M. Sarwar, "Named Entity Recognition System for Postpositional Languages: Urdu as a Case Study," (*IJACSA International Journal of Advanced Computer Science and Applications*, vol. 7, no. 10, pp. 141-147, 2016.
- [36] spaCY, "Language Models - English - en_core_web_sm," [Online]. Available: <https://spacy.io/models/en>. [Accessed 06 2019].
- [37] spaCy, "Language Models - English - en_core_web_md," [Online]. Available: <https://spacy.io/models/en>. [Accessed 06 2019].
- [38] spaCy, "Language Models - English - en_core_web_lg," [Online]. Available: <https://spacy.io/models/en>. [Accessed 06 2019].
- [39] "RabbitMQ - open source message broker," 07 2019. [Online]. Available: <https://www.rabbitmq.com>.
- [40] D. Bernstein, N. Duif, T. Lange, P. Schwabe and B.-Y. Yang, "High- speed high-security signatures," *Journal of Cryptographic Engineering*, 2012.
- [41] D. Bernstein, "New Diffie-Hellman speed records," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2006.