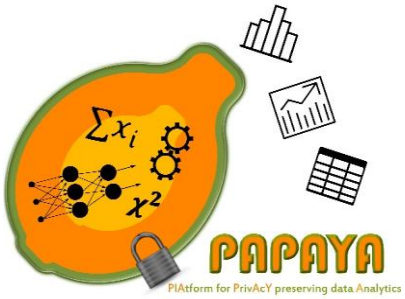



# D4.1 – FUNCTIONAL DESIGN AND PLATFORM ARCHITECTURE

| | |
|---|---|
| Work Package | WP 4, Platform Design and Development |
| Lead Author | Boris Rozenberg, Ron Shmelkin, Muhammad Barham (IBM) |
| Contributing Author(s) | Beyza Bozdemir, Orhan Ermis, Melek Önen (EURC)<br>Monir Azraoui, Sebastien Canard, Bastien Vialla (ORA)<br>Angel Palomares Perez (ATOS)<br>Tobias Pulls (KAU) |
| Reviewers | Sebastien Canard (ORA)<br>Angel Palomares Perez (ATOS) |
| Due date | 31.07.2019 |
| Date | 31.07.2019 |
| Version | 1.0 |
| Dissemination Level | PU (Public) |

**The research leading to these results has received funding from the European Union's Horizon 2020 Research and Innovation Programme, through the PAPAYA project, under Grant Agreement No. 786767. The content and results of this deliverable reflect the view of the consortium only. The Research Executive Agency is not responsible for any use that may be made of the information it contains.**

# Revision History

| Revision | Date | Editor | Notes |
|---|---|---|---|
| 0.1 | 03.04.2019 | Ron Shmelkin (IBM) | ToC |
| 0.2 | 16.04.2019 | Tobias Pulls (KAU) | ToC update, sketch of dashboards and auditing |
| 0.3 | 03.06.2019 | Orhan Ermis, Melek Önen (EURC) | EURC's contribution |
| 0.4 | 24.06.2019 | Boris Rozenberg (IBM) | Integrate partners' inputs |
| 0.5 | 24.06.2019 | Monir Azraoui, Sébastien Canard, Bastien Vialla (ORA) | ORA's contribution |
| 0.6 | 26.06.2019 | Boris Rozenberg, Ron Shmelkin (IBM) | Comments on partners' inputs |
| 0.7 | 3.07.2019 | Boris Rozenberg (IBM) | Integrate partners' inputs and review |
| 0.8 | 8.07.2019 | Boris Rozenberg (IBM) | Integrate partners' additional inputs and review – version for the 1st internal review |
| 0.9 | 24.07.2019 | All contributing authors | Address comments from the 1st internal review – version for the 2nd internal review |
| 0.10 | 29.07.2019 | All contributing authors | Address comments from the 2nd internal review |
| 1.0 | 31.07.2019 | Beyza Bozdemir (EURC) | Quality check |

## Table of Contents

## List of Figures

# Executive Summary

Rather than several standalone modules, the PAPAYA project aims at developing an integrated platform for privacy preserving data analytics to make them available in a broad spectrum of products and services, with usable, friendly and accessible safeguards options. The main goal of the platform is to be used by service developers to deploy and run privacy-preserving services, and by service consumers, that are interested to employ privacy-preserving analytics. The document presents the platform functional design, architecture, and deployment. It describes the design of the main platform components that were elicited based on the requirements presented in D2.2 [1]. Moreover, it explains how different privacy preserving primitives that will be developed in WP3 (see D3.1 [2]) are integrated into the platform in a way that they will be interoperable/compatible with each other and could work together in the integrated platform. Furthermore, it presents the design of platform dashboards that will provide the UI, configuration functionality, and visualization functionality.

PAPAYA platform services are specified based on the four generic usage scenarios, namely upload model, create model, apply model and collaborative training. In upload model, an already trained machine learning (ML) model can be uploaded to the PAPAYA platform when the client wants to delegate the computationally intensive task (which is applying a model on the client's sensitive data) in a privacy-preserving manner. The create model is used when the client is not able to create the ML model; therefore, the PAPAYA platform generates a model on the protected data shared by the client. Apply model is the use case where already uploaded or created model is applied on the client's protected data in a privacy-preserving manner. Finally, in collaborative training, two or more participants perform a ML training collaboratively while preserving the privacy of the training data.

The PAPAYA platform consists of the following services:

- Privacy-preserving analytics defined in deliverable D3.1 [2] such as classification on Neural Networks, privacy-preserving clustering, privacy-preserving statistics, and privacy-preserving collaborative training of Neural Networks
- Security and transparency services, including the identity access management (IAM) for authentication and authorization services to the different components that will be integrated in the PAPAYA platform, auditing to support auditing and towards being able to hold stakeholders accountable for their use of PAPAYA, and key manager for managing the cryptographic material during the whole lifecycle of the PAPAYA project in the cases where it will be required.
- The PAPAYA platform provides two dashboards for configuration and visualization, namely the platform dashboard and the agent dashboard. The platform dashboard is used for configuration and monitoring the services provided by the platform whereas the agent dashboard is used for viewing the data processing logs from an agent and for showing the configuration of the agent.
- Data subject toolbox services provide a number of mostly independent tools (Explaining Privacy-preserving Analytics, Data Disclosure Visualization, Annotated Log View and

Privacy Engine) which provide versatile services related to the data subject privacy. Moreover, the PAPAYA platform provides means to its clients to integrate one or more tools from the provided toolbox to generate an integrated *data subject dashboard*.

As a proof of concept usage, the platform will be deployed on IBM Kubernetes cloud service. In addition to that, all services are designed to be generic in order to be deployed on any other cloud platforms.

The intermediate platform implementation and PETs integration, including intermediate implementation of the dashboard will be presented in deliverable D4.2 and the final platform implementation and PETs integration after validation on project use cases, including final version of the dashboard will be given in deliverable D4.3.

# Glossary of Terms

| | |
|---|---|
| 2PC | Two-Party Computation |
| AA | Auditing Agent |
| ABY | Arithmetic sharing, Boolean sharing and Yao's garbled circuits framework |
| AC | Auditing Collector |
| ALT | Annotated Log view Tool |
| API | Application Programming Interface |
| BF | Bloom Filters |
| BGV | Brakerski-Gentry-Vaikuntanathan homomorphic encryption scheme [3] |
| BFV | Brakerski/Fan-Vercauteren fully homomorphic encryption scheme [4, 5] |
| CA | Certificate Authority |
| CKKS | Cheon-Kim-Kim-Song fully homomorphic encryption scheme [6] |
| CLI | Command Line Interface |
| CNN | Convolutional Neural Network |
| CPU | Central Processing Unit |
| CR | Container Registry |
| DB | Data Base |
| DC | Data Controller |
| DP | Differential Privacy |
| DNN | Deep Neural Network |
| DS | Data Subject |
| DSRM | Data Subject Rights Manager |
| DVT | Disclosure Visualization Tool |
| ECG | Electro cardiogram |
| ES | ElasticSearch |
| FC | Fully Connected |
| FE | Functional Encryption |
| FHE | Fully Homomorphic Encryption |
| GRU | Gated Recurrent Unit |
| HE | Homomorphic Encryption |
| IAM | Identity Access Manager |
| KM | Key Manager |
| ML | Machine Learning |
| MLP | Multilayer Perceptron |
| NN | Neural Network |
| PE | Privacy Engine |
| PHE | Partially Homomorphic Encryption |
| PoC | Proof of Concept |
| PPM | Privacy Preferences Manager |
| RAM | Random Access Memory |

| | |
|---|---|
| REST | Representational State Transfer |
| ReLU | Rectified Linear Unit |
| RNN | Recurrent Neural Network |
| SIMD | Single Instruction Multiple Data |

# 1 Introduction

## 1.1 Purpose and Scope

The purpose of this deliverable is to provide a comprehensive description of the PAPAYA platform architecture. The intended audience for this document consists of two main groups: (1) service developers, who could use the document to understand the design of the existing services and to develop and deploy their own services; (2) service users, who could use the document to understand how to employ the services available on the platform and how to incorporate them in their applications.

It is important to note that the description of the underlying algorithms employed by the services presented in this document is not in the scope of this deliverable (they are described partially in D3.1 [2] and will be described in D3.3 [7]).

## 1.2 Structure of the Document

The rest of the document is organized as follows:

- **Section 2** presents PAPAYA stakeholders and generic use cases.
- **Section 3** provides a high-level overview of the PAPAYA platform architecture.
- **Section 4** describes in detail the potential core PAPAYA services, which are already defined in deliverable D3.1 [2] and to be defined in deliverable D3.3 [7]. Provided services in the scope of this deliverable are: (1) Privacy-preserving classification on Neural Networks; (2) Privacy-preserving collaborative training of Neural Networks; (3) Privacy-preserving clustering; and (4) Privacy-preserving statistics.
- **Section 5** gives details about how security and transparency is achieved in the platform, including authentication, authorization, auditing and key management for cryptographic tools (if it is required).
- **Section 0** presents PAPAYA dashboards, namely platform dashboard and agent dashboard.
- **Section 7** dedicated to Data Subject Toolbox, which provides versatile services (Explaining Privacy-preserving Analytics, Data Disclosure Visualization, Annotated Log View and Privacy Engine) related to the data subject privacy. Tools presented in this section can then be used to form a *data subject dashboard*.
- In **Section 8** we describe how to deploy and run all platform components.
- In **Section 9** we describe how we plan to evaluate the integrated platform.

# 2 Stakeholders and Usage Scenarios

The platform stakeholders can be mainly separated into three permission groups:

1. Platform administrators – responsible for the platform resource allocation, monitoring of resources and services (internal and external) that will run on the platform or will communicate with the platform.
2. Service providers – the author of the services. The service providers will be able to upload/edit and delete services to the platform.
3. Platform client – will use the services provided by the platform. The platform client is split into two separate identities:
    a. Client app – the application that will communicate with the service's dedicate agent that will run on a client side.
    b. Client admin – the person that will deploy the service client-side agent and integrate it with the client app.

Figure 1 presents the main flows of the platform services. Most of the services will implement two main common phases, namely upload/create model and apply model. In this section, we describe the generic usage scenarios, while in Section 3 we provide a detailed description for each service.

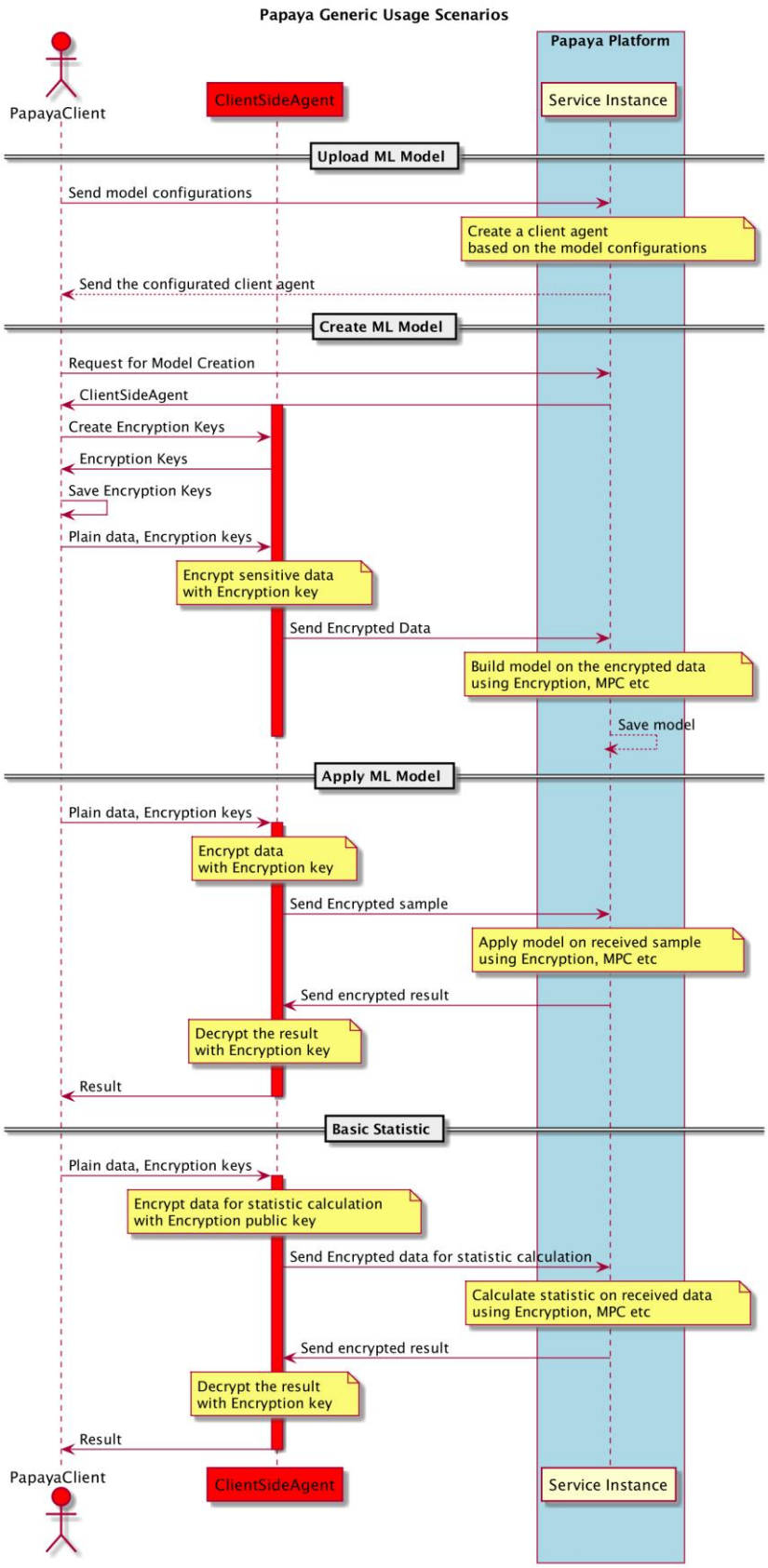
**Upload model** – this usage scenario relates to the scenario when the Machine Learning (ML) model already exists. The client is willing to apply this model on sensitive data in a privacy-preserving manner (apply the model use case). The client will upload the model to the server using the client agent.

**Create model** – this usage scenario relates to the scenario when the client does not have the ability to build a ML model and will use the platform for this purpose. The model will be built on the encrypted data. In such case, the platform will not be exposed to sensitive or personal client's data. As the result of this scenario, the client will be able to download the model and use it locally, or to apply the model, which will be hosted on the platform. The client will use the client agent in order to obtain the cryptographic keys to encrypt the data with it. By using this encrypted data, the server will create the trained ML model.

**Apply model** – this usage scenario relates to the scenario when the client is willing to apply a ML model on a sample that contains sensitive data in a privacy-preserving manner. The client will encrypt the sensitive data/sample and send it to the server using the client agent. The server will apply the model on the encrypted data. The agent will obtain the result and decrypt it. One of the usage scenario's subcases is **Basic statistic.** This usage scenario is conceptually identical to the apply model, while the main difference is that in this scenario the server will calculate basic statistic functions on the encrypted data.

**Collaborative training** – this usage scenario allows two or more participants to perform a ML training collaboratively, while preserving the privacy of the training data. The participants join the collaborative training via the client-side agents that will be provided by the platform. The participants provide the training data to the agent. The agents perform the training locally, on the

client's premises, and exchange the knowledge achieved from the data with other participants through the centralized server. As a result, the clients will be able to download a more accurate ML model than the one obtained by performing the training task only on a participant's data.


Papaya Generic Usage Scenarios
Papaya Platform
PapayaClient
ClientSideAgent
Service Instance
Upload ML Model
Send model configurations
Create a client agent
based on the model configurations
Send the configurated client agent
Create ML Model
Request for Model Creation
ClientSideAgent
Create Encryption Keys
Encryption Keys
Save Encryption Keys
Plain data, Encryption keys
Encrypt sensitive data
with Encryption key
Send Encrypted Data
Build model on the encrypted data
using Encryption, MPC etc
Save model
Apply ML Model
Plain data, Encryption keys
Encrypt data
with Encryption key
Send Encrypted sample
Apply model on received sample
using Encryption, MPC etc
Send encrypted result
Decrypt the result
with Encryption key
Result
Basic Statistic
Plain data, Encryption keys
Encrypt data for statistic calculation
with Encryption public key
Send Encrypted data for statistic calculation
Calculate statistic on received data
using Encryption, MPC etc
Send encrypted result
Decrypt the result
with Encryption key
Result
PapayaClient
ClientSideAgent
Service Instance

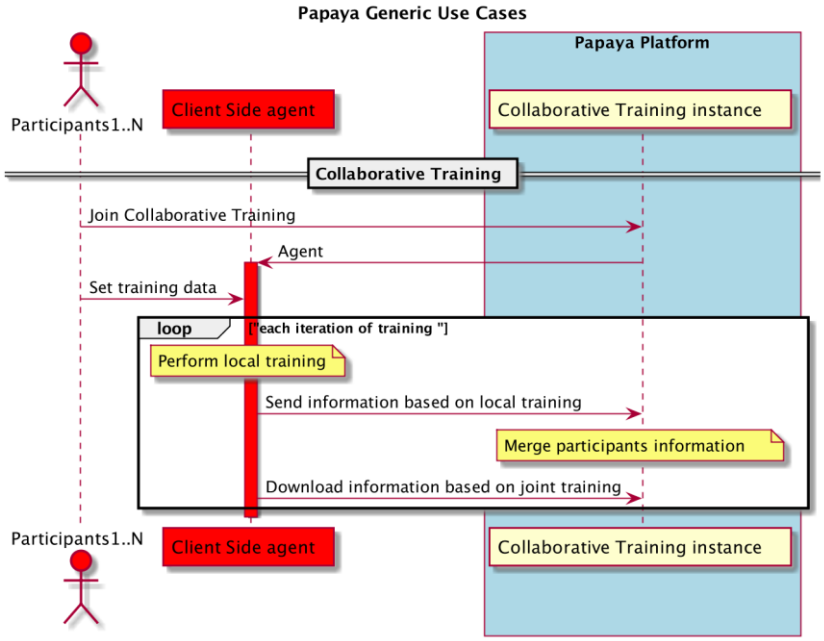



*Figure 1: PAPAYA Generic Usage Scenarios*

# 3 Platform Architecture

The PAPAYA platform (see Figure 2) is composed of two main groups of components: (1) the server-side components that will be running on the (non-trusted, but semi honest) Kubernetes[1] cloud server; and (2) agent side components, that will be running on trusted client environment.

[1] https://kubernetes.io/

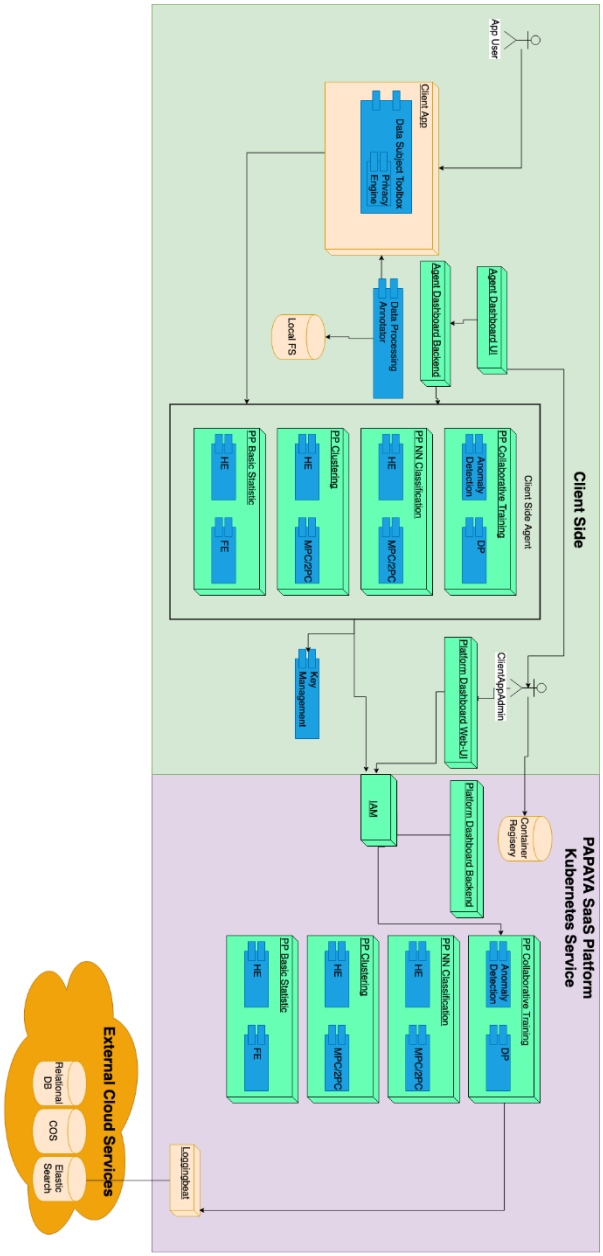



*Figure 2: PAPAYA Platform Architecture*

The **core** components of the platform can be mainly divided into the following groups:

- **The privacy-preserving analytics services** which will allow platform clients to perform service analytics of interest, in a privacy-preserving manner, such as privacy-preserving

NN classification, privacy-preserving NN collaborative training, privacy-preserving clustering and privacy-preserving basic statistics. Each service is divided into two parts:

1. Server – responsible for performing analytics of interest on encrypted data and will run on a PAPAYA's Kubernetes cluster.
2. Agent – responsible for communication with the appropriate server-side component and responsible for managing cryptographic operations for the client. The agent will be downloaded from the Container Registry (CR) as a Docker image and deployed on a client side. Deployment and execution will be under the responsibility of the application client/application admin.

These services will be described in more details in Section 4.

- **The platform security and transparency services** which will provide platform authorization, authentication (Identity and Access Management - IAM), auditing and cryptographic Key Manager (if needed). These services will be further described in more details in Section 5.

- **The PAPAYA dashboards**, Platform and Agent dashboards, which will allow platform users to choose and deploy privacy-preserving services, review operational and auditing logs. These services will be described in more details in Section 0.

- **Data Subject Toolbox** – consists of a number of mostly independent tools, which provide versatile services related to the data subject privacy. These tools will be described in more details in Section 7.

The server-side core components will communicate to external cloud services such as:

1. **Data Base** (DB) – responsible for storing relational data used by the platform dashboard or any application that runs on a platform.
2. **Elasticsearch**[2] – responsible for storing operational logs of the platform dashboard and applications that will run on a platform as described in detail in Section 5.2.
3. **Container Registry** [3](CR) – responsible for storing service images that are provided by service providers. Each service provider will provide two images for each of the services, one for the server side (will run as deployment on k8s), and the second for running on a client side.
4. **Object Storage** – responsible for storing NN model topology for privacy-preserving collaborative training service.

[2] https://www.elastic.co/
[3] https://www.ibm.com/cloud/container-registry

5. **Loggingbeat (Filebeat** [4]) – which is kind of Elastic Beat (see Section 5.2), responsible for logs harvest and shipping them to the Elasticsearch instance. The Filebeat will run as a K8s Deamonset[5].

[4] https://www.elastic.co/guide/en/beats/filebeat/5.0/filebeat-overview.html
[5] https://kubernetes.io/docs/concepts/workloads/controllers/daemonset/

# 4 Platform Services

In this section, we describe in detail the server-side components. They are divided into three main categories: (1) core platform services (Sections 4.1 - 4.4); (2) services responsible for the platform security and privacy (Section 5); and (3) services responsible for platform dashboards (Section 0).

## 4.1 Apply Neural Network Model

In this section, we describe several services for applying neural network for the purpose of classification in a privacy-preserving manner. The detailed description of these services can be found in deliverables D3.1 [2] and D3.3 [7].

Prior to using any of the services described in this section, a Neural Network model should be trained locally based on the clear text data with all the required optimization and the dimensionality reduction. After achieving the desired accuracy, the trained model (i.e. architecture and weights) should be saved in a supported format and passed to the service later as described in each of the following subsections.

### 4.1.1 Privacy-preserving NN classification based on 2PC

In this section, the functional design of privacy-preserving NN classification based on two-party computation (2PC) defined in deliverable D3.1 [2] is introduced.

#### *4.1.1.1 Main components and their relationships*

This solution has two main components:
1. Server-side component
2. Client-side component

**Server-side component:**
The server-side component consists of two modules as shown in Figure 3:

1. **Interface:** This module provides an interface to initialize the server with client's IP address, weight matrix and bias vector and also it is used to execute NN classification.
2. **NN Model:** This module is used for executing NN model using 2PC together with the client component.

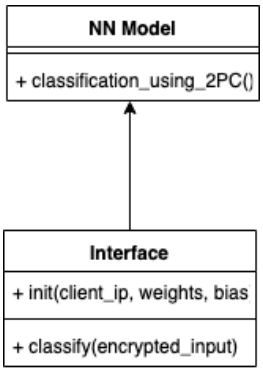



*Figure 3: Topology of server-side component*

**Client-side component:**

The client-side components consist of three modules as illustrated in Figure 4:

1. **Interface:** This module provides an interface for uploading the input data to be classified and to execute NN classification.
2. **Peak detection:** This module is used for preprocessing electro cardiogram (ECG) input file in order to obtain peak values to be used in NN model.
3. **NN model:** This module uses 2PC for executing the NN model together with the server.

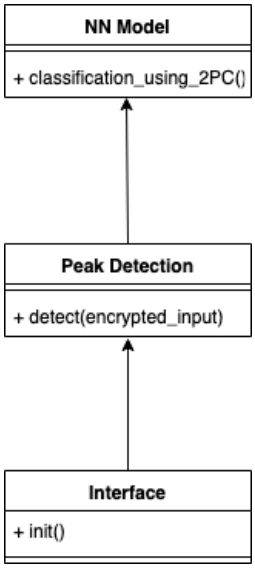



*Figure 4: Topology of client-side component*

**Detailed description of the modules:**

The NN model was obtained based on the performance and security requirements as described in deliverable D3.1 [2]. It has two fully connected (FC) layers and one activation layer in between. In this model, several approximations have been used such as $x^2$ as an activation function and the softmax function [8]. All computations involve both the client and the server, and the communication is ensured through sockets generated by ABY framework[6] (a mixed-protocol framework that efficiently combines the use of Arithmetic shares, Boolean shares, and Yao's garbled circuits). When the model is executed, the output is generated in order to show the prediction probabilities of corresponding 16 arrhythmia classes.

**Interface (server):** The interface module of the server is used for initializing the client's IP address, weights to be used in NN classification and bias vector using REST API.

**NN model (server):** The proposed model is compatible with 2PC (ABY) and is dedicated to privacy-preserving arrhythmia detection use case (UC1) defined in deliverable [9].

**Interface (client):** The interface module of the server is used for uploading the file to be classified.

**Peak Detection component:** This module is used for preprocessing the input file in order to detect detecting r-peak values in the input heartbeat file for a given time period. The peak detection, in particular R-peak – the peak used for detecting the arrhythmia class – detection operates as follows: 180 samples obtained from each heart beat is divided into 90 samples before the R-peak, 1 sample for the R-peak and remaining 89 samples after the R-peak

**NN model (client):** This component is the complementary part of the server's NN model component. The client executes NN model using ABY in order to accomplish the classification of heartbeat. Prior to the execution of NN model, in the client side, principle component analysis (PCA) is applied to reduce the dimension of the input r-peak values to comply with the first matrix to be used in the next FC (Fully connected layer). More details can be found in deliverable D3.1 [2].

#### *4.1.1.2 Behavioral Analysis*

There are two main flows in this solution: (1) initialization of the server; and (2) NN classification.

1. **Server initialization:** By using *init* REST API (see API definition in Section 4.1.1.5) call weight matrix, bias vector and the client IP address is initialized in the server side as shown in Figure 5. As an output, a string value to show whether the API call is succeeded or not is sent to the client.

[6] https://github.com/encryptogroup/ABY

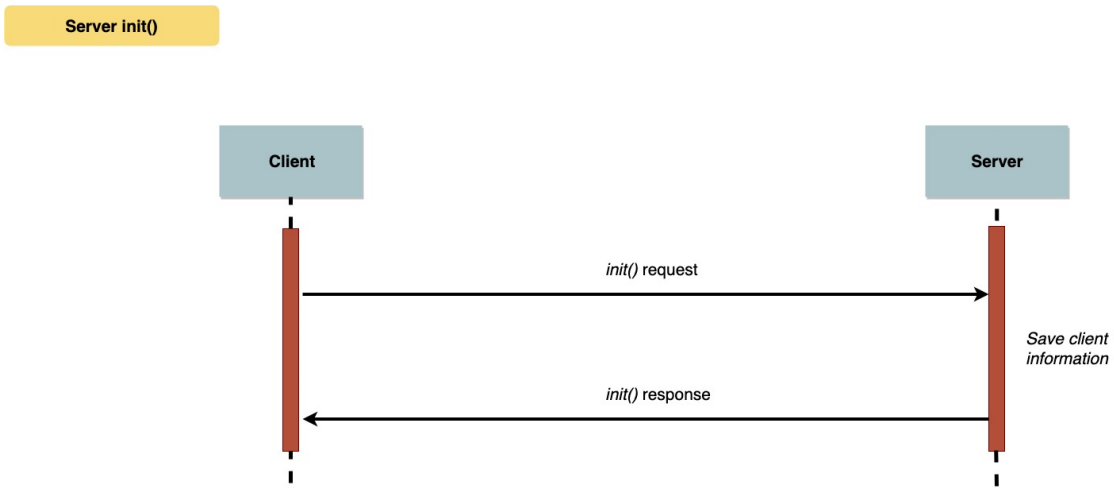



*Figure 5: Initialization of client information into the server*

2. **NN model execution:** The NN model component is executed in both the client-side and in the server-side. In order to classify a heartbeat input, *classify* REST API call of the server is used. In data flow, client sends the encrypted input to the server. Then, by using ABY, client and server make computations for the classification of the given input data in a privacy-preserving manner. All client to server and server to client API calls are automatically realized by ABY. Finally, server sends the result of the classification as a vector of arrhythmia classes. The illustration of the classification is as shown in Figure 6.

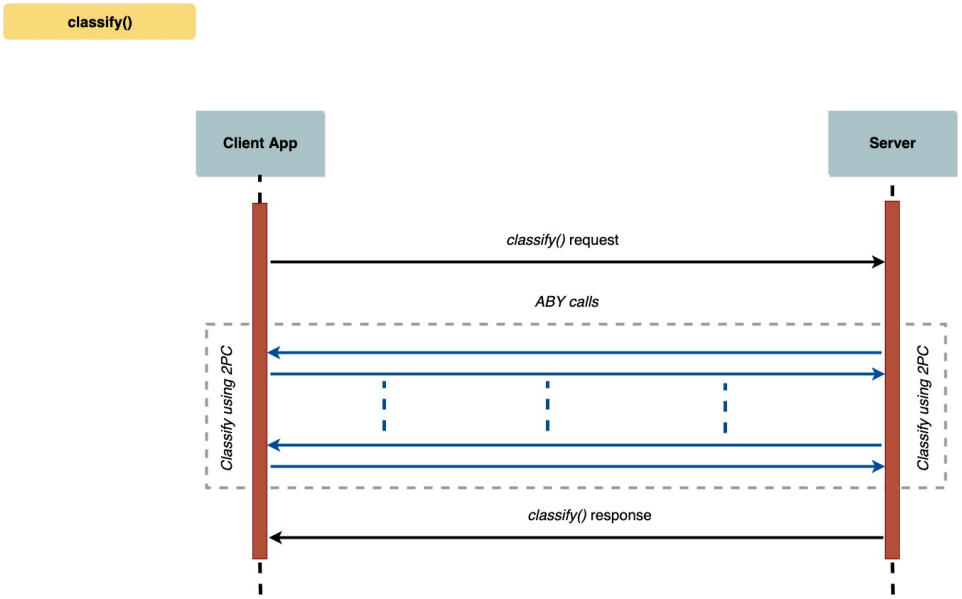



*Figure 6: NN Classification by using 2PC*

#### *4.1.1.3 Deployment and configuration*

In this section, we introduce the basics for the deployment and configuration for client and server components.

**Deployment:**
The server-side and the client-side components will be deployed as Docker containers.

**Configuration:**
There are two main configurations for this model:

1. **Server configuration:** by using the *init()* call of server-side API (see Section 4.1.1.5), the client IP address and port, weights matrix to be used for classification will be configured.
2. **ABY configuration:** In this configuration, ABY and its dependencies are installed in the server-side component.

#### *4.1.1.4 Implementation constraints*

Implementation constraints of the proposed approach is as follows:

1. The NN model used in this version is static which means that all computations are realized on a dedicated and optimized circuits involving arithmetic gates and very few Boolean gates.

2. Square function ($x^2$) is used as an activation function.

#### *4.1.1.5 APIs*

**POST / init**

The *init* REST API call is used to initialize client's IP address, weight matrix and bias vector as shown in the Figure 7.

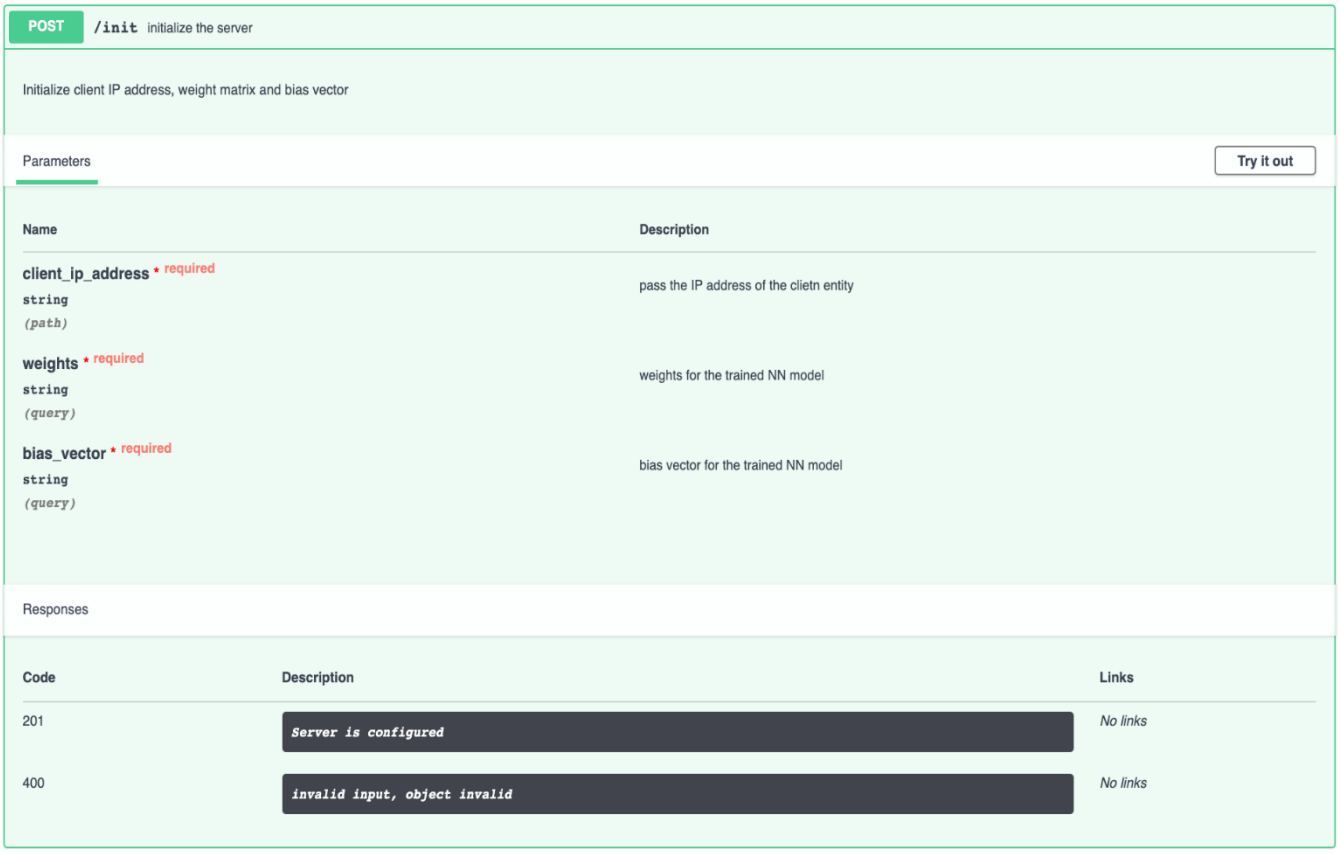



*Figure 7: POST / init REST API call for server initialization*

**POST /classify**

In this API call, the client sends the encrypted text to the server in order to start the classification as shown in Figure 8. Then, the ABY framework organizes the rest of the socket API calls between the client and the server.

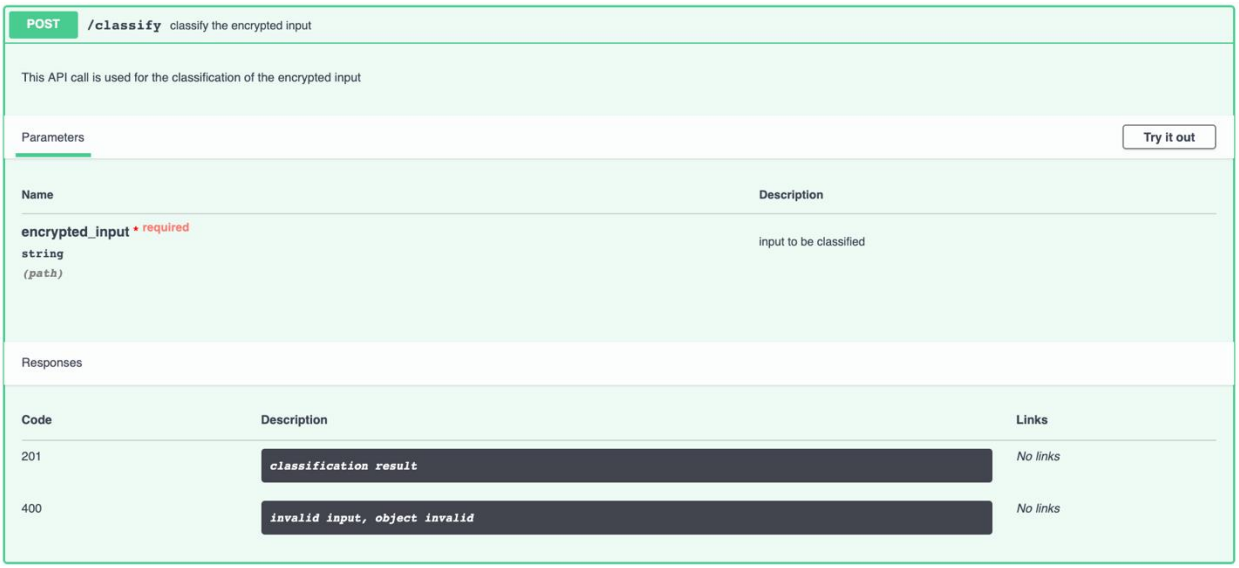



*Figure 8: POST /* classify *REST API call for classification*

### 4.1.2 Privacy-preserving NN classification based on PHE

In this section, the functional design of privacy-preserving NN classification based on partially homomorphic encryption (PHE) defined in deliverable D3.1 [2] is introduced.

#### *4.1.2.1 Main components and their relationships*

This solution consists of two main components:
1. Server-side components
2. Client-side components

**Server-side component:**
There are three modules in the server-side component as shown in Figure 9:
1. **Interface:** This module is used for initializing the NN model, weight matrix, bias vector and IP address of the client.
2. **Non-interactive computation:** This module is used for server-side only computations while computing the NN model.
3. **Interactive computation:** This module is used for interactive computations with the client while computing the NN model.

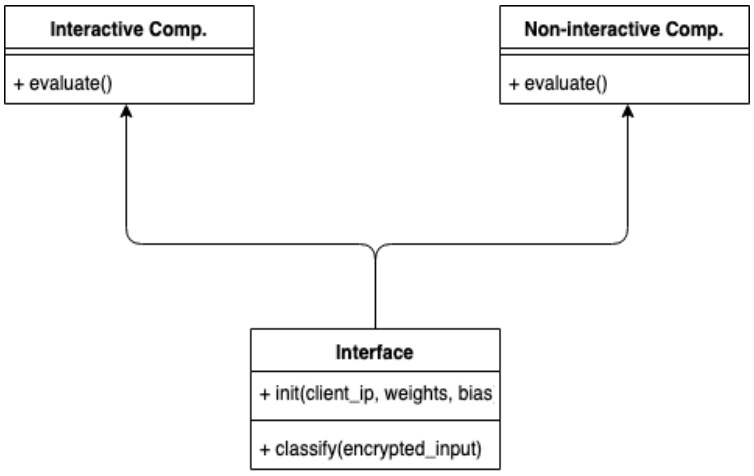



*Figure 9: Topology of server-side components*

**Client-side component:**

There are three main modules on the client-side component as shown in Figure 10:

1. **Interface:** This module is used for uploading the input to be classified
2. **Encryption/decryption module:** This module is used for encrypting and decrypting the input when it is necessary for classification.
3. **Interactive computation module:** This module is used for accomplishing the interactive computations together with the server such as the computation of activation layer.

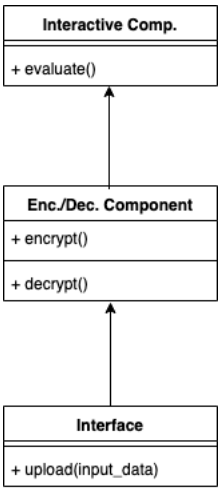



*Figure 10: Topology of client-side components*

**Detailed description of the modules:**

**Interface (server):** The interface module of the server is used for initializing the client's IP address, NN model, weights to be used in NN classification and bias vector via using REST API.

**Computations on the server side:** There are two types of computations in the server-side, namely non-interactive computations and interactive computations. The non-interactive computations are realized in the server-side. These non-interactive computations consist of all linear operations (Convolution and FC layers) which are supported by partially homomorphic encryption (PHE). In the interactive computations, the activation function $x^2$ is computed by involving both the server and the client using PHE. Components of the server are as follows:

1. **Non-interactive computation component:** In this component, server makes necessary computations for convolution layer, fully-connected layer and mean pool layer of NN without interacting with the client. All computations realized in this component consist of linear computations only.
2. **Interactive computation components:** This server-side component computes $x^2$ as the approximation of *sigmoid* function.

**Interface (client-side):** The interface module of the server is used for uploading the file to be classified.

**Encryption/decryption component:** This component is responsible for accomplishing encryption and decryption using the public/private key pair of the client.

**Interactive Computation components (client):** This module is used for the interactive computations in the activation layer, in particular the computation of the activation function $x^2$. This module is called interactive since both the server and the client involve into the computation of the activation function.

### *4.1.2.2 Behavioral analysis*

There are three main data flows in this solution: (1) initialization of server; (2) classification of the input; and (3) secure square computation.

1. **Initialization of server:** This operation is accomplished by using *init* REST API call. With this API call, client passes its IP address, NN model to be used in the classification, weight matrix and bias vector to the server as shown in the Figure 11.

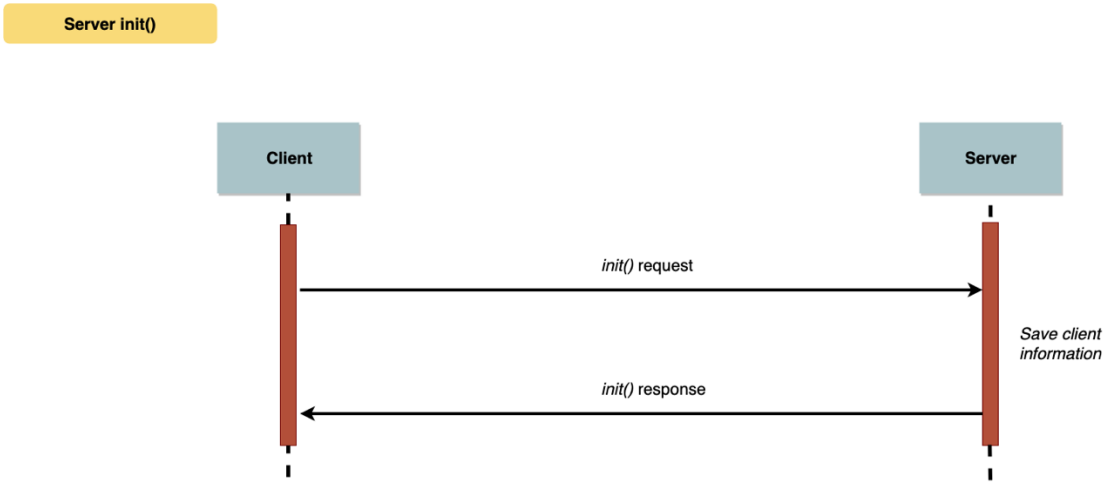



*Figure 11: Server Initialization*

2. **Classification of the input:** The client encrypts the input to be classified and then transmits the encrypted input to the server as shown in Figure 12. Upon receiving the encrypted input, server starts making non-interactive computations. Later, to make interactive computations in the activation layer, server interacts with the client by using REST API calls that are defined on the client. Finally, the result is decrypted by the client.

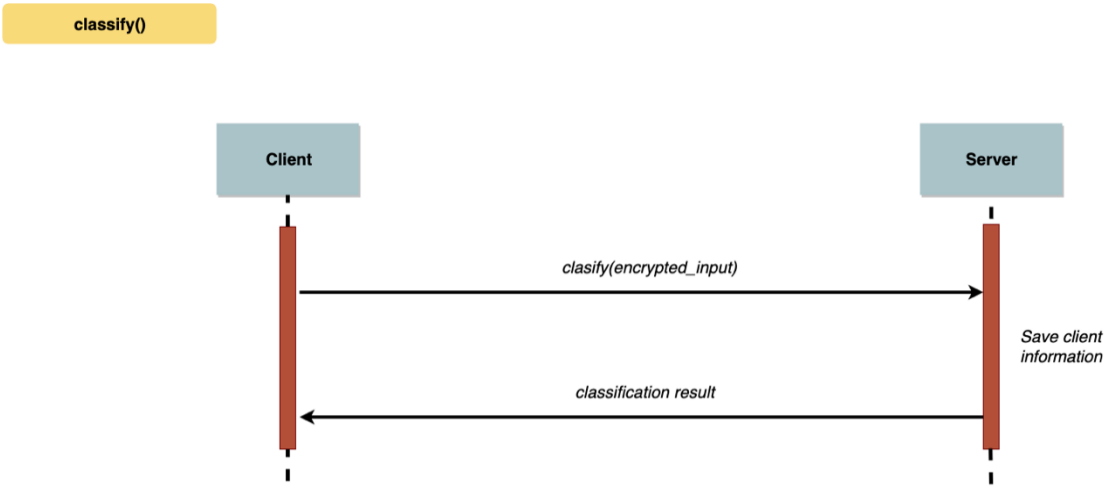



*Figure 12: Classification of the input*

3. **Secure square computation:** In this solution, we propose to use $x^2$ as an activation function. Since most of the partially homomorphic encryption libraries only support addition on the encrypted input, in our approach, we propose the involvement of the client into the

computation of $x^2$. Therefore, server makes an API call to accomplish this computation as shown in Figure 13.


*Figure 13*: *Secure Square Computation*

#### *4.1.2.3 Deployment and configuration*

In this section, we describe the deployment and configuration constraints of the privacy-preserving NN classification based on partially homomorphic encryption.

**Deployment:**
The server-side components and the client-side components will be deployed as docker containers.

**Configuration (Rest API):**
IP address of the client should be initialized in the server-side before executing the classification. In addition to that, the *POST / init* RESTAPI call also used for initializing the NN model, weight matrix and bias vector.

#### *4.1.2.4 Implementation constraints*

- $x^2$ which approximates the sigmoid function is used as an activation function.

#### *4.1.2.5 APIs*

**POST /init**

This API call is used for initializing the server as shown in

Figure 14 in order to initialize client's IP address, weight matrix, bias vector and NN model in the server-side.

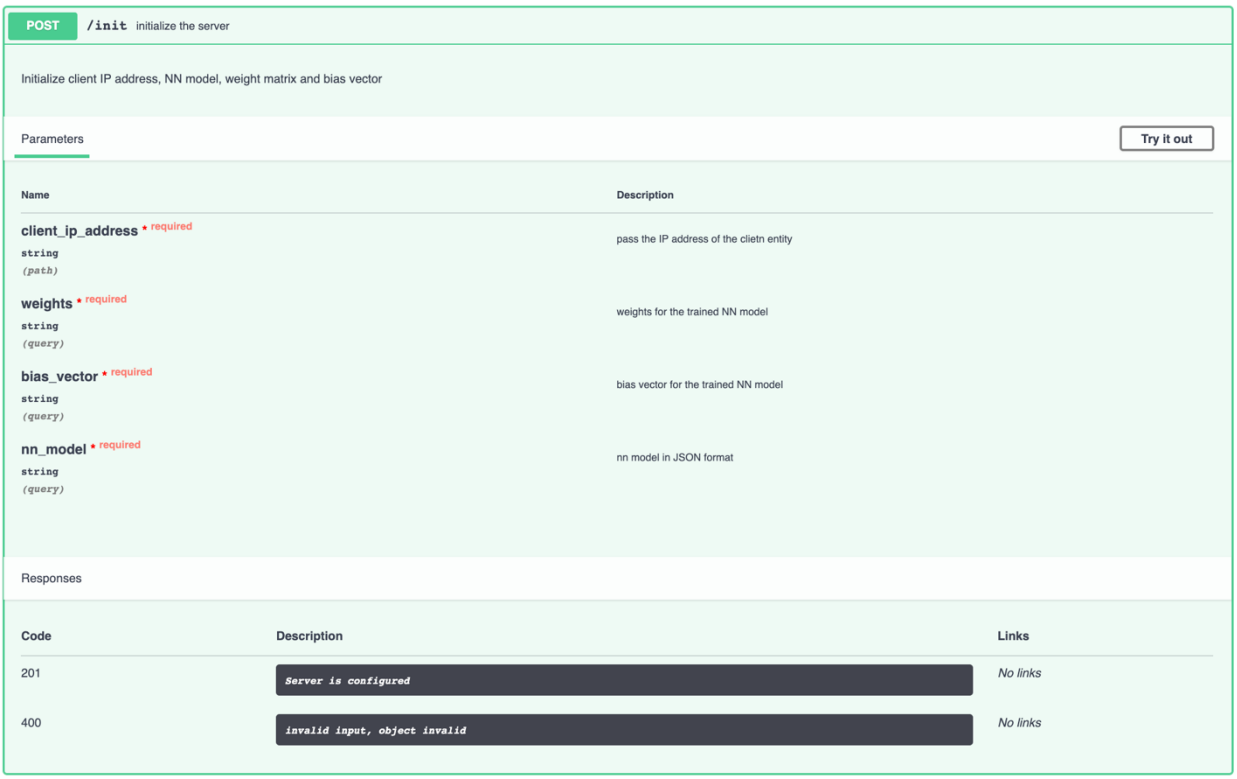



*Figure 14*: *POST / init REST API call*

### POST /classify

This API call is used for classifying the encrypted input and operates as shown in Figure 15.

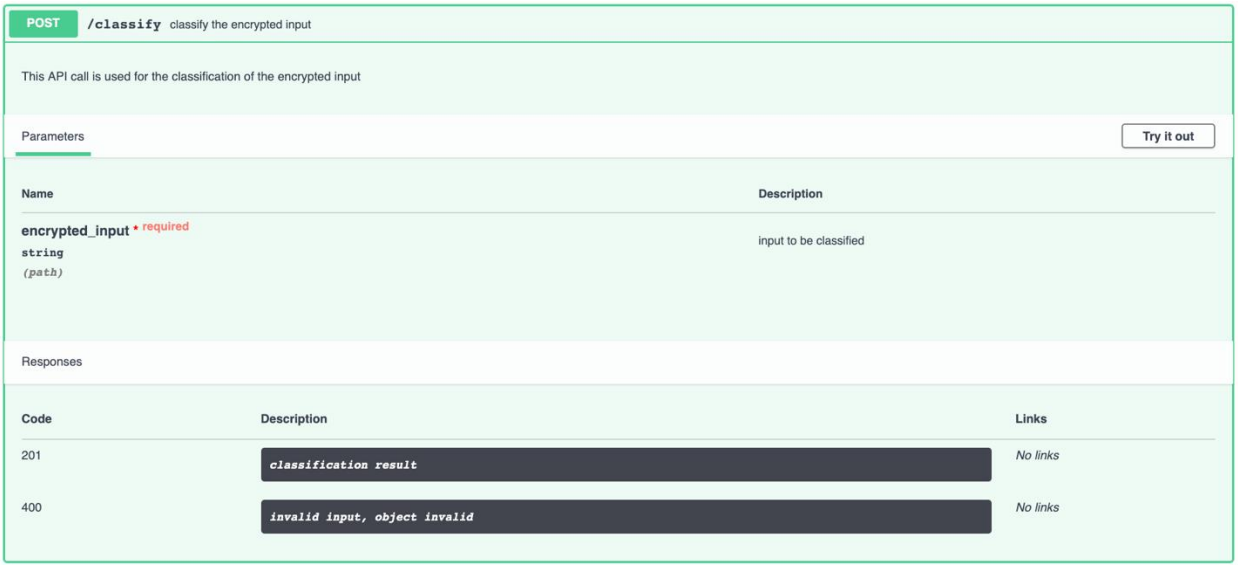



*Figure 15*: *POST / classify REST API call*

**POST /securesquare**

This API call is used for securely computing and operates as shown in Figure 16.

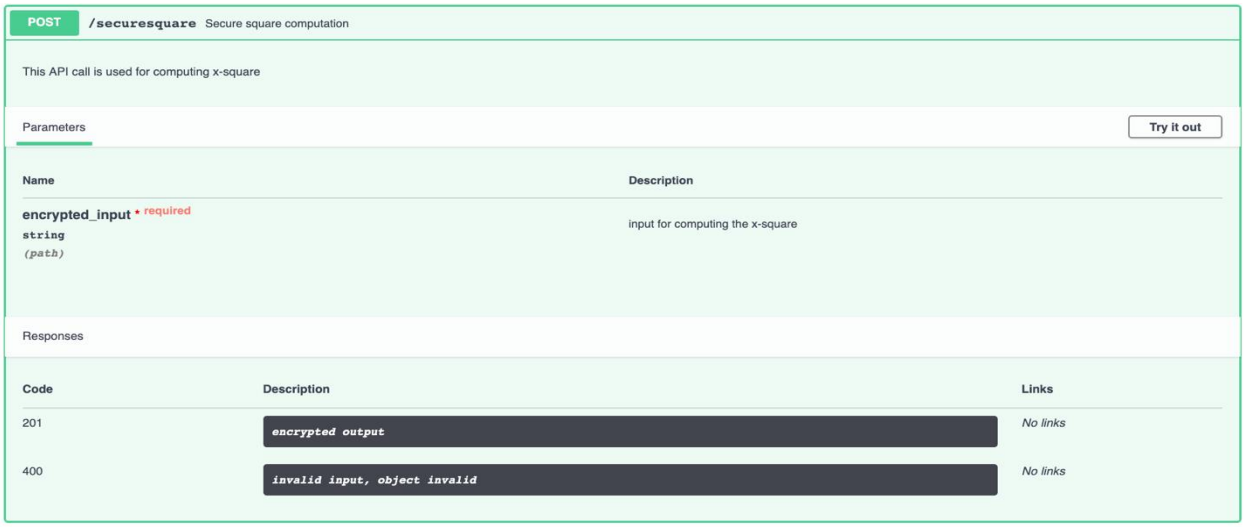



*Figure 16*: *POST / securesquare REST API*

### 4.1.3 Solution based on Fully Homomorphic Encryption

This solution uses Homomorphic Encryption (HE) in order to build a privacy-preserving neural network inference solution. This solution uses the CKKS scheme [6] implemented in Microsoft SEAL library [10] .

#### *4.1.3.1 Main components and their relationships*

This solution consists of three main components:

1. Server-component.
2. Company-side component.
3. Client-side component.

A company provides a service of privacy-preserving neural networks inference to its clients using the PAPAYA platform hosted on a server.

**Server-side component:**

The server-side component consists of 3 modules (Figure 17):

1. **SEAL**: the homomorphic encryption library that provides basic homomorphic operations on ciphertexts.
2. **NN**: this module provides methods to evaluate neural networks using homomorphic operations.

3. **Interface**: this module provides a REST API of the server-side component.

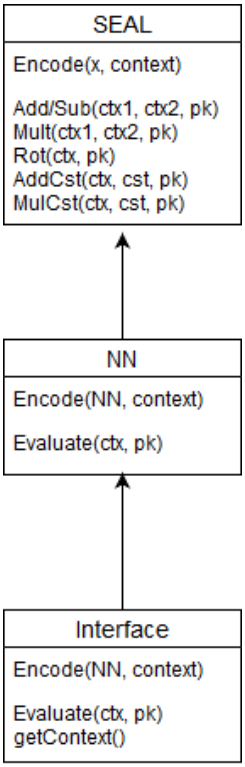



*Figure 17: Topology of server-side modules*

**Company-side component:**

The company-side component consists of one module (Figure 18):

1. Agent interface: this module provides a REST API of the company-side component; it will deal with all the communications with the server.

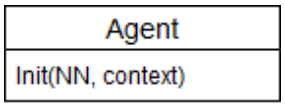



*Figure 18: Topology of company-side modules*

**Client-side component:**

The client-side component consists of 3 modules (Figure 19):

1. **SEAL**: will be used to generate the keys, encrypt and decrypt data.

2. **Ext_libs**: will provide the necessary methods to process data before encryption and after decryption. The methods will change depending of the use-case.
3. **Agent**: will provide the REST API of the agent and deal with communications to the server.

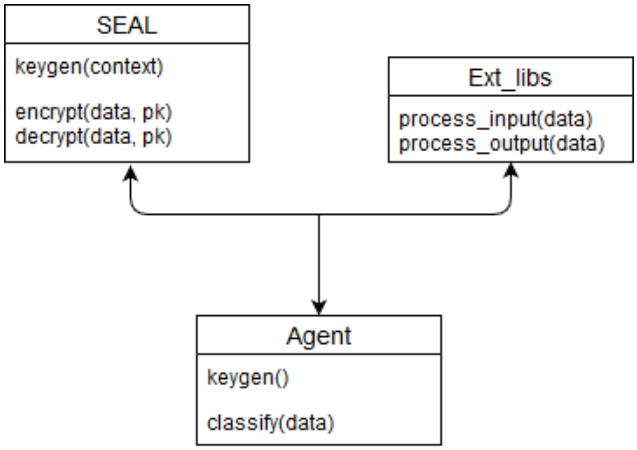



*Figure 19: Topology of client-side modules*

**Detailed description of the modules:**

**SEAL**

Microsoft SEAL is an easy-to-use homomorphic encryption library developed by researchers in the Cryptography Research group at Microsoft Research. Microsoft SEAL is written in modern standard C++ language. The library implements the BFV scheme [5] and CKKS scheme [6]. This module is used on both the client side and the server side. On the client side, the module is only used for the generation of the secret key and public key and for encryption and decryption procedures. On the server side the module is used to perform the homomorphic operations.

**Ext_libs**

Depending on the use case, it may be needed to perform some preliminary computations (e.g. cleaning, normalization …), or some final computations (e.g. max, softmax functions …). The Ext_libs module packages all the necessary methods.

**NN**

This module packages the basic routines of the inference of a neural network, and the methods to encode a neural network to make it works with homomorphic encryption. This module is responsible for:

1. Initializing the network by encoding its weights and saving it.
2. Evaluating the different layers (convolution, dense …) of a network using the SEAL module.

#### *4.1.3.2 Behavioral Analysis*

There are three main flows in the framework: (1) model initialization, (2) keys generation, and (3) classification. Next the detailed description of each of them.

1. **Model Initialization (**Figure 20**).** In this phase, the company wants to upload the neural network model to the platform. We assume that the NN model is already trained and modified for homomorphic encryption (e.g, replaced non polynomial functions by polynomial approximations). Along with the model, the company transmits the context variable that describes the homomorphic encryption parameters that will be used by the platform. Therefore, when the agent's *init* method is called by the client application with the NN architecture and the context (Step 1), the agent calls the server and sends the model (Step 2). Upon the model reception, the interface invokes the *init* method of the NN module (Step 3). The NN module creates a new network by encoding the network weights into plaintexts with SEAL, based on the parameters defined in the context variable (Step 4).

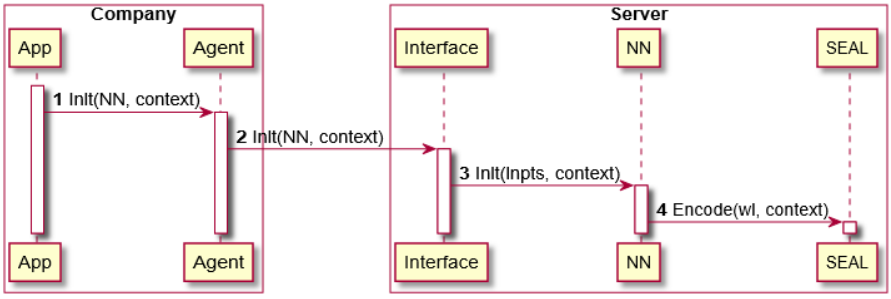



*Figure 20: Sequence diagram of the initialization*

2. **Key generation** (Figure 21)**.** In this phase, the client wants to generate the keys to use the NN service. The client application calls the Agent (Step 1), which checks whether the context data is cached. If the context data is cached, the agent loads this information. Otherwise, agent requests the server interface with the *getContext* method (Step 2), which sends back the context data (Step 3). Then the agent calls the *keygen* method of the SEAL module (Step 4) which generates the secret key (*sk*) and the public key (*pk*), based on the provided context

information (Step 5). These keys will be used to encrypt the data to classify and decrypt the result of the classification.

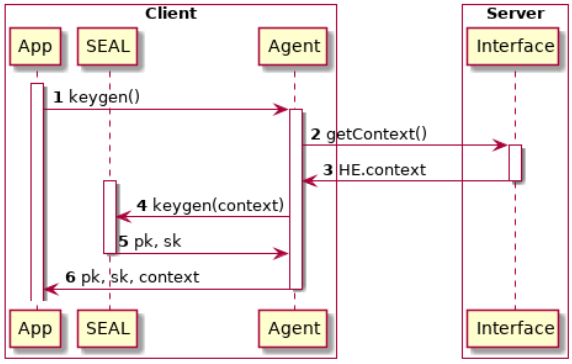



*Figure 21: Sequence diagram of the key generation.*

3. **Classification (**Figure 22**)**. The classification phase is more complex. The client application calls the *classify* method of the agent with the raw data to classify (Step 1). Depending of the use case, the raw data have to be processed before encryption. Therefore, if needed, the client agent calls the *process_input* method of the *Ext_libs* module (steps 2 and 3). Once processed, the agent calls the *encrypt* method of the SEAL module (Step 4), that sends back an encryption of the data (Step 5). The agent can then call the *classify* method of the server interface with the encrypted data as inputs (Step 6). Upon reception of the encrypted data, the server interface invokes the *evaluate* method of the NN module (Step 7) that homomorphically evaluates the neural network, with the help of the SEAL module to perform the requested homomorphic operations. Once the computations are done, the interface sends back the encrypted result to the client agent (steps 12 and 13). Upon reception of the encrypted result, the client agent calls the SEAL module to decrypt the result (steps 14 and 15). If needed, the agent calls the *process_output* method of the *Ext_libs* module (steps 16 and 17) to finalize the computations (e.g. computations of max, softmax …). Finally, the agent sends the final result of the classification to the client application (Step 18).

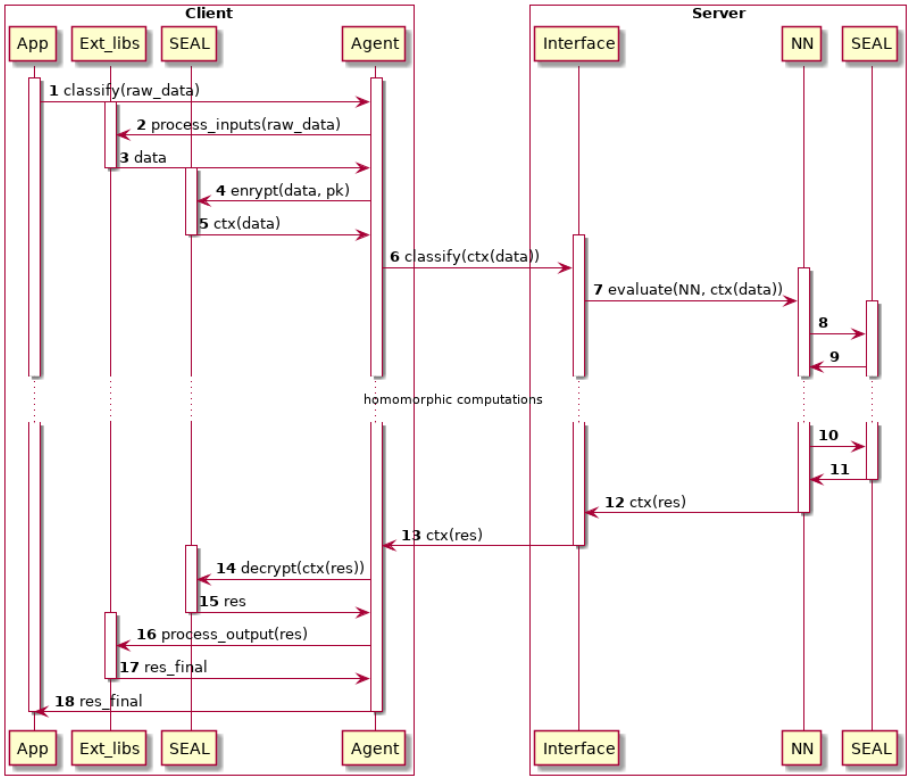



*Figure 22: Sequence diagram of the classification phase.*

### *4.1.3.3 Deployment and configuration*

**Deployment:**
The server-side and client-side components will be deployed as Docker containers. The user will use the PAPAYA dashboard to manage the deployment of containers.

**Configuration:**
A dedicated configuration file will set all the parameters the service expects to receive, the agent needs to know server IP/URL for REST_API.

### *4.1.3.4 Implementation constraints*

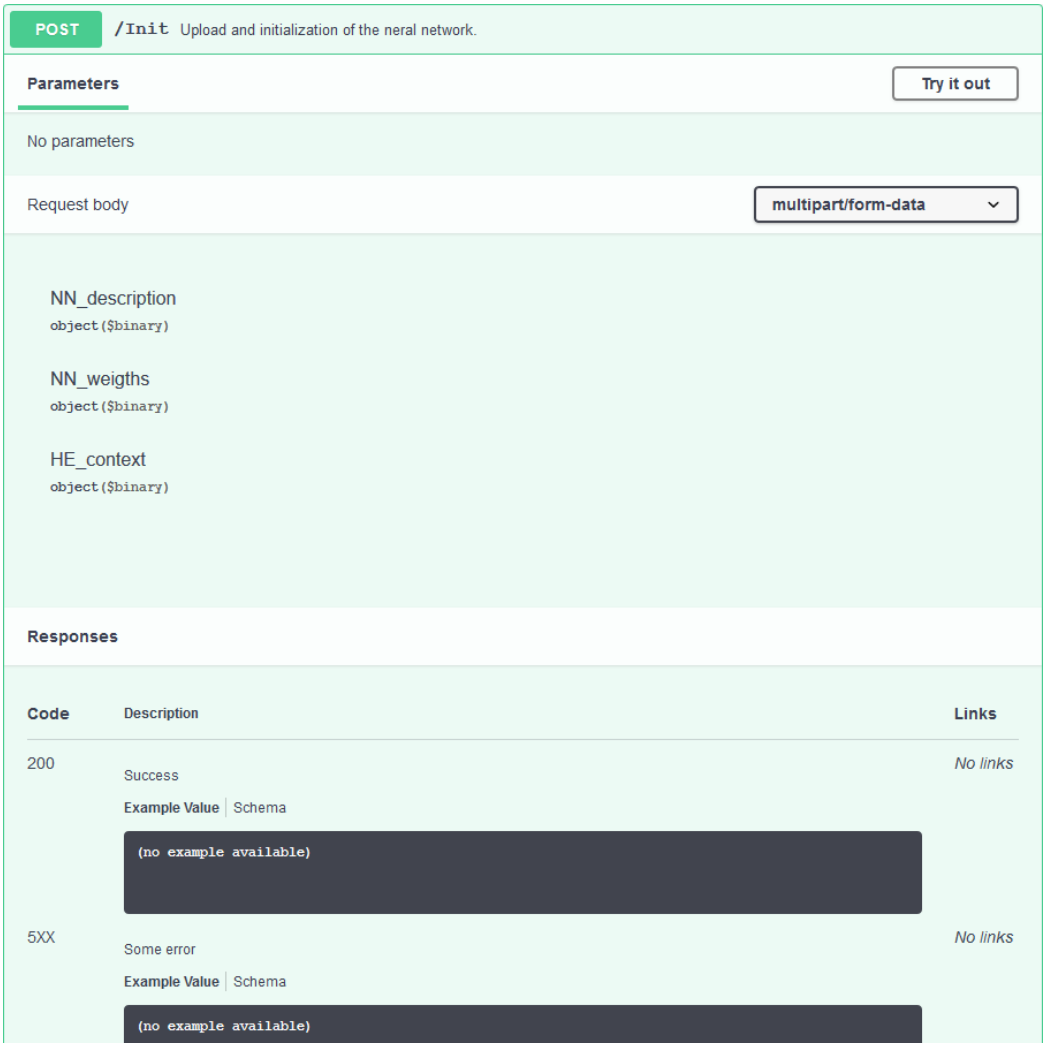
At the moment, the current version expects to receive as input a pretrained neural network as two files. One JSON file describing the network architecture and one h5 file containing the weights. The files will be uploaded to the server during the model initialization phase using the REST API.

### *4.1.3.5 APIs*

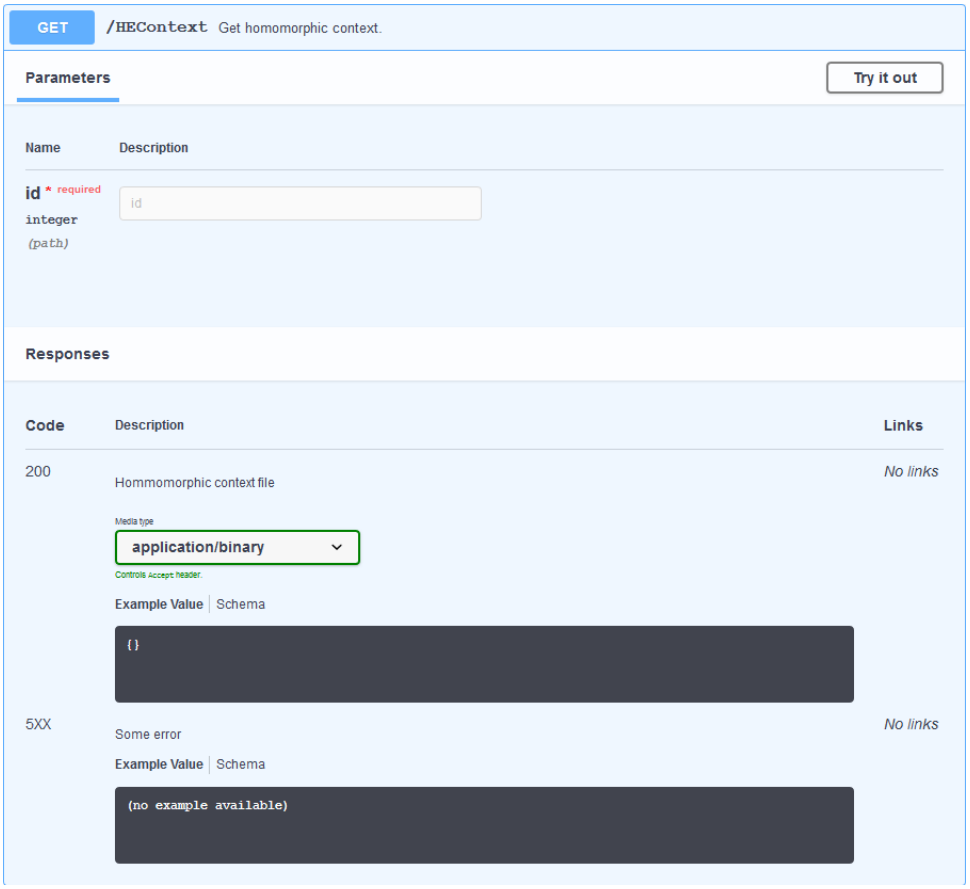
Server API:

Description of the server-side REST API.

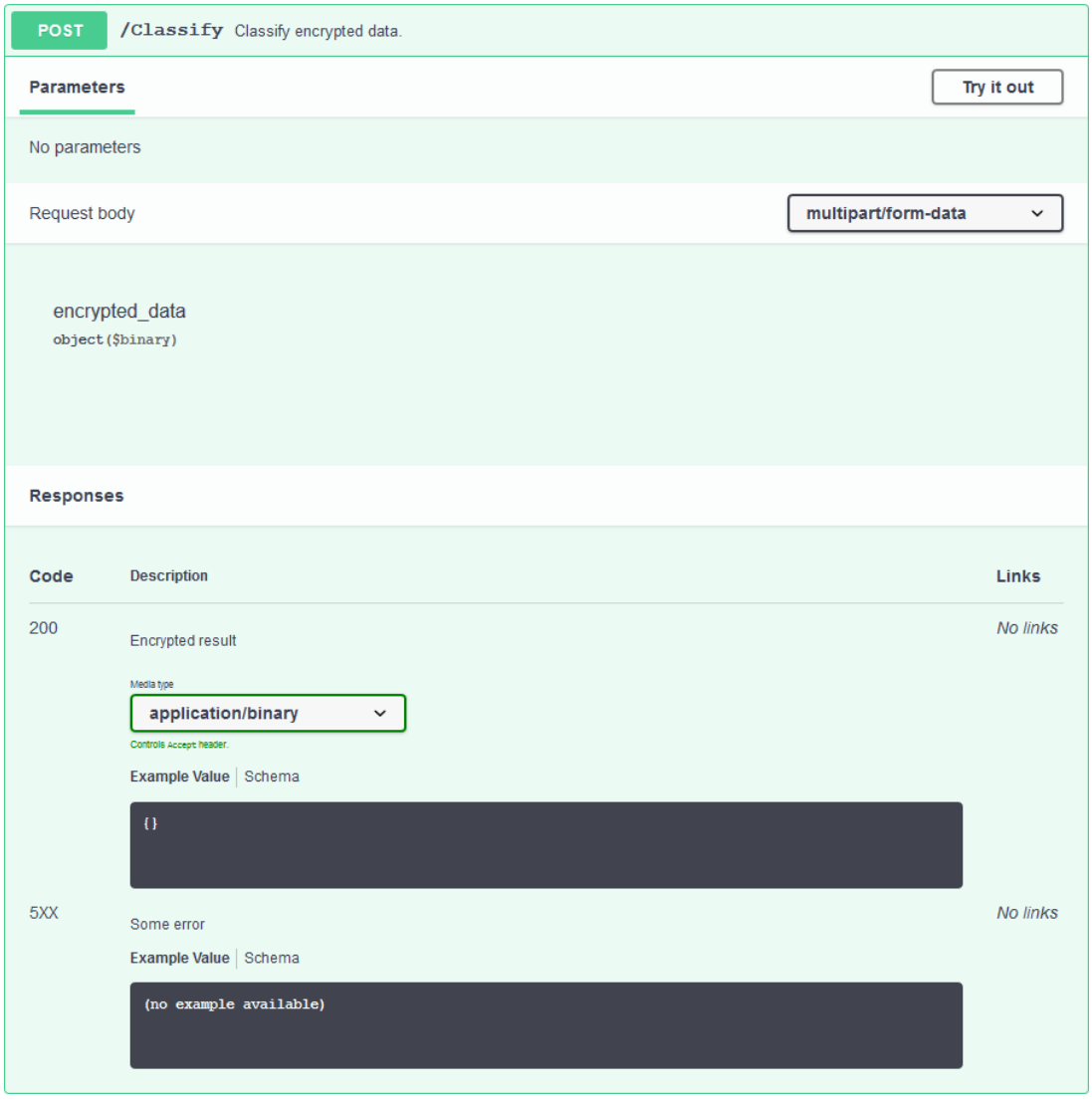
**POST/Init**



## GET/HEcontext



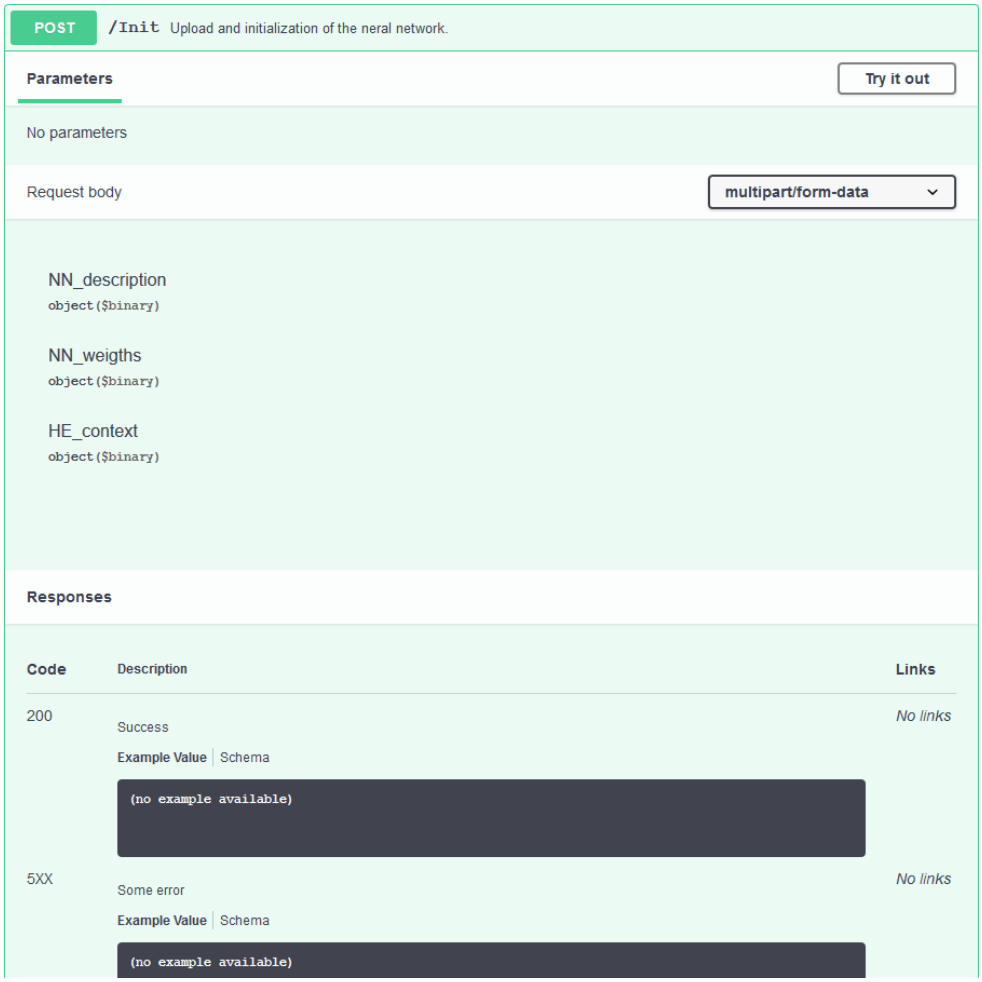
## POST/Classify



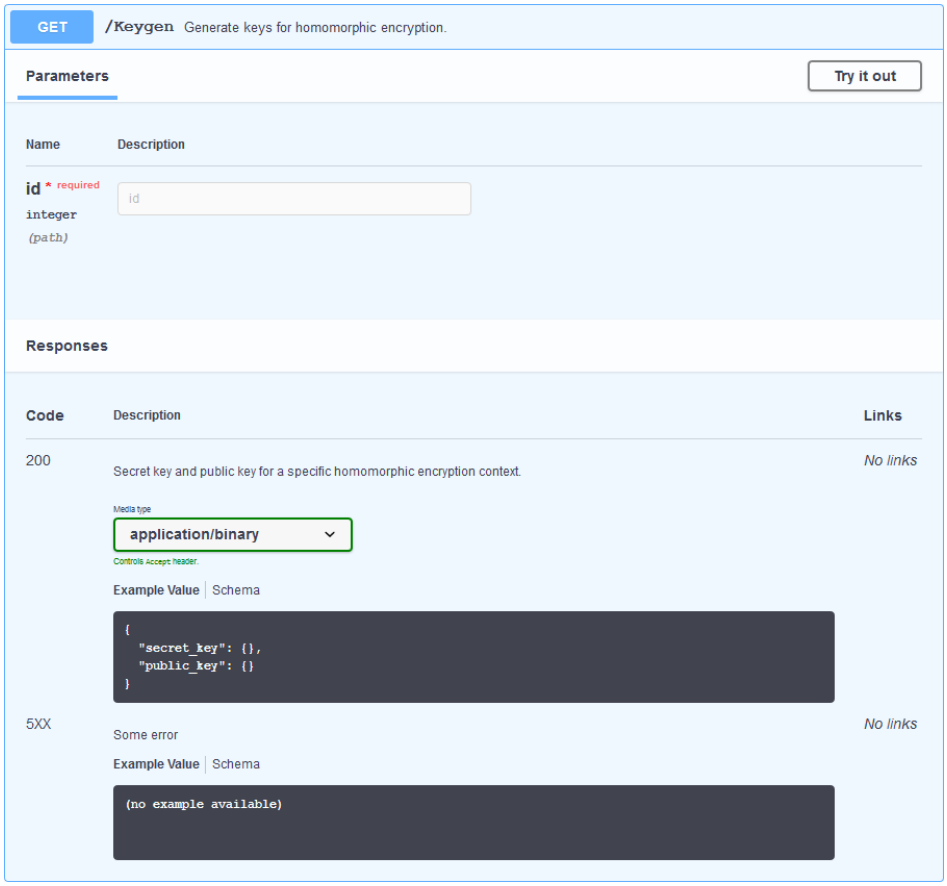
Client API:

Description of the client-side REST API.

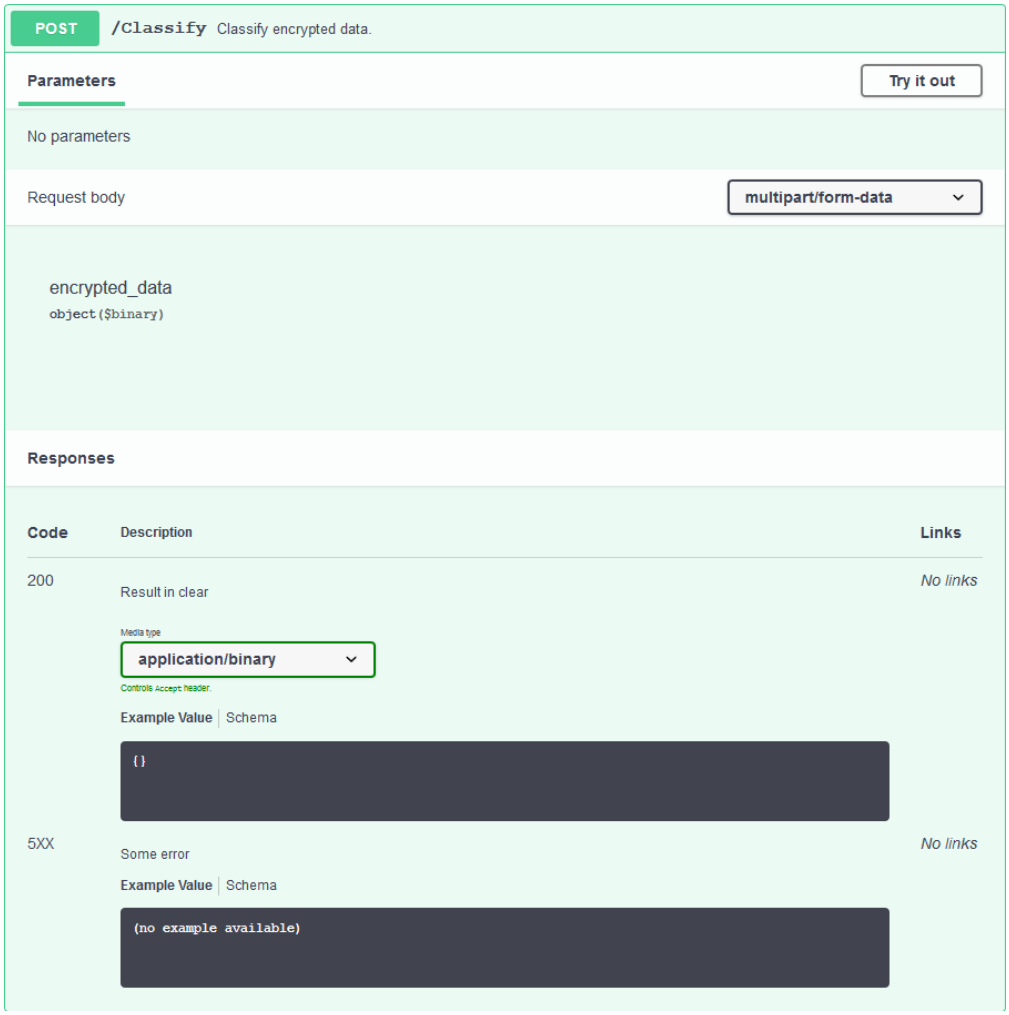
**POST/Init**



## GET/Keygen


GET
/Keygen
Generate keys for homomorphic encryption.
Parameters
Try it out
Name
Description
id * required
integer
(path)
id
Responses
Code
Description
Links
200
Secret key and public key for a specific homomorphic encryption context.
No links
Media type
application/binary
Controls Accept header.
Example Value
Schema
{
"secret_key": {},
"public_key": {}
}
5XX
Some error
No links
Example Value
Schema
(no example available)

**POST/Classify**




### 4.1.4 Privacy-preserving NN classification based on hybrid approach

The hybrid solution uses both, the HE and 2PC, in order to build a privacy-preserving NN classification framework and maximize the efficiency of classification on deep NN. The solution is *practically* generic, namely, it supports different types of DNN (i.e., MLP, CNN, and RNN) with any number of layers, any number of neurons in each layer and any activation function (from the set of supported activation functions), while the performance still acceptable (grows linearly with the DNN's depth).

*4.1.4.1 Main components and their relationships*

The solution consists of two main components:

1. Server-side component
2. Client-side component (Agent)

**Server-side component:**

The Server-side component consists of seven modules:

1. HElib [11]: the homomorphic encryption library which provides the basic capability to encrypt and decrypt homomorphically. In addition, it allows executing operations on ciphertexts, such as addition, multiplication and rotation.
2. Linear algebra module: the module allows the server-side component to compute the linear layers of the NN (i.e., convolution layers, average pooling, fully connected layers, and GRU layers to support recurrent NN)
3. Garbling module: We use Yao's garbling circuits [12] approach to compute the activation functions. Specifically, we use JustGarble [13] as a garbling library. In our case, the client will garble the circuit, and the server will evaluate it to obtain the garbled result. Only the client can map between the garbled result and the real values.
4. LibOTe [14]: this library will be used to execute oblivious transfer [15] while computing the activation functions.
5. 2PC module: runs 2PC protocol between the server and the agent. This module combines garbling module and libOTe to create a full secure multiparty computation functionality. This module will be used mostly to compute the activation functions privately and without revealing the intermediate values to any party.
6. NN module: this module will represent the privacy-preserving version of the original neural network. It will be able to compute the mathematics behind the NN in a privacy-preserving manner. The module will receive as input the NN architecture and the network weights and will construct the corresponding NN model in a generic way.
7. NN Interface module: this module provides the REST API of the server-side component and invokes all the previously mentioned modules.

Figure 23 illustrates the topology of the server-side component.

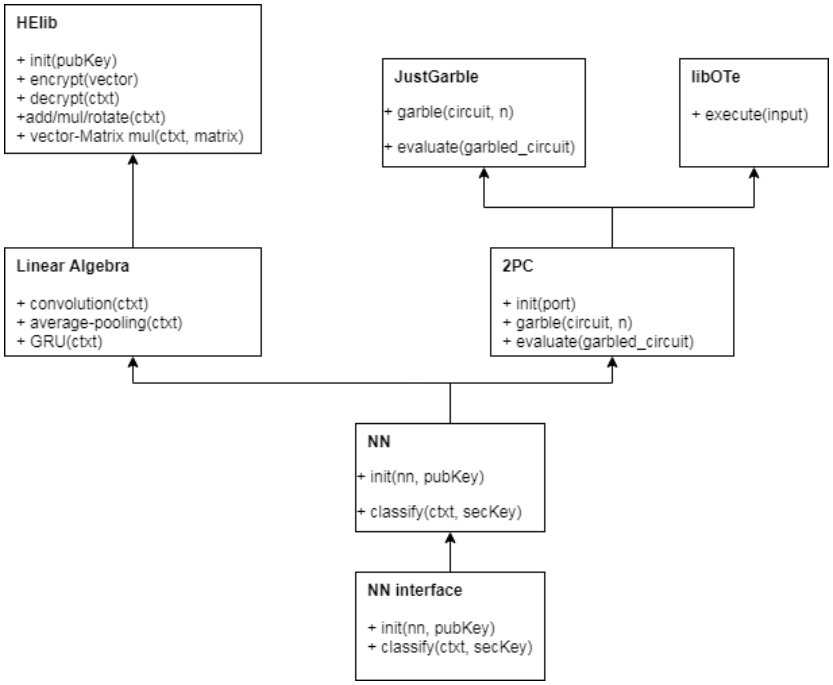



*Figure 23: Server-side components – topology*

**Client-side component:**

The client-side component consists of three modules:

1. HElib module: it will be used to encrypt the input and to decrypt the server-side component's output. Also, it will be used to generate the keys.
2. 2PC module: it is responsible for secure two-party computation. This module will use JustGarble and libOTe. Upon receiving a noised ciphertext, the client decrypts the ciphertext, to obtain a noised message. Then it garbles the circuit which reduces the noise and computes the activation function, and evaluate the circuit with the server-side component.
3. Agent Interface module: this module provides the REST API of the agent and manages cryptographic protocols with the server-side component. The agent interface provides the following functionality:
    a. GenKeys: generate keys for the homomorphic encryption.
    b. Init: receives several parameters (i.e., NN architecture, server credentials, FHE keys) and configures both, the agent and server.
    c. Classify: receives a plaintext vector and keys and runs a protocol with the server to classify the input vector in a privacy-preserving manner.

Figure 24 illustrates the topology of the client-side component.

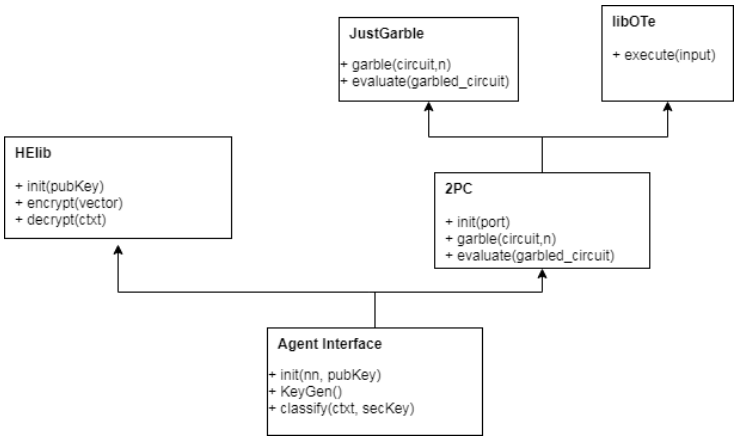



*Figure 24: Client-side component – topology*

**Detailed description of the modules (for both components):**

**HElib**

HElib is a very well-known and open source library. The library implements the BGV scheme [3] with SIMD (single instruction multiple data) property [16]. The library provides the following main functionalities:

1. Encrypt: the encrypt operation receives a plaintext (long) vector and convert it to a ciphertext vector.
2. Decrypt: the decryption operation receives an encrypted vector and decrypts it to the corresponding plaintext vector.
3. Addition/multiplication/rotation: the library allows basic operations on ciphertext vectors. We use mostly three of them: addition, multiplication and rotations.
4. Vector-Matrix multiplication: HElib provides basic linear algebra operations, specifically, vector-matrix multiplication.

**Linear Algebra**

The linear algebra module will implement all linear algebra needed for neural network classification on ciphertext. In fact, there are will be many instances of Linear Algebra module, each one responsible for the specific layer type. In the first version of the system, we will support convolution, pooling, and dense layers. In the next version we plan to add support for GRU layers to deal with RNN networks.

**JustGarble (from Gazelle)**
We will use JustGarble library for garbling and evaluating garbled circuits. In our implementation we will use the modified version of the library which Gazelle [17] used in their system.

**libOTe**
libOTe [14] is one of the most efficient oblivious transfer implementations we could find. It supports various oblivious transfer protocols. We are going to use a 1-out-of-2 semi honest protocol [18].

**2PC**
This module uses both, the JustGarble and libOTe to provide a full secure multiparty computation module. The four main functionalities of this module are: (1) circuit garbling; (2) circuit evaluation; (3) execution of oblivious transfer protocol; and (4) interactive protocol to compute a circuit privately. It uses sockets to communicate between the server and the client. In more details, the main functionalities of the 2PC module are as follows:

1. **Garble (will be running on the client side):** receives a circuit as input and garbles it. We will provide the system with implementations of few activation functions (i.e., ReLU, tanh and sigmid) for a single neuron. JustGarble can take a circuit for a single neuron and extend it to any number of instantiations. This feature is very useful and allows us to compute activation function for any number of neurons and help the system to be more generic.
2. **Evaluation (will be running on server side):** receives a garbled circuit and evaluates it to compute the real output of the function.
3. **OT:** executing oblivious transfer on the inputs as part of 2PC algorithm.
4. Communication between the server and the client.

**NN**
The neural network (NN) module will encapsulate the neural network functionality. It will contain the neural network architecture, configuration, and combine all the previously mentioned modules to provide a complete and efficient privacy-preserving neural network classification framework. The module will be responsible for the following tasks:

1. Create and initialize appropriative Linear Algebra module for each layer of the network.
2. Initialize the 2PC module. The NN will provide the 2PC component with port number on which it will be listening for connection request from the 2PC module running on the client-side.
3. Initializing HElib with correct parameters and create HE context.
4. Implementing switching protocol to convert between FHE and 2PC without revealing any intermediate values. The protocol adds input noise to the ciphertext, to prevent revealing the intermediate values to the agent. Also, it adds output noise to the result of 2PC (computing the activation functions). The server, removes the noise after receiving the encrypted result from the agent, see Figure 27 for more details.
5. To support persistency, the module has the ability to save and load the neural network. In case the server is down, the module can reload itself. We will use HDF5 format to store or load a neural network weights and JSON format to store/load network architecture.

Note that HElib and 2PC are shared modules in both, the server-side and the client-side components. However, the functionality that used may differ between the components. For example, the client will use the garbling functionality of 2PC, while the server will use the evaluation functionality.

#### *4.1.4.2 Behavioral Analysis*

There are three main flows in the framework: (1) key generation; (2) system initialization; and (3) vector classification. Following is a detailed description of each of them.

1. Key Generation (keyGen): the flow involves a client app which calls the agent to create homomorphic encryption keys for HElib. The client app is responsible for managing the keys (they will be used for system initialization and vector classification). Figure 25 illustrates the *key generation* process.

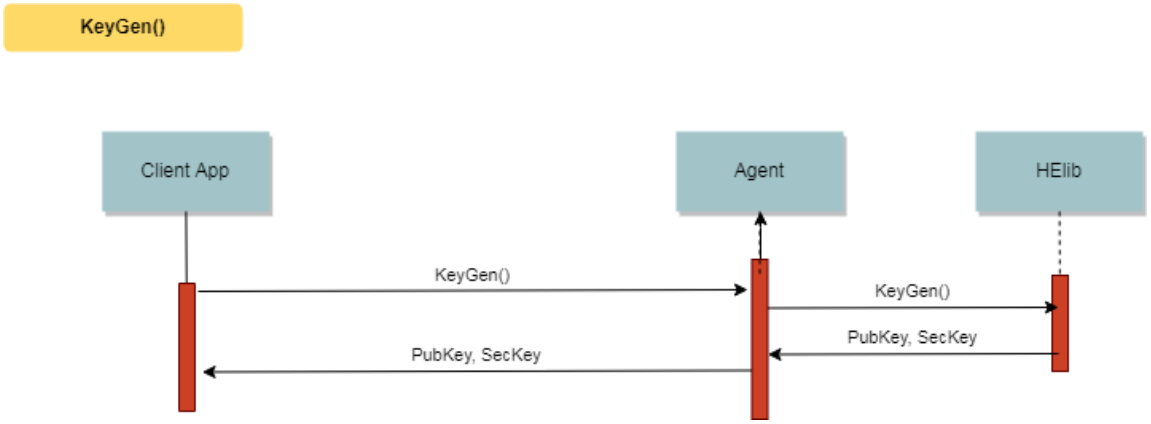



*Figure 25: Key generation – sequence diagram*

2. **System initialization:** Figure 26 describes the sequence diagram of the initialization phase. When the agent's *init* method is called with NN architecture, matrices weights (depicted by h5 in the Figure 26) and HE keys as parameters, the agent saves the network architecture for future use (i.e. calculation of activation functions for each layer), initializes HElib to create HE context, and calls the server's *init* method (NN interface) with the NN architecture, weights and the public key. The NN interface, in turn, calls *init* method of the NN module. The NN module saves both, the network architecture and the network weights (for feature use),

provides the 2PC with port number to listen on (the port number set while configuring the system), and initializes HElib.

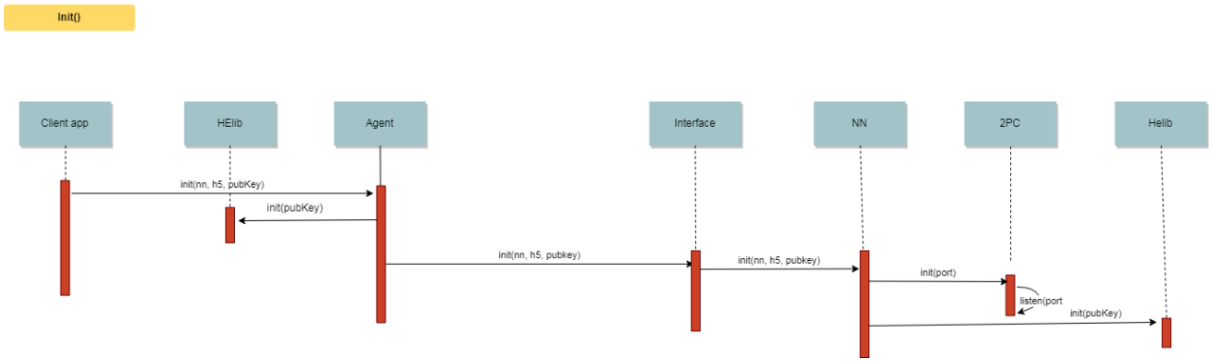



*Figure 26: Initialization - sequence diagram*

3. **Vector classification:** the classification procedure is much more complicated. Upon calling the agent's 'classify' method with plaintext vector, the agent encrypts it using HElib and sends it to the server. The server iterates the layers one by one (in NN module). For each layer there are two main parts:
    a. **Computing the layer:** the server creates the current layer object with corresponding weights (e.g. convolution, FC) and computes the linear algebra part of the layer.
    b. **Computing the followed activation function (if exists):** the server adds noise (input noise, chosen randomly over the integer group *P* – FHE parameter; for details see D3.1 [2]) to the (linear algebra) result and sends the encrypted vector (with noise) to the client. The client decrypts it. In case of computing activation function, the client garbles the corresponding circuit (for the entire layer: it keeps pointer to the current layer, it knows the number of neurons in this layer and the type of activation function from the network architecture, and it is provided with implementation of specific activation functions for a single neuron), sends it to the server and execute OT protocol to exchange the inputs. The circuit reduces the input noise (added by the server), calculated the activation function, and adds noise to the output (on the server side). The server evaluates the circuit to obtain the garbled output (with the output noise) and sends it to the client. The client maps the garbled output to the noised output, decrypts it and sends it again to the server to reduce the noise. Finally, the server reduces the noise.

Figure 27 depicts the sequence diagram of the procedure of computing one layer in details. Computing the rest of the layers is the same. For simplicity, we mention the 2PC module only (in practice, it uses JustGarble and libOTE modules internally).

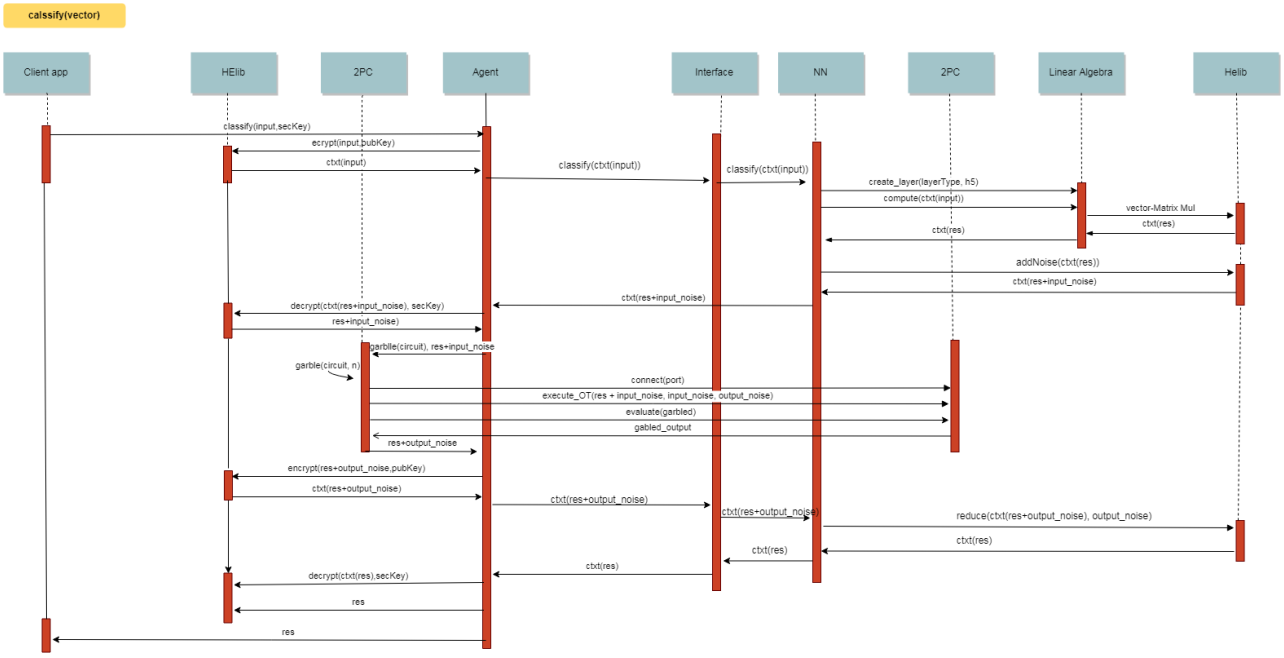



*Figure 27: Classify vector - sequence diagram*

### *4.1.4.3 Deployment and configuration*

This section describes the deployment and configuration phases.

**Deployment:**
Both, the server-side component and the agent-side component will be deployed as Docker containers (see Section 8 for details). The user can use PAPAYA dashboard in order to deploy the containers.

**Configuration:**
It will be a dedicated configuration file for all the parameters the service expects to receive

- Agent needs to know server IP/URL for REST_API.
- Server needs to know agent IP and port to use sockets for 2PC (agent could pass as params in init)

The mentioned requirements (server/agent IP's & URL's), will be provided to the agent in the configuration phase, before the deployment.

### *4.1.4.4 Implementation constraints*

Following is a list of implementation constraints/limitations of current version:

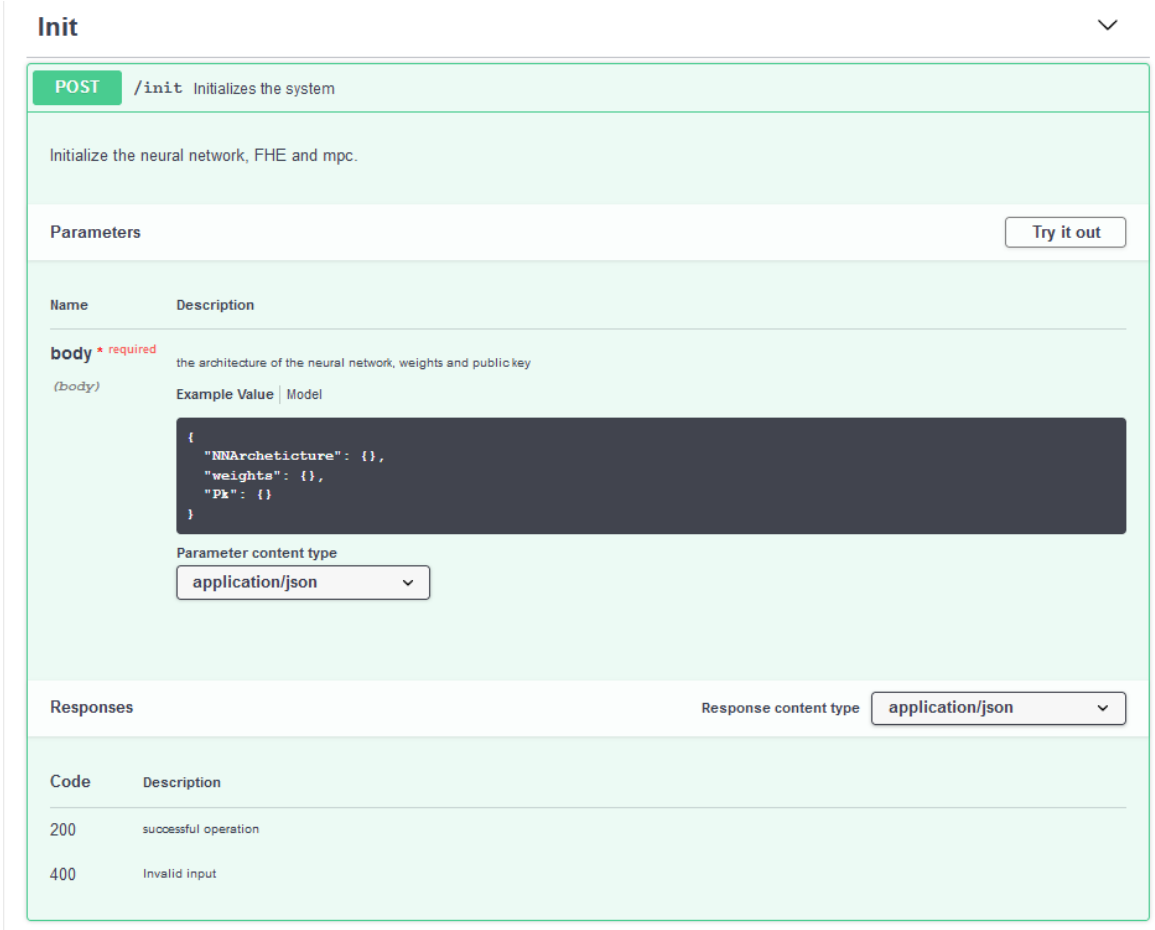
1. The service expects to receive as input a pretrained neural network. This network could be trained using any framework providing this functionality (e.g., Keras [19]). The network architecture should be saved in JSON format while the network weights should be saved in h5 format. Both files will be transferred to the server-side component during initialization using REST API.
2. To make our implementation generic (in terms of supporting different types of neural networks with different architectures), we need to support different type of layers and different activation functions. The first version of the system will support convolution, pulling and fully connected layers, and ReLU, tanh, and sigmoid activation functions. More layers and more functions will be supported in the next version if required.

#### *4.1.4.5 APIs*

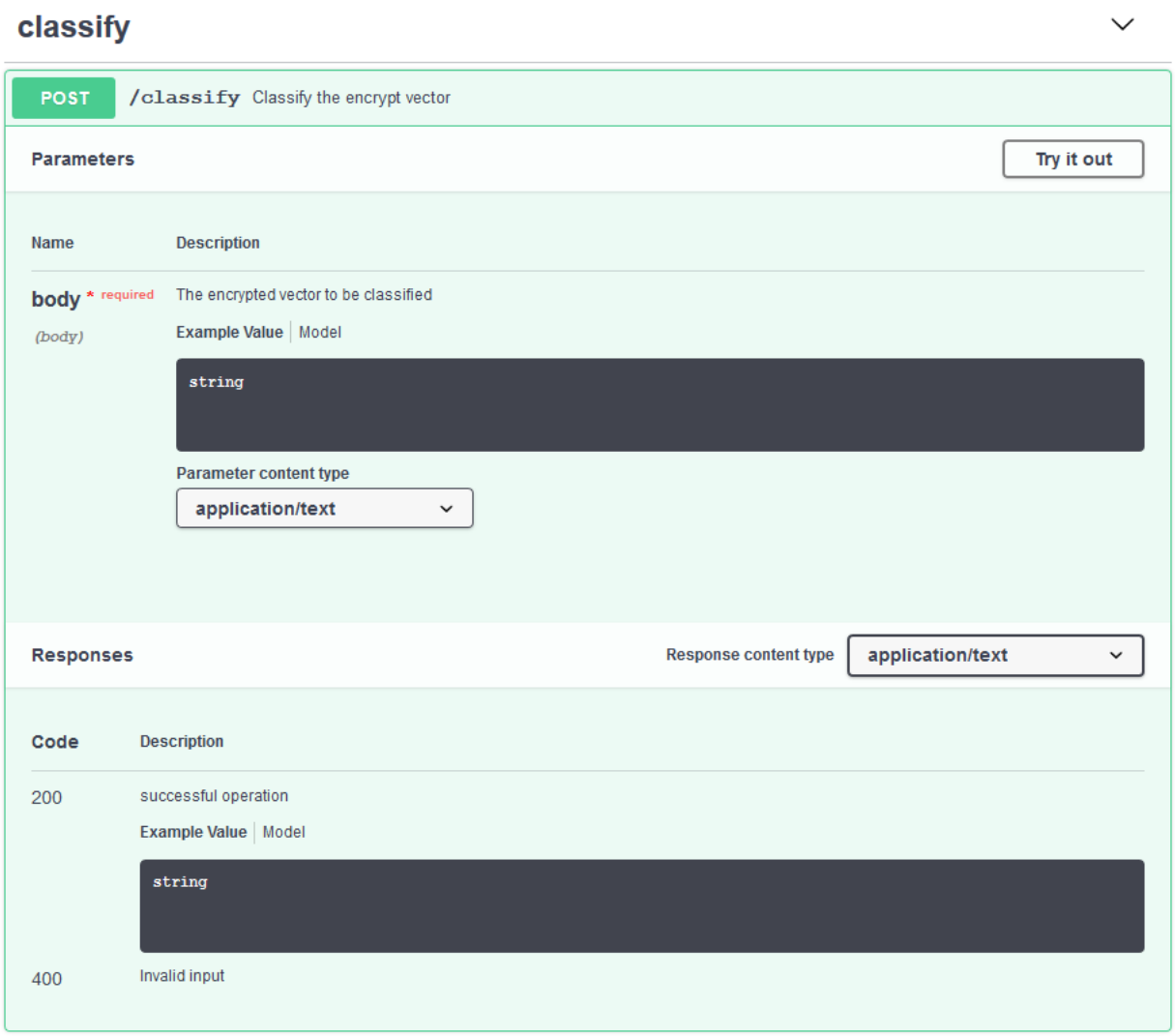
Server API:

There are two REST API calls supported by the Server-side component: (1) *Init;* and (2) *Classify*. The *Init* call configures the server and creates the required model. The *Classify* call receives as input an encrypted vector and classifies it. Following is a detailed description of the calls.

## POST /init


Init
POST
/init
Initializes the system
Initialize the neural network, FHE and mpc.
Parameters
Try it out
Name
Description
body * required
(body)
the architecture of the neural network, weights and public key
Example Value
Model
{
"NNArchetiсture": {},
"weights": {},
"Pk": {}
}
Parameter content type
application/json
Responses
Response content type
application/json
Code
Description
200
successful operation
400
Invalid input

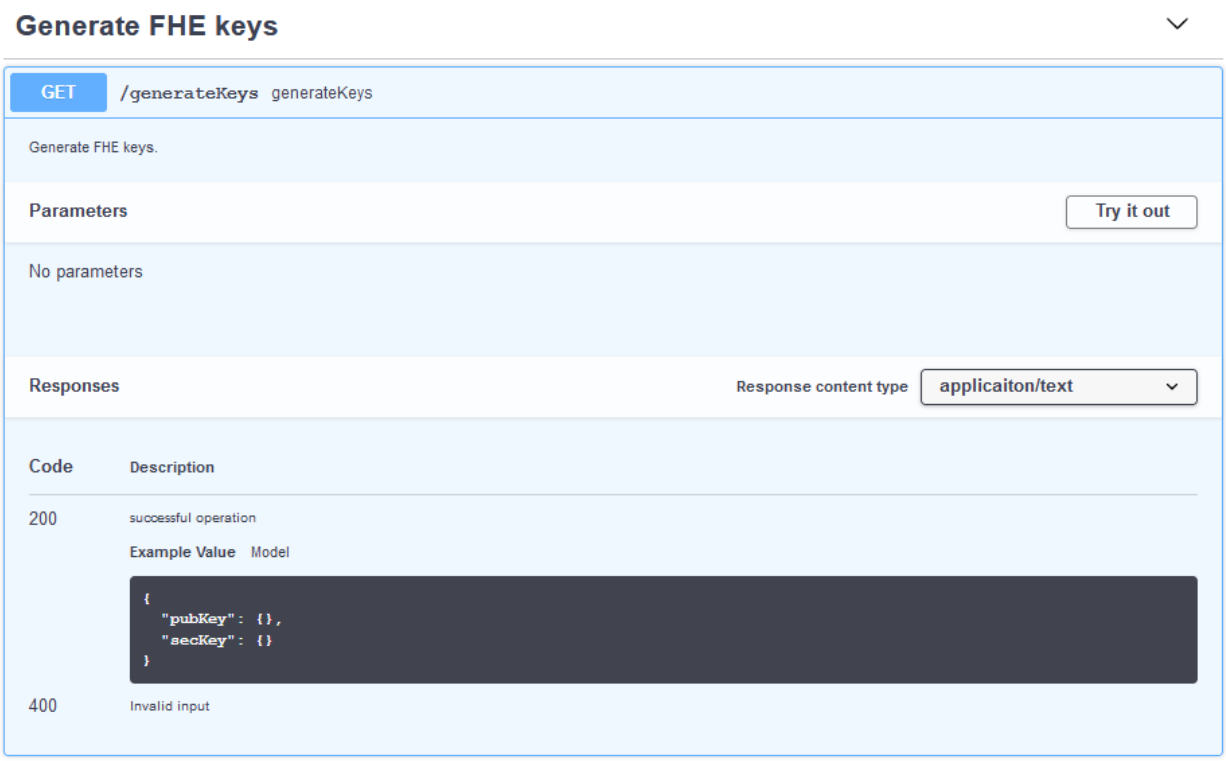
**POST /classify**



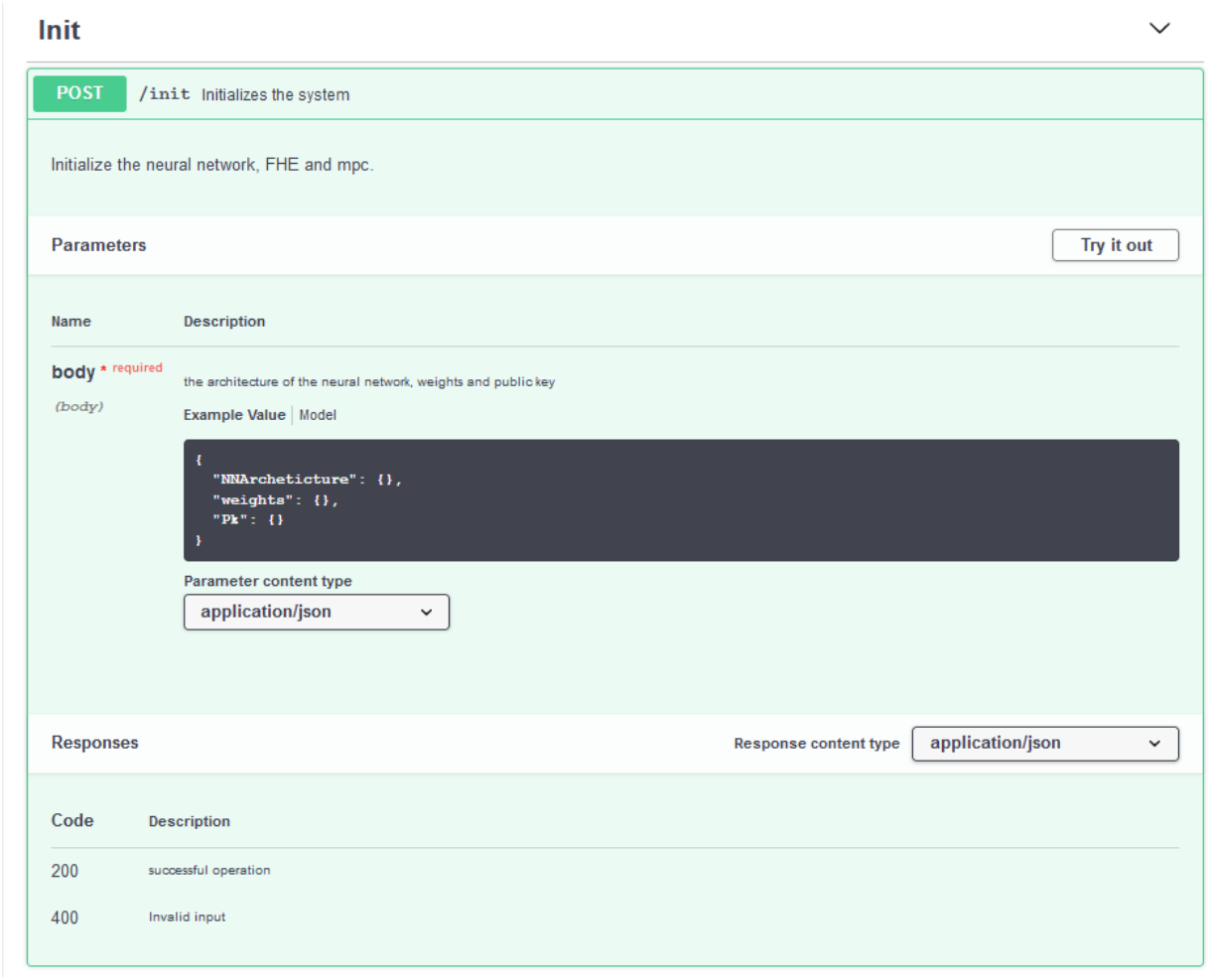

Client API:

There are three REST API calls supported by the Client-side component: (1) *GenerateKeys*; (2) *Init;* and (3) *Classify*. The *GenerateKeys* call will generate public and secret keys for homomorphic encryption. The *Init* call configures the client-side component and sends configuration parameters to the server-side component. The *Classify* call receives as input an encrypted vector, sends it to the server side-component for classification and receives the encrypted result. Following is a detailed description of the parameters:

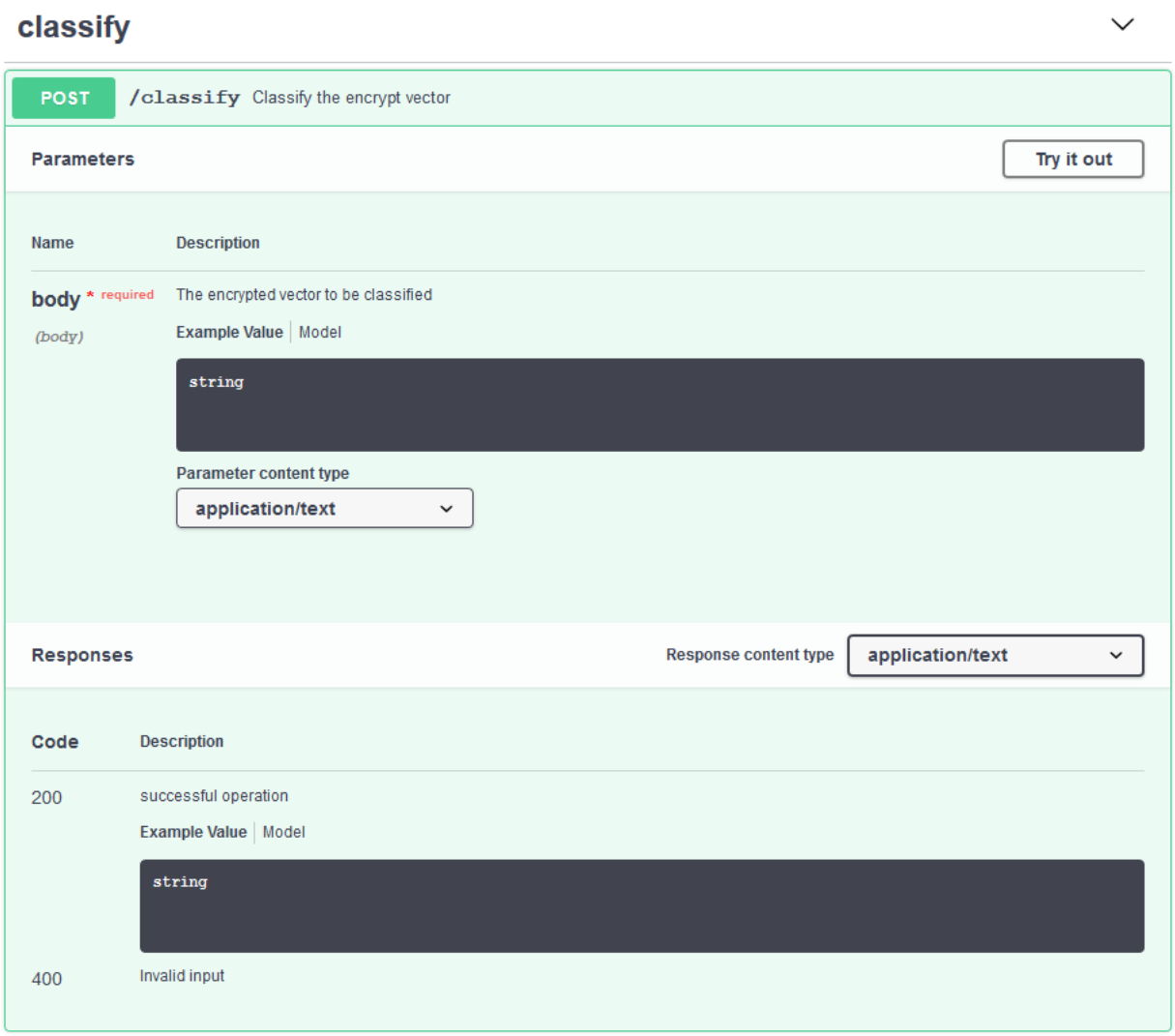
## GET /generateKeys



## POST /init


Init
POST
/init
Initializes the system
Initialize the neural network, FHE and mpc.
Parameters
Try it out
Name
Description
body * required
(body)
the architecture of the neural network, weights and public key
Example Value
Model
{
"NNArchetictureе": {},
"weights": {},
"Pk": {}
}
Parameter content type
application/json
Responses
Response content type
application/json
Code
Description
200
successful operation
400
Invalid input

**POST /classify**




The classify API call runs interactive protocol between the client and the served components underneath. This protocol computes the layers of the NN iteratively and generically. Namely, it computes the layers of the NN one by one (depending on the layer type) on the server side and it computes the activation functions using 2PC.

## 4.2 Collaborative Training of Neural Network

Collaborative training of NN allows multiple participants to perform a ML training collaboratively, while preserving the privacy of the training data.

### 4.2.1 Main components and their relationships

Privacy-preserving collaborative training will consist of the following components:

1. The client side (client agent) – it is responsible for performing the following functionalities:
    a. NN training
    b. Adding Differential Privacy (DP) noise
    c. Performing Anomaly detection
    d. Uploading noised gradients and downloading the model parameters to/from centralized server.

Figure 28 presents the client agent components that will run on the client side (trusted environment).

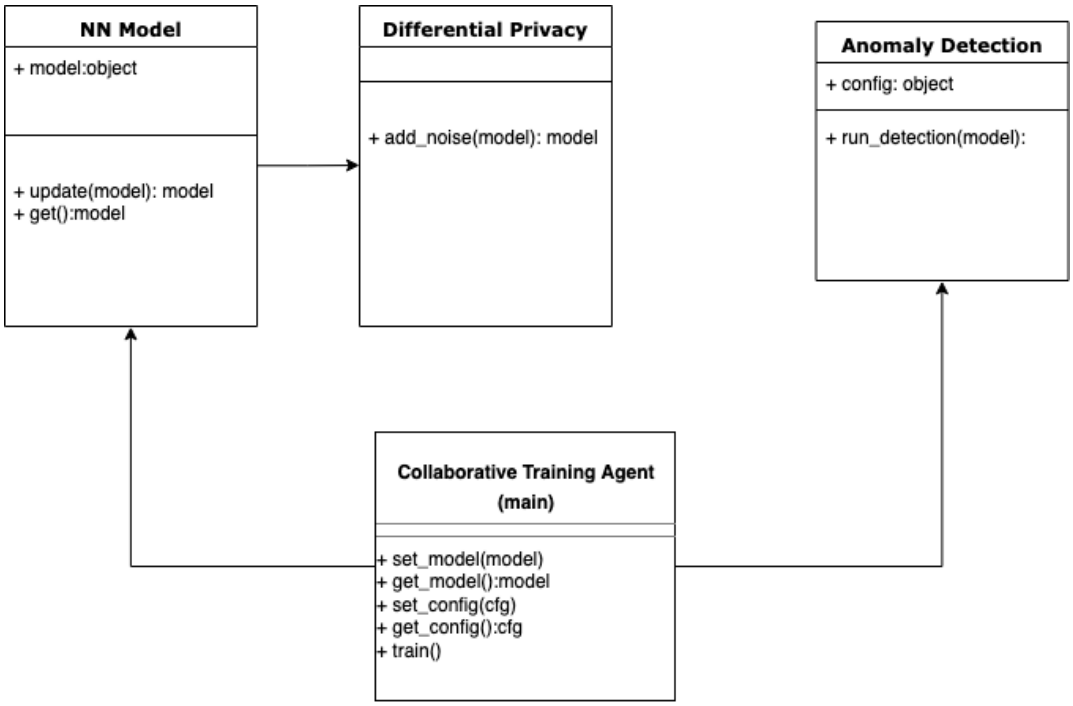



*Figure 28: The client agent components*

2. The server side (centralized server) – it will provide the following functionality:
    a. Allow participants to define and download the initial model
    b. Aggregate the gradients from the collaborative training participants.
    c. Allow participants to download the model parameters during the training phase.

d. Perform anomaly detection to identify adversary participants.

Figure 29 presents the server-side components that will run on PAPAYA K8s cluster. The service instance will be allocated per training task.

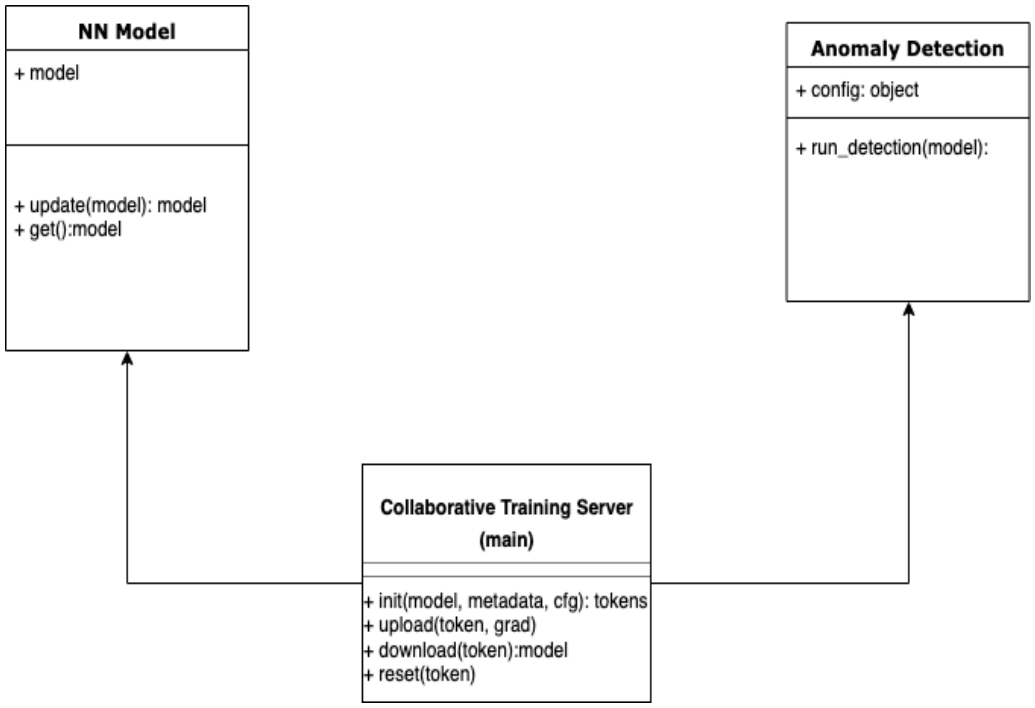



*Figure 29: The server components*

3. Object Storage:
    a. The server will store the model on the object storage. Storing model will allow users to download a trained model and to minimize damage and recovery in case of crushes during the training phase.

### 4.2.2 Behavioral analysis:

The training phase begins when all participants have completed the configuration phase, as defined in configuration section below. Each client will provide a path to the local dataset to the agent. The agent will perform a NN training localy and upload the noised gradients to the centralized server (located on the PAPAYA cluster). The server will agregate the gradiennts to the centralized model and will allow agents to download the updated model. Client agents will download the model and overwrite the local one. The agent will proceed the training phase on the updated model and so on. In addition, the agent and the server will apply anomaly detection on

the model and the model's updates. Each client will define the steps that the agent will perform in case an anomaly is detected. Figure 30 depicts the behavioral analysis and the configuration stage as described in the following section.

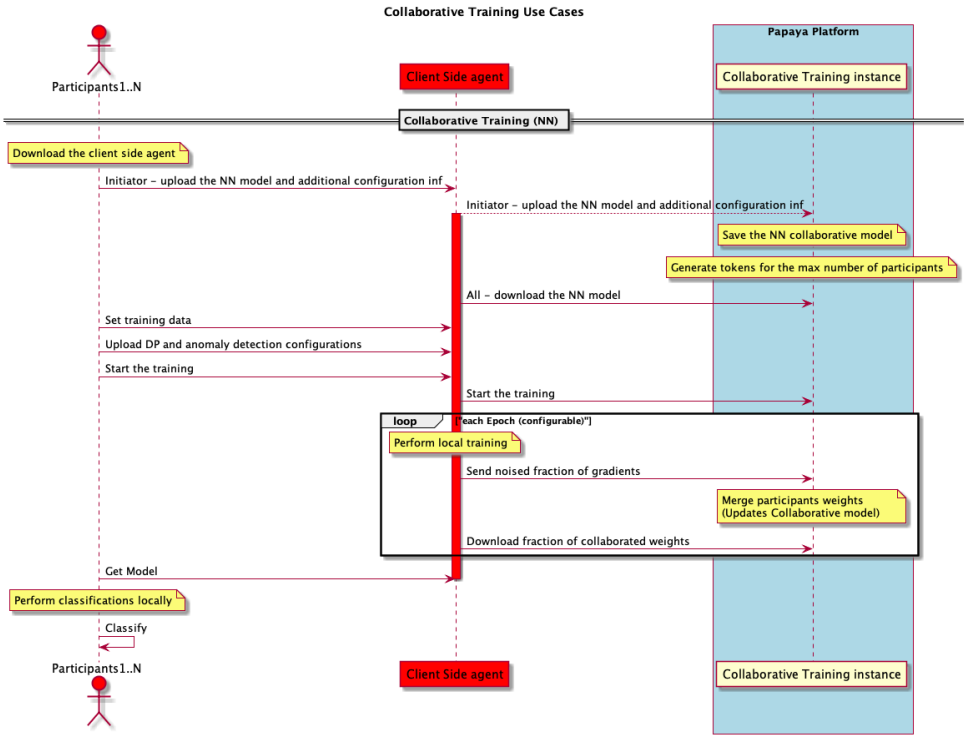



*Figure 30: Collaborative training - sequence diagram*

The client agent will communicate with the server via REST-API calls. The client app will communicate with the client agent via REST-API calls.

### 4.2.3 Deployment and configuration

**Deployment:**
The server and the client-side agent will be provided as container images and uploaded to the PAPAYA's CR. The deployment of the service on the platform is described in (Section 8.2). The application administrator will deploy the client-side agent container following the client app requirements.

The initiation step will be performed only by one of the participants. For the clarity of this explanation we will refer to this agent as the "**initiator**". After the deployment and before the service initiation is performed, the server doesn't contain neither NN model architecture and nor model parameters.

**Configuration:**
The client app administrator will provide to the agent an NN model (based on TensorFlow[7] as backend).

The initiator will upload the NN model to the server in hdf5 format (TensorFlow compiled model). In addition to model architecture, the initiator will provide the minimal and maximal number of participants. The server will save the uploaded model and generate tokens equal to the maximal number of participants. Only with this token the participant will be able to join and perform training. By using the tokens, we will be able to block unauthorized users from model downloading and training participation. The initiator will distribute the tokens between the participants in the offline process.

Each participant will be able to download the model and join the training. Only after the minimal number of the participants joined (registered), the training will be able to start (till then the server will respond to upload gradients/ download model parameters with an appropriate error).

Participant will not be able to join the training when the number of registered participants is equal to the maximal number.

The initiator will define the number of epochs performed by the model and whether the server will perform anomaly detection or not.

Only the initiator will be able to terminate and reset the training process. This feature could be useful for subsequent execution of the collaborative training with different parameters. In case the server detects any kind of anomaly, it will notify the participants on the detection via the response object of the download model or upload gradients API calls. The agent will perform anomaly detection on the client side. Client app administrator will be able to define the action (e.g. terminate the training or ignore), which will be performed if an anomaly (either on the client side or on the server side) is detected.

### 4.2.4 Implementation constraints

The service (both the client and the server side) will be implemented as REST-full web service using python[8] v3.* and Flask framework. Both client and server will document the APIs via Swagger[9].

The server will store the model by using the COS (cloud object storage).

[7] https://www.tensorflow.org/
[8] https://www.python.org/
[9] https://swagger.io/

As mentioned above, the server side will allocate legal tokens per training task and will expect to receive this token on every request. When the training is finished the model will be stored in the COS and will be available for download as long as the service is available.

The service will receive the COS credentials using Kubernetes secret mechanism.

**Constraints:**

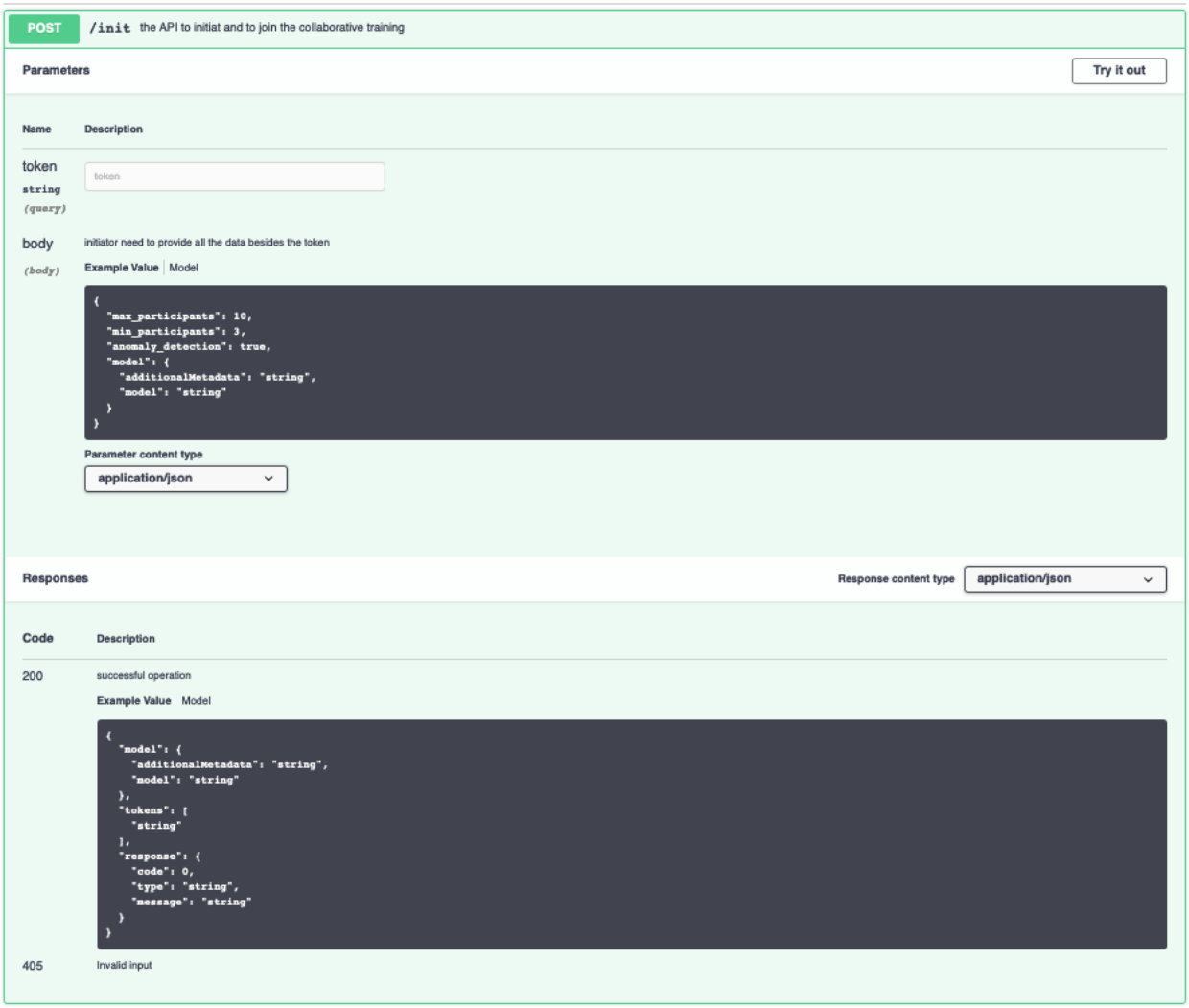
1. Service instance per NN model. Each service instance will support a single topology only.
2. We support only hdf5 file format (TensorFlow compiled model) to represent NN models.

The distribution of token to participants will be performed offline by the service initiator. (out of the PAPAYs scope)

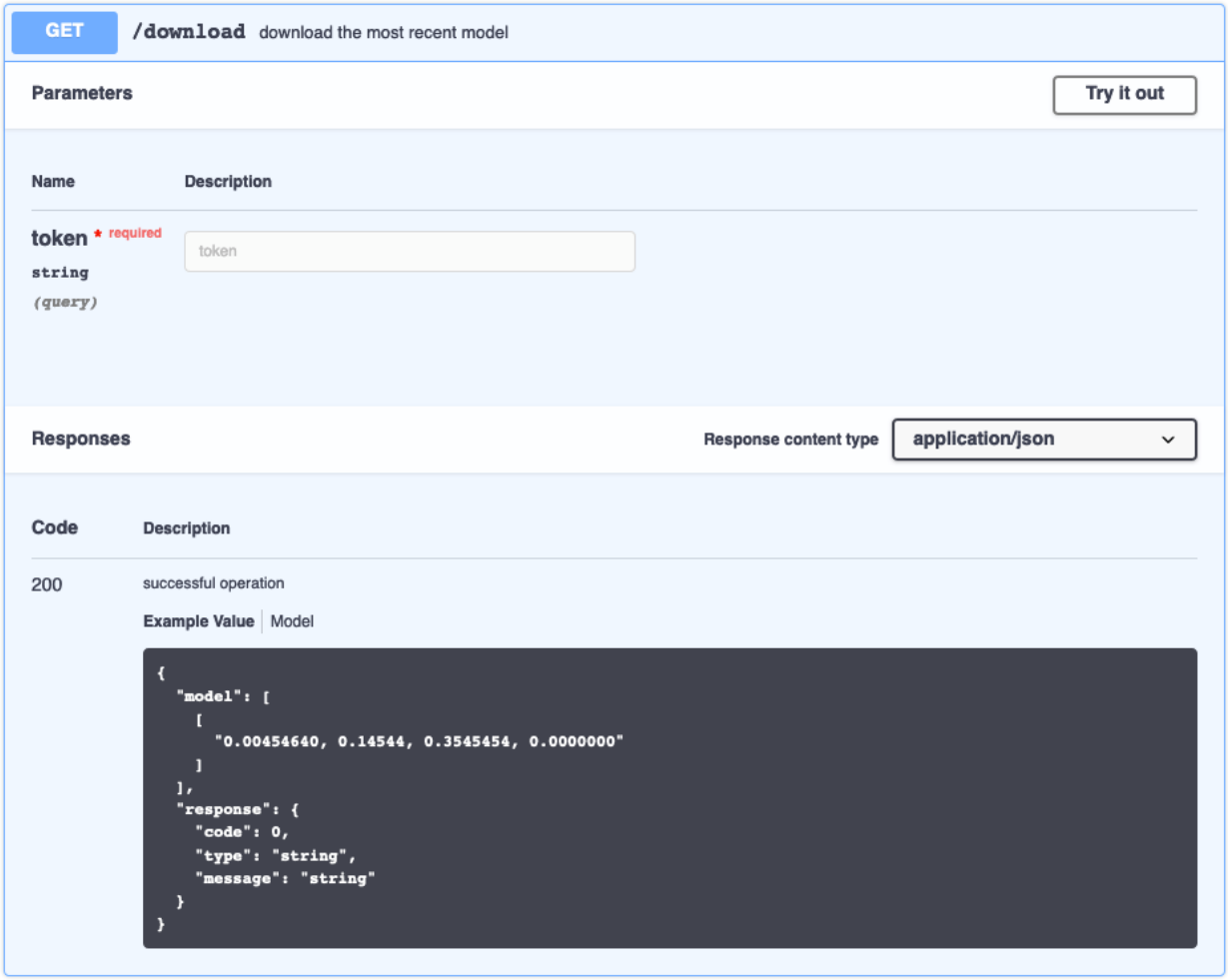
### 4.2.5 APIs

The initial version of the service APIs (documented with swagger) provided bellow.

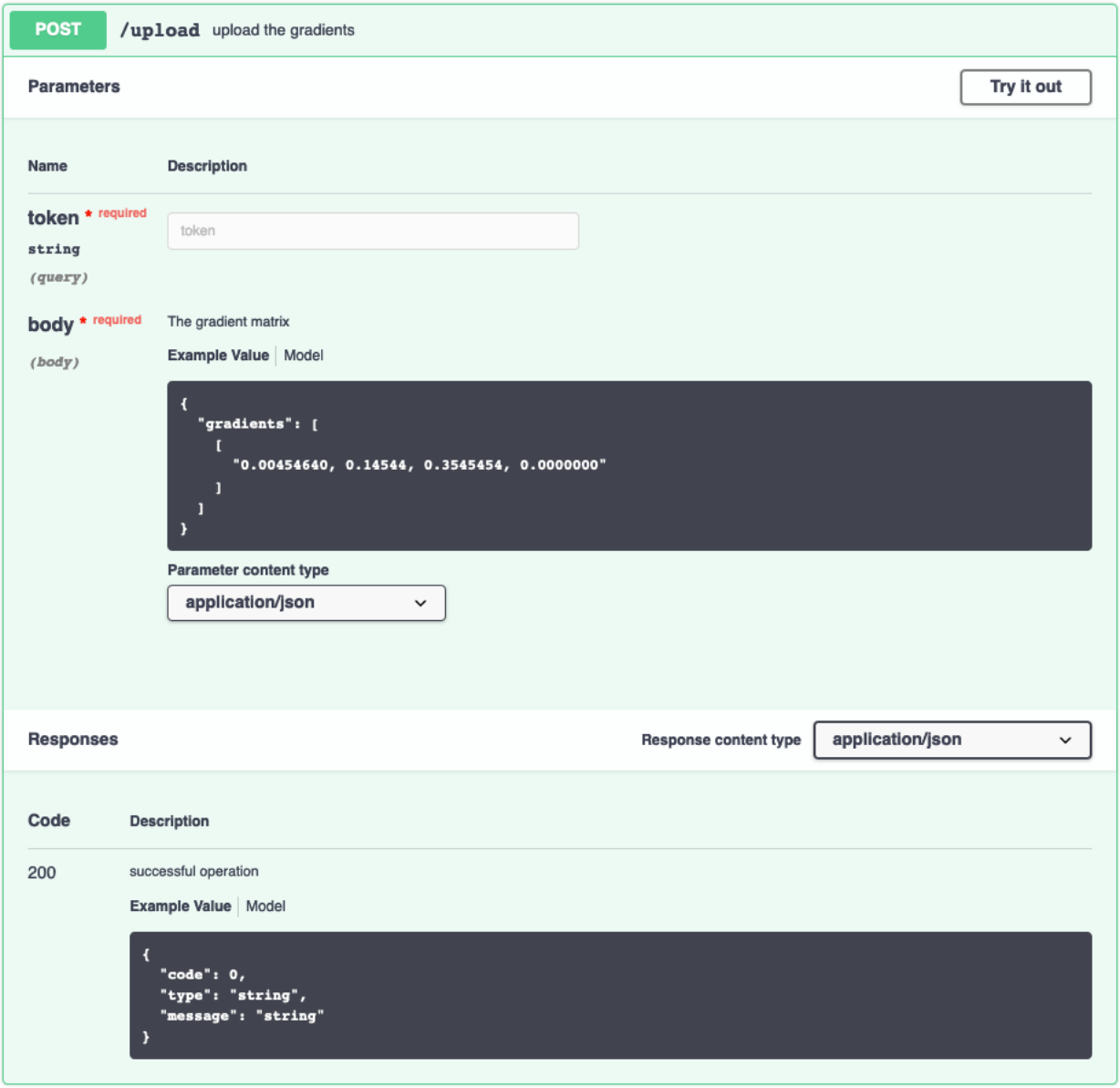
**POST /init**



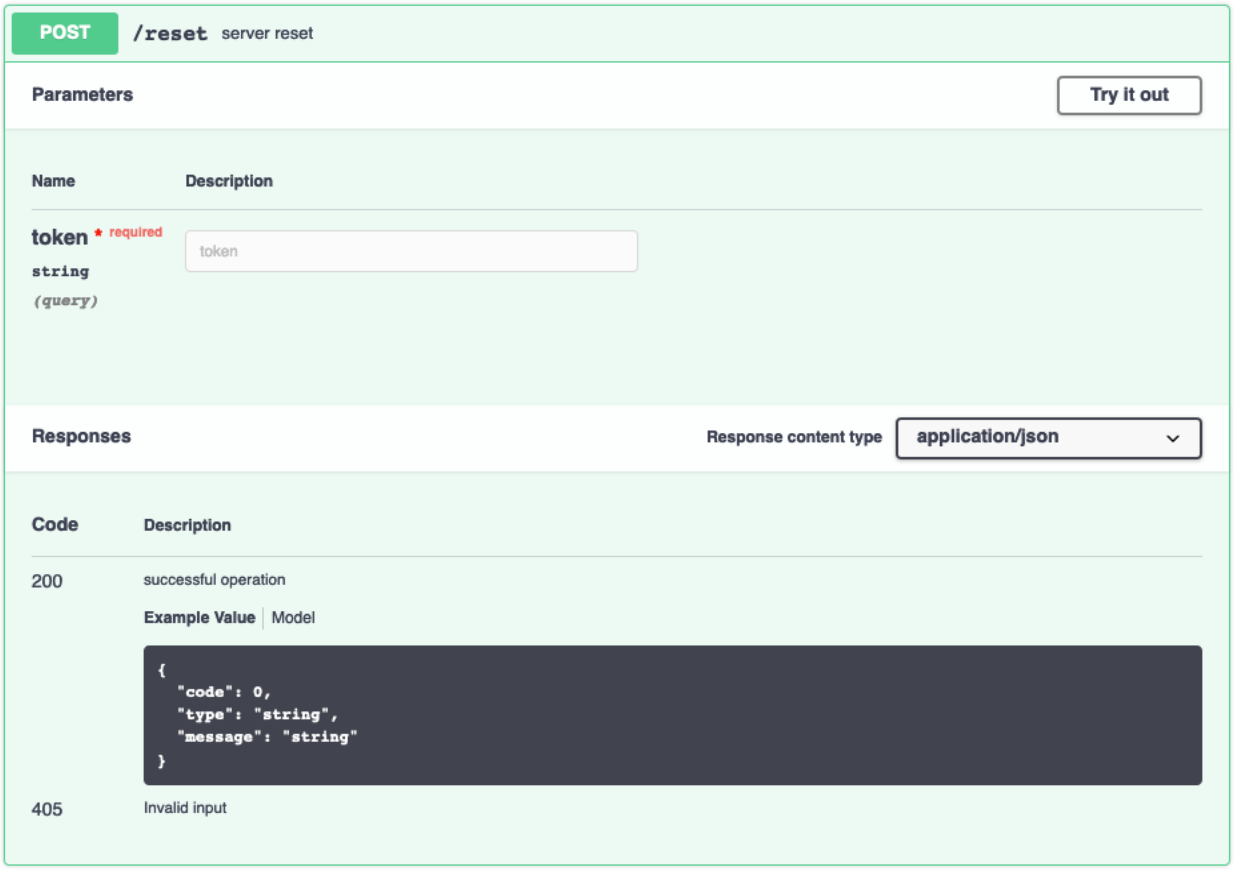
## GET /download

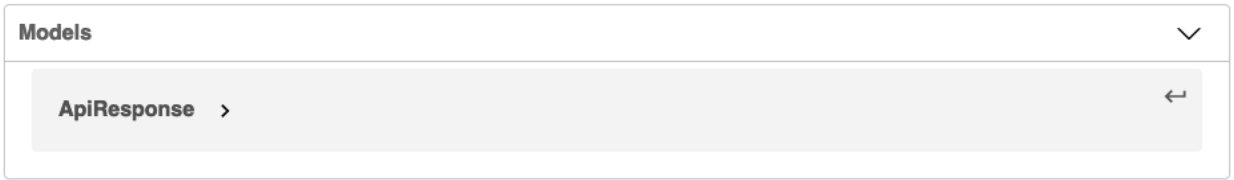


## POST /upload


POST
/upload upload the gradients
Parameters
Try it out
Name
Description
token * required
string
(query)
token
body * required
(body)
The gradient matrix
Example Value | Model
{
"gradients": [
[
"0.00454640, 0.14544, 0.3545454, 0.0000000"
]
]
}
Parameter content type
application/json
Responses
Response content type
application/json
Code
Description
200
successful operation
Example Value | Model
{
"code": 0,
"type": "string",
"message": "string"
}

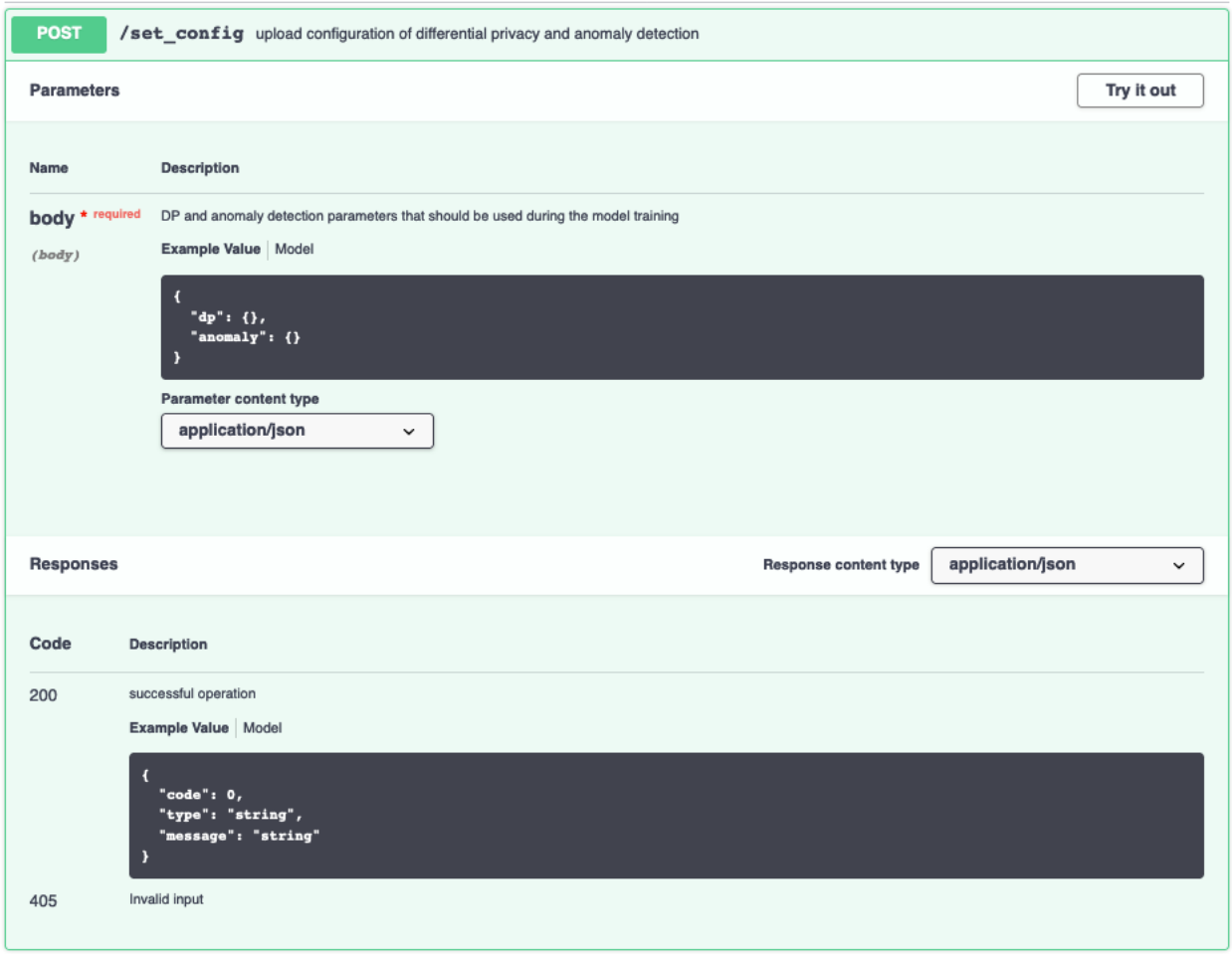
## POST /reset



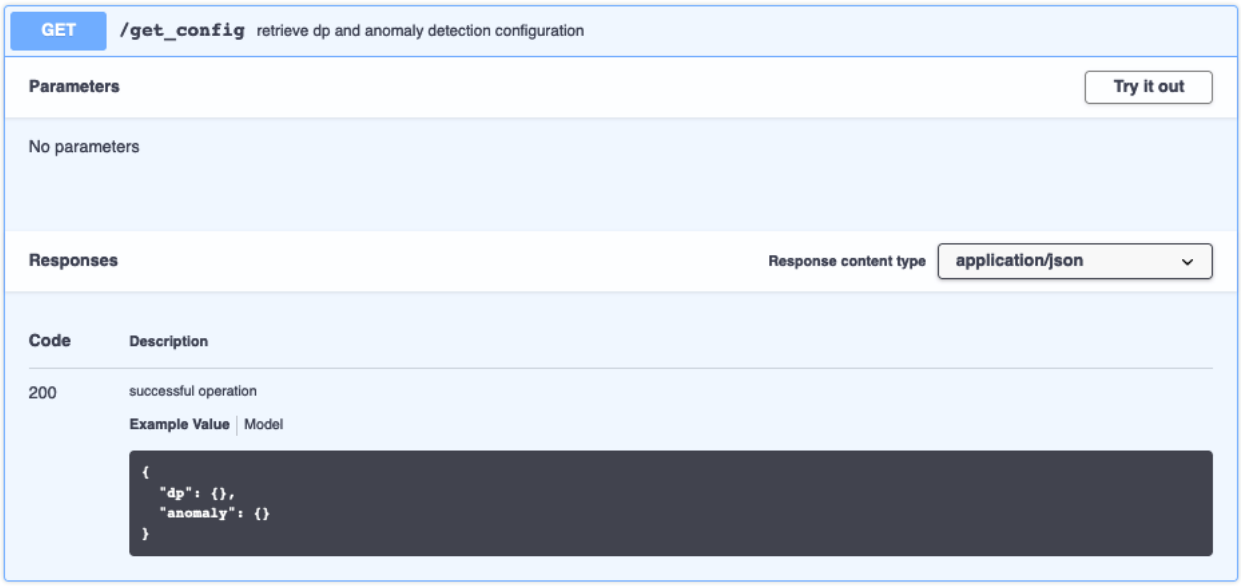




The initial version of the client agent APIs, documented with swagger, provided next.

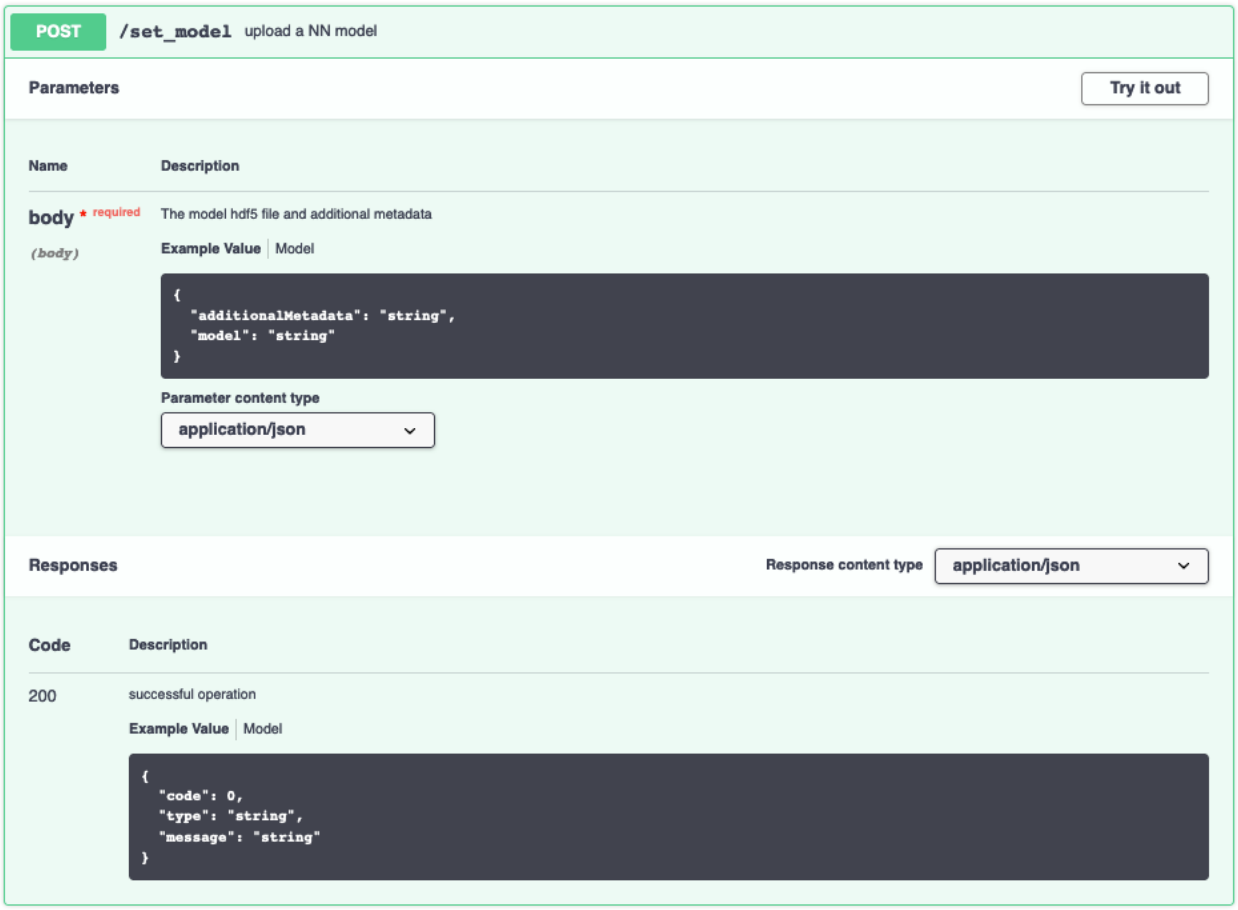
## POST /set_config



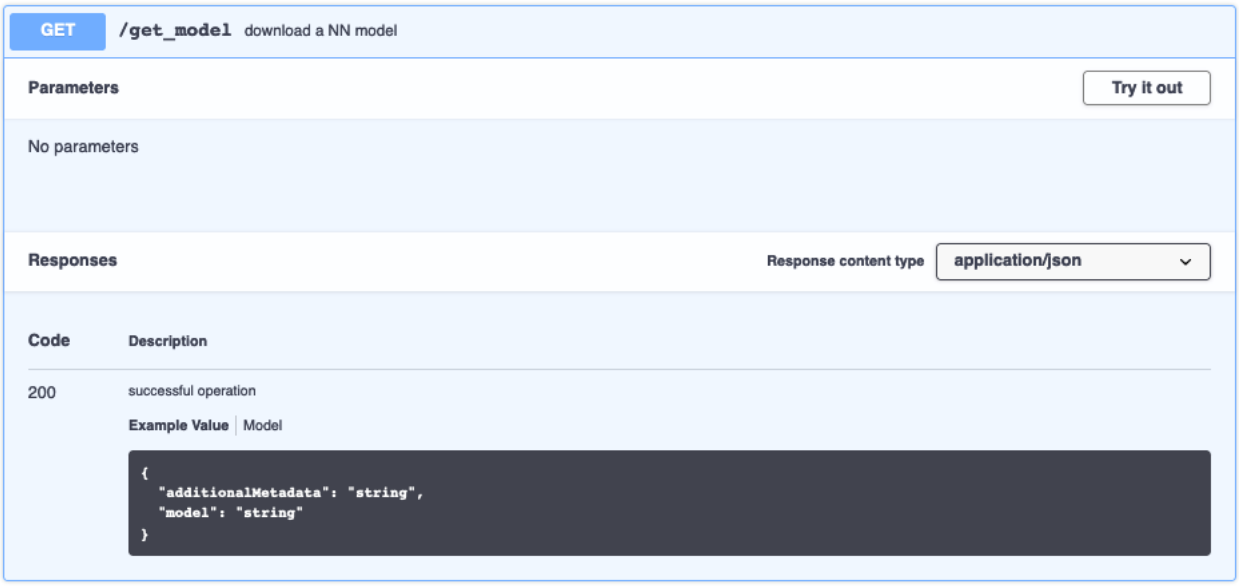
## GET /get_config



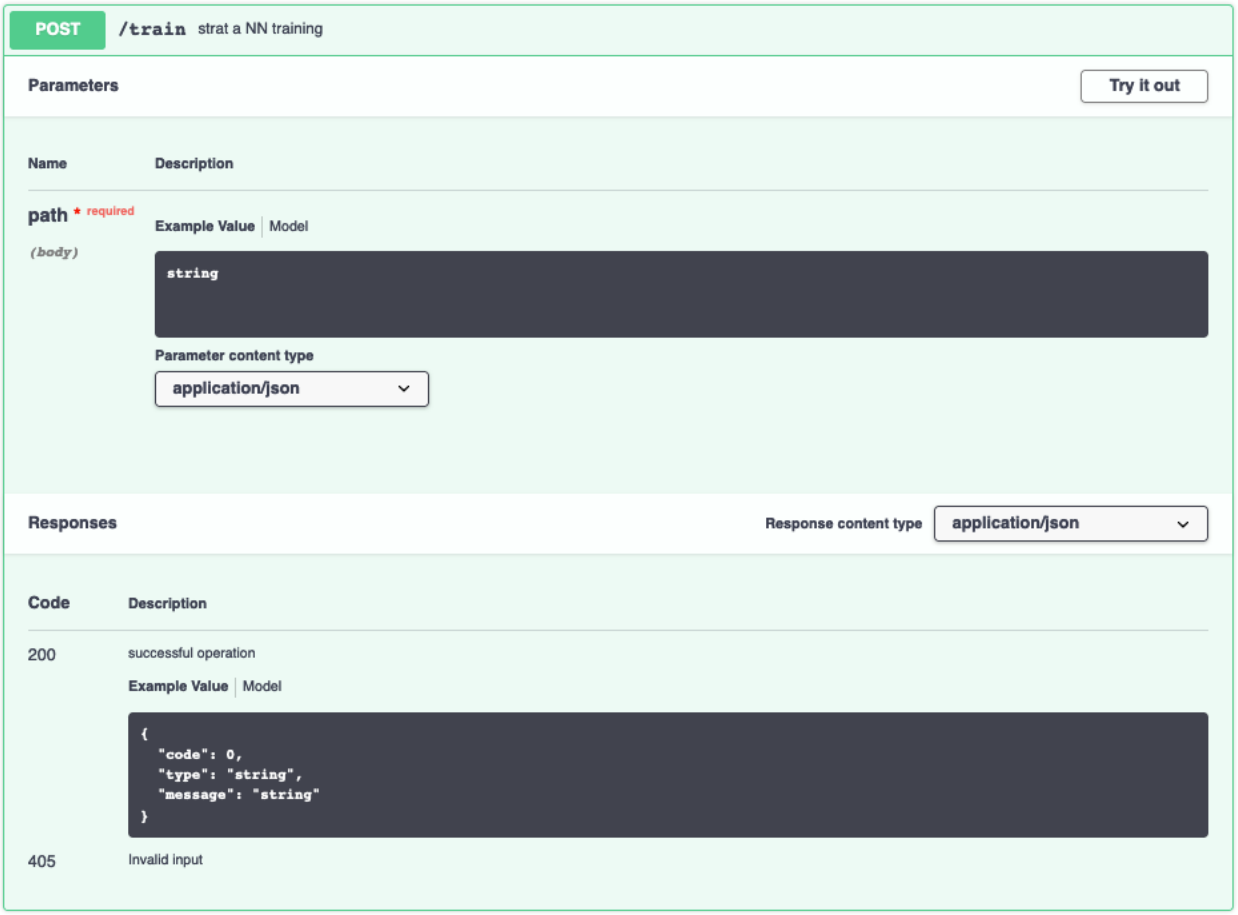
## POST /set_model



## GET /get_model



**POST /train**




## 4.3 Clustering

### 4.3.1 Privacy-preserving clustering based on PHE

In D3.1 [2], the preliminary design of a privacy-preserving (PP) trajectory clustering solution based on partially homomorphic encryption, in particular the Paillier encryption scheme, was introduced. In the proposed approach, there were several transformations of the operations in the original trajectory clustering algorithm [20] was replaced in order to support homomorphic encryption (HE) and optimize the computational cost. Such transformations consist of replacing the original distance metric with the Euclidean distance and employing an additional server to help support necessary computations with the Paillier encryption scheme. Nevertheless, the experimental results show that the proposed protocol remains costly in terms of the total execution time. Therefore, we are planning to develop a 2PC-based solution in order to achieve better performance in trajectory clustering. The complete design and the functional specification of the

PP trajectory clustering solution will be presented in deliverables D3.3 [7] and D4.2 [21], respectively.

#### *4.3.1.1 Main components and their relationships*

This solution has two main components:

1. Server-side component
2. Client-side component

**Server-side component:**

The planned solution is expected to be compatible with 2PC. Thus, the server-side component will contain Boolean and Arithmetic circuits to perform the computations for the clustering algorithm together with the client.

**Client-side component**

The client-side component consists of two modules:

1. Partitioning phase module: the client converts trajectory information to line segments and then encrypts/protects those line segments for their use in the clustering phase.
2. Clustering phase module: similar to the server-side component, the client also executes the computations with the server for the clustering phase and this component will also contain Boolean and arithmetic circuits.

#### *4.3.1.2 Behavioral Analysis*

The potential solution will operate as shown in Figure 31. The client processes trajectory data to obtain line segments. Later, it encrypts and sends those line segments to the server. Then, the client and the server will perform the actual clustering algorithm using 2PC. In addition to the 2PC-based solution, we also plan to work on the Paillier-based version to improve the performance of the protocol in terms of the total execution time.

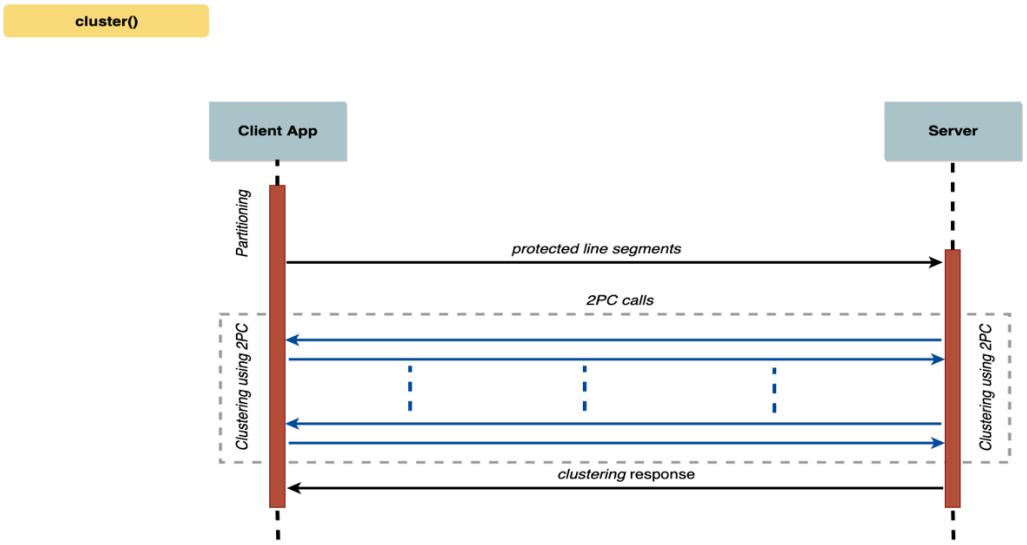



*Figure 31: Privacy-preserving Trajectory Clustering*

#### *4.3.1.3 Deployment and configuration*

Will be presented in D4.2 [21].

#### *4.3.1.4 Implementation constraints*

Will be presented in D4.2 [21].

#### *4.3.1.5 APIs*

Will be presented in D4.2 [21].

## 4.4 Basic Statistics

### 4.4.1 Privacy-preserving statistics based on Functional Encryption

In this section we present the architecture of the solution for the privacy-preserving computation of statistics using functional encryption.

#### *4.4.1.1 Main components and their relationships*

The solution consists of three main components:

1. Server-side component
2. Client-side component
3. Requestor-side component

**Server-side component:**

It consists of the following two modules (Figure 32):

- **FE module**. It provides the *evaluation* functionality that allows to perform operations (mainly addition, multiplication, dot product).
- **Operation module**. It will help define the functionality $f$ that will be evaluated on encrypted data and outputs a description of $f$. For instance, in the case of the dot product $\langle ., y \rangle$, with a vector $y$ owned by the server, the module will output a (possibly encoded) description of $y$.

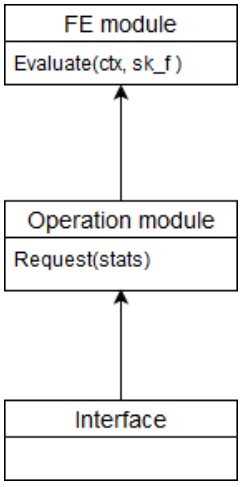



*Figure 32: Server-side component*

**Client-side component:**

It consists of the following two modules (Figure 33):

1. **FE module**. It provides all the client-side functionalities, namely *KeyGen*, which generates the secret key for the client, *encrypt* which takes as input the plaintext data and *DKeyGen* which outputs the evaluation key for the server-side component.
2. **Ext_libs**. It provides additional methods that may be necessary for the client-side (such as a secret sharing library).

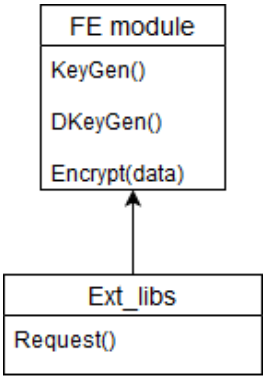



*Figure 33: Client-side component*

**Requestor-side component:**

It consists of the following two modules (Figure 34):

1. **Agent**. It has a *request* method that allows the Requestor to request analytics from the service.
2. **Crypto module**. This module may be solicited to provide the Requestor with keying material that will be used to decrypt the final results.

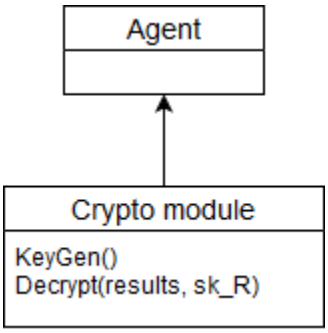



*Figure 34: Requestor-side component*

**Detailed description of the modules:**

**FE module**. The FE module implements a functional encryption scheme (see [2]) specifically for inner product in the decentralized and multi-client scenario. The library may provide the following algorithms: KeyGen (to generate the client's key pairs), Encrypt (to encrypt a plaintext), DKeyGen (to generate the functional decryption key) and Evaluate (to evaluate the requested function on the ciphertexts).

**Operation module**. This module tailors the function to be evaluated. Since the current existing functions that are implemented in practical FE schemes are inner products and quadratic polynomials, this module will, based on the request, output either a description of the vector $y$ used for the inner product or a description of the matrix representing the quadratic polynomial.

**Ext_libs**. In the current version of the privacy-preserving statistics with FE, we may need this additional library to invoke a (threshold) secret sharing algorithm. This aspect will be detailed in subsequent deliverables (in D3.3 [7] and D4.2 [21]).

#### *4.4.1.2 Behavioral analysis*

The service operates in two main phases:

1. System initialization (including the analytics initialization and the key generation)
2. Statistics (including the computation on encrypted data and the decryption of the results)

**System initialization (*Figure 35*)**

In this phase, the Requestor generates its key material by invoking the Crypto module (steps 1 and 2). Afterwards, the Requestor invokes the Agent module to issue an analytics request which includes the description of the requested statistics operation and the description of the data to be analyzed (Step 3).

The server receives the request and invokes the Operation module to generate a description of the function $f$ based on the requested statistics operation (steps 4 and 5). The description of $f$ and the request for data is forwarded to the client (Step 6).

The client calls the FE module to generate its own key material (steps 7 and 8). It also generates the functional key $sk_f$ that will be used by the server to perform the requested operation (steps 9 and 10). Depending on the implemented FE scheme, each client issues a partial functional key; in this case, $sk_f$ is the combination of all partial functional keys. If needed, the client also calls the Ext_libs module to generate auxiliary information (steps 11 and 12). The functional key $sk_f$ is transmitted to the server (Step 13).

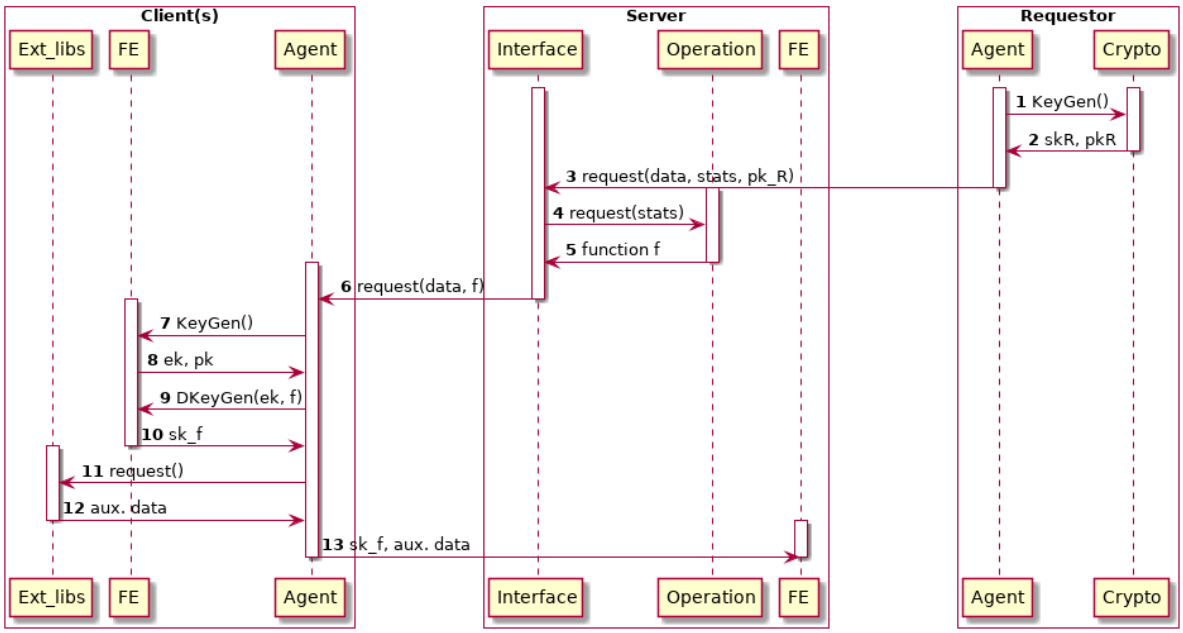



*Figure 35: Initialization (stats)*

**Statistics phase (*Figure 36*)**

The client encrypts the requested data using the key material produced in the initialization phase by invoking the FE module (steps 1 and 2). If necessary, the Ext libs module is called to produce auxiliary information (steps 3 and 4).

The ciphertexts are sent to the server (Step 5), which invokes the *evaluate* method of the FE module to perform the statistics operation over encrypted data, using $sk_f$ (steps 6 and 7). The result is somehow blinded using Requestor's public key.

The encrypted result is sent to the Requestor still in the encrypted form (Step 8). The latter invokes the Crypto module to decrypt the result using its own private key (Step 9). Finally, the Requestor gets the result in plaintext format (Step 10).

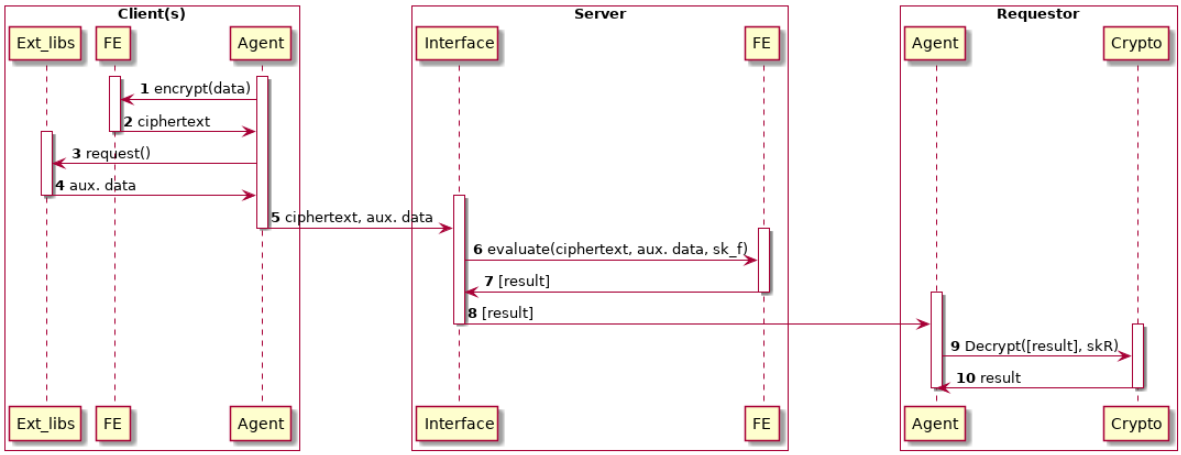



*Figure 36: Statistics phase*

### *4.4.1.3 Deployment and configuration*

The server and requestor components will be deployed as Docker containers. As the client component will run on users' mobile devices, they will be deployed through a mobile phone application (most probably running Android).

Regarding the server configuration, it should accept both the clients' and the requestor's IP addresses and port numbers. It should also receive from the requestor the total number of clients taking part in the study and the minimum number of clients (threshold value) below which nothing can be learned by the requestor. Note that the requestor does not need to interact with the clients.

On the other side, the clients should have the server's IP address and port number. Besides, the clients must hold a (possibly random) public identifier (or index) that may be used to link keys to their holders. Finally, the clients must agree on a common public key (in addition to their respective key pairs). This common public key is used to generate the key shares, that the server needs to perform the requested evaluation.

### *4.4.1.4 Implementation constraints*

As mentioned before, the requestor and the clients do not interact with each other. More importantly, clients do no interact with each other, while they are required to jointly agree on a common public key. This agreement step may be performed prior to the initialization phase.

During the collection of encrypted data provided by the clients, the server should wait until it receives more than the threshold value. Note that the server maintains a list of indices of those clients that participate in the analytics and deletes this list right after the completion of the computation.

Considering the current state of the art, only two families of functions are implemented in FE schemes, namely the dot product [22, 23, 24, 25] and quadratic functions [26, 27], hence limiting the number of available implementations.

Finally, the computations at the client side should be lightweight, since all operations described above are performed in their mobile phones.

#### *4.4.1.5 APIs*

The server handles the privacy-preserving computation requests issued by the requestor. The server finally returns the (encrypted) computation results to the requestor.

It is assumed that an active connection exists between one client and the server and that the clients somehow listen to data requests from the server. When the clients receive a request from the server, specifying which (encrypted) data attributes to collect for the requested computation, they generate the (partial) functional key (as shown in *Figure 35*) as well as the key material used to encrypt their data (*Figure 36*).

## Requestor API

| | | |
|---|---|---|
| POST | /Request/dotproduct | Requests for computation of a dot product |
| GET | /Request/dotproduct | Returns the encrypted result of a dot product |
| POST | /Request/quadratic | Requests for computation of a quadratic function |
| GET | /Request/quadratic | Returns the encrypted result of a quadratic function |

## Server API

| | | |
|---|---|---|
| POST | /FE/funcKey | Forwards functional key to server |

## Client API

| | | |
|---|---|---|
| POST | /FE/encData | Sends encrypted data with FE scheme |

### 4.4.2 Privacy-preserving Counting using Bloom Filters

In this section, we present our solution to privacy-preserving counting using Bloom Filters.

#### *4.4.2.1 Main components and their relationships*

The solution consists of three main components:

1. Server-side component.
2. Requestor-side component.
3. Client-side component.

**Server-side component:**

It consists of the following modules (Figure 37):

- **Storing module**. Such a module is used by the Server to store the request received from the Requestor. In particular, this module is used to store the received cryptographic keys for future use.
- **Operation module**. Depending on the analysis stored by the Storing module and taking as input one or several encrypted Bloom Filters, this Operation module performs the analytics (set intersection, set union and counting). The result is finally sent, in the encrypted form, to the Requestor.

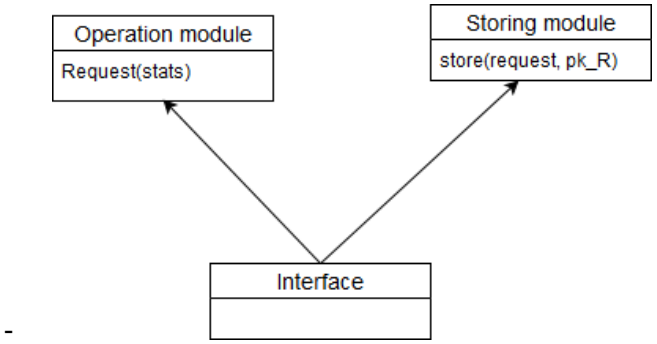



*Figure 37: Server-side component*

**Requestor-side component:**

It consists of the following modules (Figure 38):

1. **Agent module**. It contains a *Request* method that allows the Client to request analytics from the service.
2. **Crypto module**. Such module is composed of two methods. The first one is used to generate the cryptographic keys used to eventually obtain the results of the statistics. The second one is the decryption method, used to decrypt the received result.

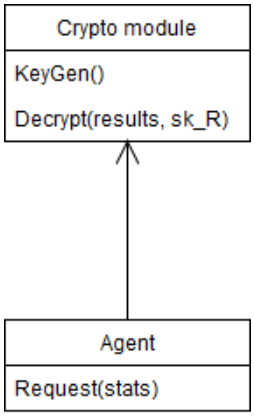



*Figure 38: Requestor-side component*

**Client-side component:**

It consists of the following modules (Figure 39):

1. **Agent module**. It interfaces with the Server, collects the probe data from antennas and creates the plain Bloom filters. It also invokes the Crypto module to encrypt the generated filters.
2. **Crypto module**. Such a module encrypts the input plain Bloom Filters, used after their reception. This module makes use of the cryptographic keys that have been received previously by the server.

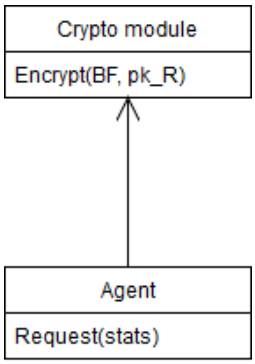



*Figure 39: Client-side component*

### *4.4.2.2 Behavioral analysis*

The service operates in three main phases (see Figure 40):

1. System initialization, including the analytics initialization and the key generation;
2. Encryption phase - used to encrypt the Bloom Filters;
3. Statistics phase, including the computation on encrypted data and the decryption of the results

**System initialization**

In the system initialization phase, the Requestor first invokes the key generation method of the Crypto module (Step 1) to generate the cryptographic keys (Step 2) that will be used to protect the probe data (stored in a Bloom filter). The Client then generates a description of the analytics it wants to obtain (quantity of probe data, locations, dates and analytics, among counting, set intersection and union). It finally generates analytics request which includes the description of the analytics and the cryptographic key used to encrypt the data (Step 3).

The Server receives the request and invokes the Storing module (Step 4) to store analytics and the cryptographic key. It then generates a request to an internal interface to obtain the Bloom Filters related to such analytics (Step 5).

**Encryption phase**

At different times, depending on the analytics, the Client (Orange Network) obtains from internal resources one or several Bloom Filters (steps 6 and 7). It requests the encryption module to encrypt the Bloom Filters (steps 8 and 9) and store the result internally. Depending on the request, the Client may have to wait for the reception of several Bloom Filters (at different times) before processing the next phase.

**Statistics phase**

When all the data have been obtained, and according to the initial request by the Client (Requestor), the Server makes use of the Operation module (steps 11 and 12) to perform the analytics (counting the number of people in one Bloom Filter, or in the intersection/union of several Bloom Filters). The result, still in the encrypted form, is sent to the Client (Step 13).

The Client executes the Decryption method of its Crypto module (Step 14) to obtain the result in plain (Step 15).

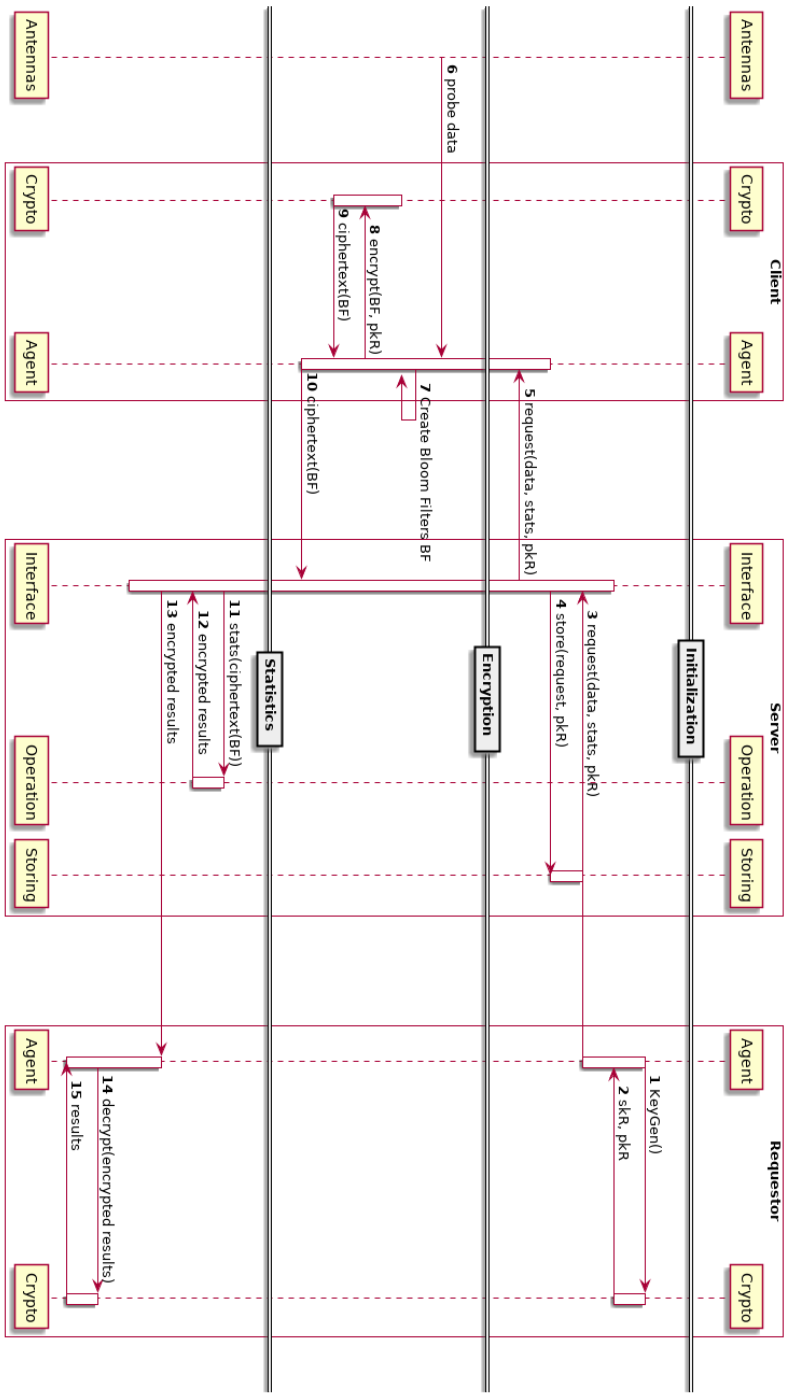



*Figure 40: Privacy-preserving statistics with Bloom Filters*

#### *4.4.2.3 Deployment and configuration*

The above-described components will be deployed as Docker containers.

Configuration parameters of the server include: the IP addresses and port numbers of the two types of clients, and in turn the clients should have the server's IP address and port number. Requestor and client do not interact with each other.

#### *4.4.2.4 Implementation constraints*

During the collection of encrypted Bloom filters provided by the client, the server may wait for the reception of several Bloom Filters (at different times) before applying the statistics operation.

The input Bloom filters must have all the same size so that operations such as intersection could be defined.

#### *4.4.2.5 APIs*

The server handles the privacy-preserving computation requests issued by the requestor. It also interacts with the client, requesting to provide with encrypted Bloom filters as inputs to the requested computation. The server finally returns the encrypted results to the requestor.

It is assumed that an active connection exists between the client and the server and that the client listens to data requests from the server. When the client receives a request from the server, it invokes the following POST method.

## Requestor API

| | | |
|---|---|---|
| POST | `/Request/request` | Requests for computation of a statistic operation |
| GET | `/Request/encResult` | Returns the encrypted result of the requested computation |

## Server API

| | | |
|---|---|---|
| POST | `/Request/publicKey` | Sends the public key |

## Client API

| | | |
|---|---|---|
| POST | `/data/encData` | Sends encrypted data to server |

# 5 Platform Security and Transparency

## 5.1 IAM

### 5.1.1 Main components and their relationships

The Identity Access Manager will provide the Authentication and Authorization services to the different components that integrate the PAPAYA framework with especial attention to the PAPAYA platform. In order to do so and taking into consideration that the research and development of these types of services is not a main part of the scope of the PAPAYA project, the Consortium has chosen to use a solution based on an already available and wide used open source solution. Starting from it, the Consortium plans to adapt and adjust the solutions chosen to suit into the PAPAYA developments. After researching in the state-of-the-art available solutions within the market the Consortium has selected Keycloak[10] as the underlying framework for providing Authentication and Authorization services.

In order to have a minimum impact from the point of view of time and resources needed for the integration of this type of services within the PAPAYA platform, the consortium has chosen to apply the classical bastion-hosts approach, where a gateway/proxy filter all the incoming request to a private network to assure that they are authenticated and authorized and if so redirect the corresponding request to the appropriate component. Figure 41 shows in detail the design of the selected approach, defining the following components: IAM server, User Data Base (DDBB) and the Security Proxy and their iteration with the rest the components that comprise the PAPAYA platform and clients. Further information describing these components can be seen in 5.1.2. It is worth mentioning that among the family of different solutions that Keycloak facilitates, there is already a gateway/proxy software devoted for applying the bastion-host paradigm named Security Proxy[11].

Applying this solution to the development of the PAPAYA Platform will not only have a minimum impact on the rest of the developments but it also provides a solution that based on common and widely used components from the industry of the protocols such as *OpenID Connect*[12] and *OAuthz 2*[13]*.* This approach will also facilitate the future integration with legacy systems, improving then the market opportunities of the final results of the PAPAYA project.

[10] https://www.keycloak.org/
[11] https://github.com/YunSangJun/keycloak-proxy-demo
[12] https://openid.net/connect/
[13] https://oauth.net/2/

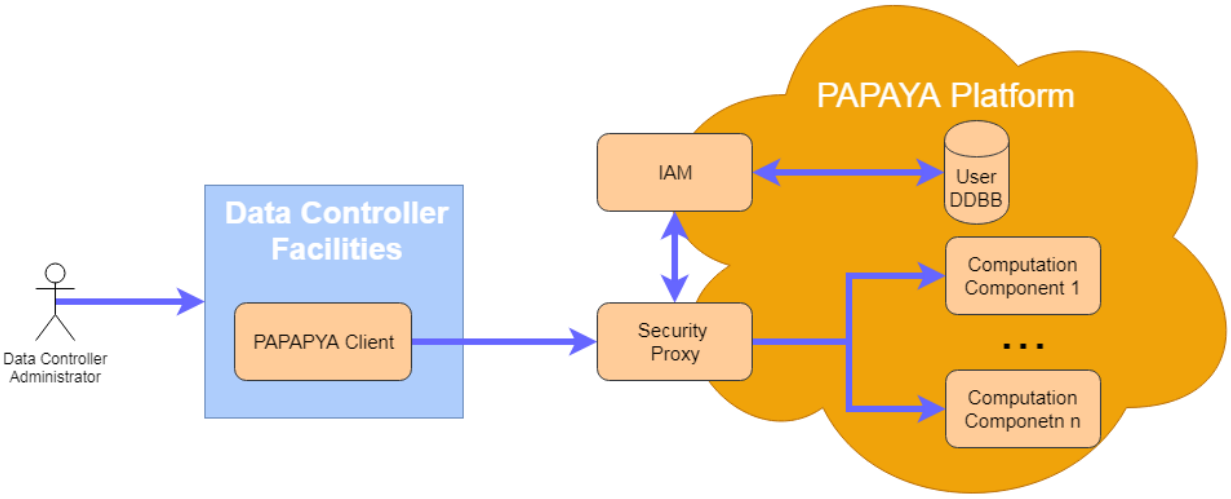



*Figure 41: IAM and Security Proxy in the PAPAYA Platform*

The diagram below (Figure 42) shows in detail the communication flow when requesting a service to the PAPAYA platform. As first step, the Data Controller Administrator, when setting up the PAPAYA Platform, will register the new PAPAYA client components as clients in the IAM, obtaining then the corresponding token associated to that client. Once the client has been registered within the IAM system, it can use the received bearer Token that can be utilized for several different requests to the Computation Components. In order to do so the IAM will provide an interface to allow the Data Controller Administrators to update/create new clients. It should be noted that only with including the bearer token on the corresponding header of the request by the PAPAYA client, the Security Proxy will carry out with all the Authentication and Authorization operations in a transparent way from the point of view of the developer. Moreover, in the case that the operation is granted by the IAM, the Security Proxy will redirect the request directly to the appropriate Computation Component.

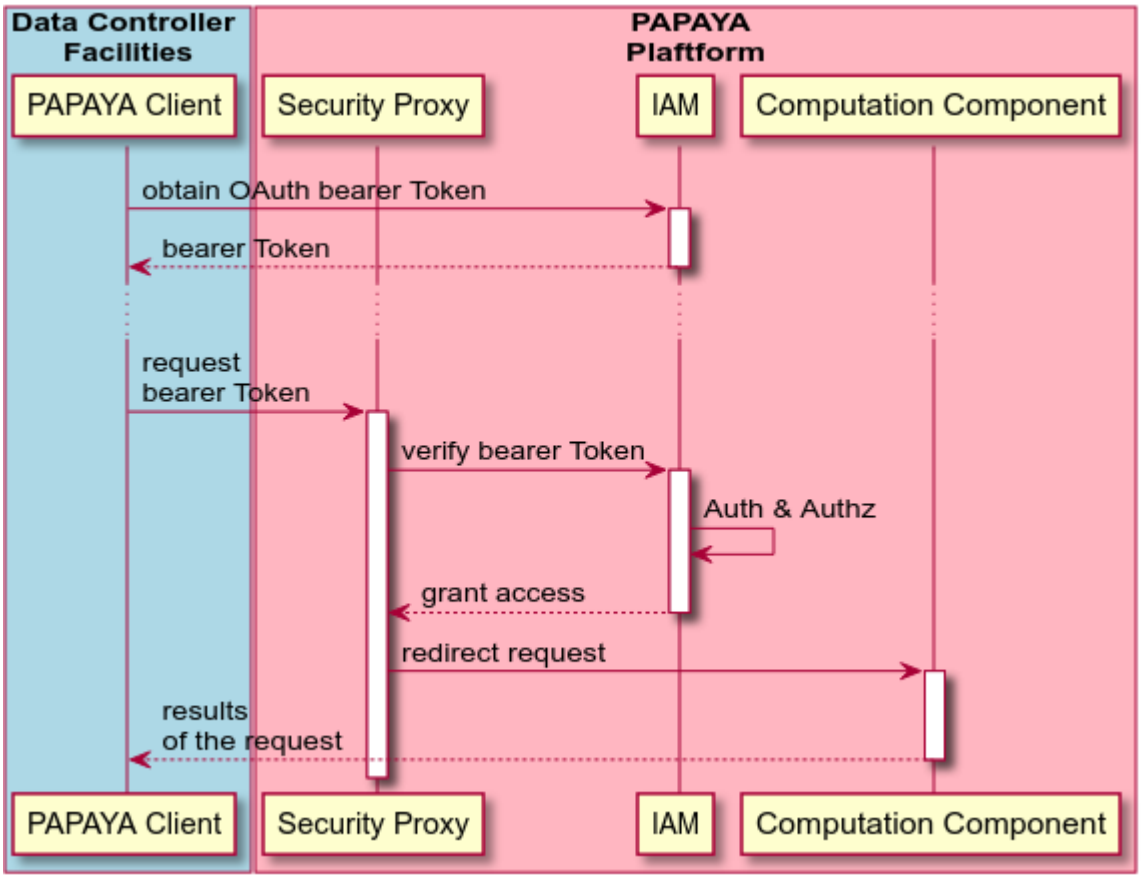



*Figure 42: Communication flow requesting a service to the PAPAYA platform*

### 5.1.2 Deployment and configuration

For the deployment and configuration, as mentioned before, it had been chosen a classical bastion-host approach. Therefore, it has a direct impact on the deployment of the whole PAPAYA Platform. In order to secure the system properly, all the access to the platform must past through the Security Proxy server and then be redirect to the different computation components that conform the PAPAYA Platform. Hence, it must be mentioned that the different computation components will be never accessed from the exterior directly.

In addition, as it shown in the Figure 41, the final deployment will need to include three different components in order the PAPAYA Platform can be provided with Authentication and Authorization services: IAM server, User Data Base (DDBB) and the Security Proxy itself. It is worth mentioning that these three different components will de integrated into the PAPAYA platform using Docker containers and defining the associated services with the help the Kubernetes functionality The **User Data Base** will be used to store the user and client's information In our particular case it is foreseen that will need to incorporate end users into system (just the administrators of the

system). However, the system will need to store the clients' information appropriately and this component is devoted for it. Regarding the connectivity of this component, the User Data Base component won't need to have an external access, as the IAM server will be only component that will access to it. The **IAM Server** will be deployed using an adapted version of the Keycloak server configured to suit the project purposes. This component will be used from the exterior for two different purposes: for the IAM administrator to update the PAPAYA clients and for the PAPAYA clients to obtain the corresponding bearer Token. Thereby this component will need to be accessible from the exterior hence it will be necessary to define the proper service on the final Kubernetes deployment configuration.

And last but not least the final deployment will include the **Security Proxy**. In order to configure the Security Proxy, we need to appropriately set up the different parameters of the Security Proxy server itself but also the type of redirections to the computation components. An example of the configuration file of the Security Proxy defining the access and redirections to the *computation component 1* is as follows:

```
replicaCount: 1
image:
  repository: papaya/security-proxy
  tag: 1.0.1.Final
  pullPolicy: IfNotPresent
...
configmap:
  targetUrl: http://computation-component-1-service
  realm: computation
  realmPublicKey: "MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCg...4qwIDAQAB"
  authServerUrl: http://iam.papaya.eu/authvb
  resource: computation-component-1
  secret: 2b2c17f0-245e-4978-a663-9a02a268a8f4
  pattern: /computation-component-1
  rolesAllowed: computator
```

This configuration file will be uploaded when a new service is defined in PAPAYA platform, in order the client will be able to access to the new mentioned service.

Obviously, this component will be required to be accessible from the exterior (on our particular case for the different PAPAYA clients configured). Therefore, as with the IAM Server commented before, it will be necessary to define properly the specific service on the final Kubernetes deployment configuration.

### 5.1.3 Implementation constraints

The solution selected is widely accepted and used across the web wide world. However as mentioned before, selecting this bastion-hosts paradigm implies some limitations in the final deployment. The main constrain associated with this approach is that all accesses (HTTP and/or TCP) from the exterior to the computing components within the PAPAYA Platform must pass through the Security Proxy. Therefore, on one hand the final deployment must assure that no external access can be performed to the internal components. On the other hand, the fact that all the accesses are done through the Security Proxy can produce a bottleneck effect when accessing to the PAPAYA Platform services. However, this effect can be easily corrected by incrementing the number of nodes devoted to execute the Security Proxy service.

### 5.1.4 APIs

As the solution selected is based on an already existing open source software, the definition of the API and the details of the different type of parameters involved can be obtained using the following link: https://www.keycloak.org/docs-api/6.0/rest-api/index.html

Moreover, for the different installation and configuration aspects, it can be consulted in: https://www.keycloak.org/docs/latest/server_installation/index.html

## 5.2 Auditing

To support auditing and towards being able to hold stakeholders accountable for their use of PAPAYA, data processing is logged both as part of the platform and locally at agents.

### 5.2.1 Platform auditing

Platform auditing should enable a platform admin to view all logs from all processing *on the platform* through a centralized log analysis system. Further, clients should be able to view all of their relevant logs from processing on the platform. Both of these views should be provided through the platform dashboard (see Section 3).

#### *5.2.1.1 Main components and their relationships*

There are two main components dedicated to auditing:

- the *Auditing Agent* (AA) component that is responsible for transporting logs from platform services, and
- the Auditing *Collector* (AC) component that collects all logs for centralized analysis.

There are many AAs on the platform—conceptually up to one per platform service instance—that transport logs from the service instance(s) to one or a few (for redundancy/scaling) ACs. The AC is queried by the platform dashboard for retrieving logs. Figure 43 shows the setting of the components with some additional details explained later.

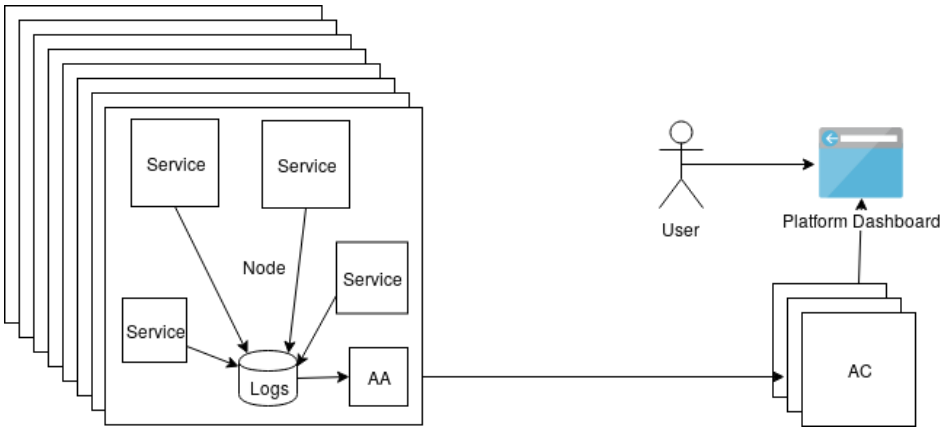



*Figure 43: Each Kubernetes node runs an Audit Agent (AA) that transports the logs generated by services on the node to one or more Auditing Collectors (AC). The Platform Dashboard reads logs from AC and makes them available to Platform Dashboard users.*

Concretely, the AA and AC will be based on the Elastic stack[14]. The AA will be implemented as an Elastic Beat[15]. A Beat is a framework for a lightweight data transport to other components of the Elastic Stack, providing a standardized way of configuring the data transport and dealing with aspects like transport security over TLS and authentication. In our case, we will first customize a Beat to read from the audit logs generated by instances of services in the platform. Later, out Beat can be augmented to provide more advanced security features, as discussed shortly.

AAs will transport logs to a AC that is implemented as an Elasticsearch[16] instance. Elasticsearch is an analytics engine suitable for log and metrics analysis. To view logs stored in Elasticsearch, the platform dashboard will use a customized Kibana[17] user interface, as part of the Elastic stack. If particular functionality for securing logs (see later discussion under implementation) needs any centralized processing of logs, then we will add Logstash[18] as part of the AC component before logs are sent to Elasticsearch.

#### *5.2.1.2 Deployment and configuration*

The collector component will be deployed as one or more dedicated containers by the platform operator. The operator needs to provide the location of the collector(s) and means for AAs to authenticate the AC(s) as part of the AA configuration. With Elastic Beats, this is preferably (for any sizeable deployment) done by setting up a local Certificate Authority (CA). AAs (beats) are then configured to resolve a (potentially internal) domain (or host) name and validate that the

[14] https://www.elastic.co/
[15] https://www.elastic.co/products/beats
[16] https://www.elastic.co/products/elasticsearch
[17] https://www.elastic.co/products/kibana
[18] https://www.elastic.co/products/logstash

certificate presented by the AC is signed by the local CA. An interim solution could be to make the AC(s) internal through a pod with a ClusterIP service.

There are a number of possible ways to deploy AAs such that they can access logs from an instance of a platform service running as a pod in Kubernetes[19]. We opt for the simple but recommended solution of a *node logging agent*. First, we require that all platform services must generate logs by writing to standard output and standard error[20]. This will ensure that Kubernetes can receive the logs from the pods and then write the logs to the node's filesystem. Next, using a DeamonSet[21] replica, we run one logging agent pod on each node. The logging agent pod contains one container that runs our AA component. Finally, the AA component reads logs from the filesystem and transports them to the AC(s).

The configuration of all AAs should take place as part of the orchestration in Kubernetes to deploy an AA at every node. In addition to knowing the location of the AC(s) and how to authenticate the connection for log transport, we require that an opaque blob of unspecified data (in the order of kilobytes) can also be provided to each AA as part of the configuration. This blob may be used to distribute cryptographic key material for use by AAs in making data authentic beyond the transport phase, i.e., when log data is, e.g., exported from Kibana for use by a third party for the sake of accountability or auditability.

We plan to deploy our platform auditing components in two phases. For the first phase, in M24, we will use any many off-the-shelf Elastic stack components as possible to make sure that we can create, collect, transport, save, and ultimately read logs. For the second and final phase, in M36, we will augment the AA and/or AC components to make all logs (i) tamper proof and (ii) verifiable—in terms of authenticity and time—by third parties. These modifications for M36 is what may require cryptographic key material distributed to each AA as part of their configuration, hence the requirement of an opaque blob as part of the configuration.

#### *5.2.1.3 Implementation constraints*

The primary implementation constraint imposed by our approach is that all platform services must log to standard output and standard error. The only noteworthy constraint for the logging components is that a Beat in Elastic for the AA is written in the programming language Go[22].

[19] https://kubernetes.io/docs/concepts/cluster-administration/logging/
[20] In Kubernetes, everything a containerized application writes to standard output and standard error is handled and redirected somewhere by the container engine used by Kubernetes. By default, when Docker is used, this results in logs written to /var/log/. We may want to look at creating a custom log driver for Kubernetes for M36, but for M24 we should try to stick to as much off-the-shelf as possible.
[21] https://kubernetes.io/docs/concepts/workloads/controllers/daemonset/
[22] https://github.com/elastic/beats/

#### *5.2.1.4 APIs*

The platform auditing components ingests arbitrary bytes from standard output and standard error of containers, and then transports the data in Elasticsearch. There are no other applicable public-facing APIs to document that we are sure of that we need.

We may consider exposing an API towards clients to download their logs as part of the platform API. This could be useful for clients to ingest into their logging infrastructure and also for increasing transparency towards data subjects. At the time of writing, we cannot think of any explicit information that would not be available from the agent auditing logs (described next) as-is, but this might become clearer as we progress in the project.

### 5.2.2 Agent auditing

Agent auditing should enable a *client* of the PAPAYA platform to (i) directly view logs of data processing performed by an agent, e.g., for debugging purposes, and (ii) easily collect and transport logs from the agent into the client's own logging infrastructure, should one be available. For (i), the minimal agent dashboard (see Section 3) supports viewing logs. For (ii), the AA (and AC if they wish) used for platform auditing will be completely reusable by a client. This is because logs are extracted from containers by AA and collected at ACs. The security modifications for M36 to make the AA and AC produce tamper proof and verifiable logs will be designed with agent auditing use in mind.

## 5.3 Key Manager

### 5.3.1 Main components and their relationships

The Key Manager (KM) component will carry out of managing the cryptographic material during the whole lifecycle of the PAPAYA project in the cases where it will be required.

The KM is accessible over a network and provides support to client app components of the system to store the different ccryptographic material necessary for performing the cryptographic operations securely throughout the regular business flows of the platform.

The cryptographic material that the Key Manager works with includes, among others, symmetric keys, public keys, private keys, and certificates. Furthermore, the cryptographic objects may be of specific types according to particular needs of the components (end points) that uses them.

In addition, the Key Manager unifies the key management for local and remote endpoints. Especially, we take into consideration that the KM provides services compliant with Key Management Interoperability Protocol (KMIP)[23], which has been standardized by OASIS[24].
The Key Manager consists of three main components:

[23] https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=kmip
[24] https://www.oasis-open.org/

- **Key Manager Server** (KM Server): this component is a server based on Barbican[25] and provides a REST API designed for the secure storage, provisioning and management of secrets such as passwords, encryption keys and X.509 Certificates. It is aimed at being useful for all environments, including large ephemeral Clouds.
- **Key Manager Database** (KM Database). The database is a configured PostgreSQL instance, that will allow the management of the cryptographic material. The KM Server will provide an interface and database session for accessing the database.
- **Key Manager Client** (KMClient). This library is designed to be easily integrated within the clients components and it is composed of the following parts:
    - The main service KMClient, which needs the following parameters: the KMServer URL and an authorization token. It responsible of the connection between the client components and the KM Server
    - The models, bean objects which adapt the dataset that the KM Server needs to create the secrets such as cryptographic material name, cryptographic material type.
    - The API's services such as add a cryptographic material or delete a cryptographic material.

Using these components, an http request can be created by any component of PAPAYA and access to the KM Server services. Thereby the KM Server will verify the token provided by the client component within the request connecting to the corresponding IAM on the Data Controller Facilities, checking that it is authenticated and authorized and performs the requested service.

The following diagram (Figure 44) shows the relationship between the KM components and their integration in PAPAYA.

It is worth mentioning that the KM services will be provided to the Data Controller Controller components, hence the Data Controller will not only host the KM services on its premises (to have the control on the cryptographic material) but also control the access to the KM services.

[25] https://wiki.openstack.org/wiki/Barbican

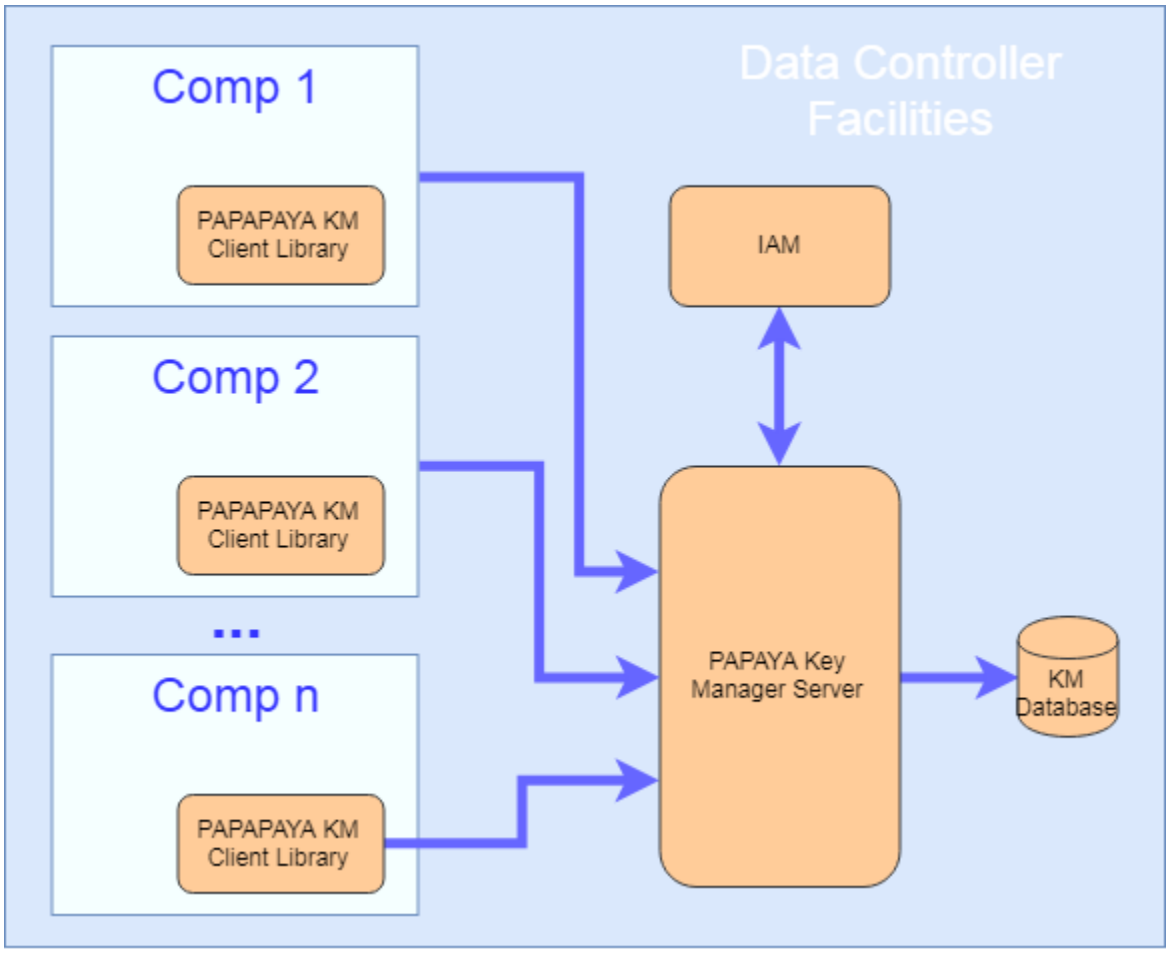



Figure 44: Key Manager components and their integration in PAPAYA.

### 5.3.2 Deployment and configuration

**KM Server** should be reached from the client app that would require any operation related with secrets; hence, it will not be accessible from the exterior. The access to the KM Server is performed through the KM Client via http request. The KM Server deployment will be based on Barbican deployment adapting it and configured for client needs. Thereby for this component it is necessary to define the correspondent service on the final deployment configuration (using Kubernates, Docker Compose or any other technology supporting Dockers).

**KM Database** should be reached only from KM Server (and not from other exterior components). The installation will be a PostgreSQL installation and configuration. For this component, it is necessary to define the correspondent service on the final Kubernetes deployment configuration.

**KM Client** is a library that should be imported in the component that needs to be able to call the KM to ask for their services. In addition, this component should be configured with the URL of KM Server in their properties file.

### 5.3.3 Implementation constraints

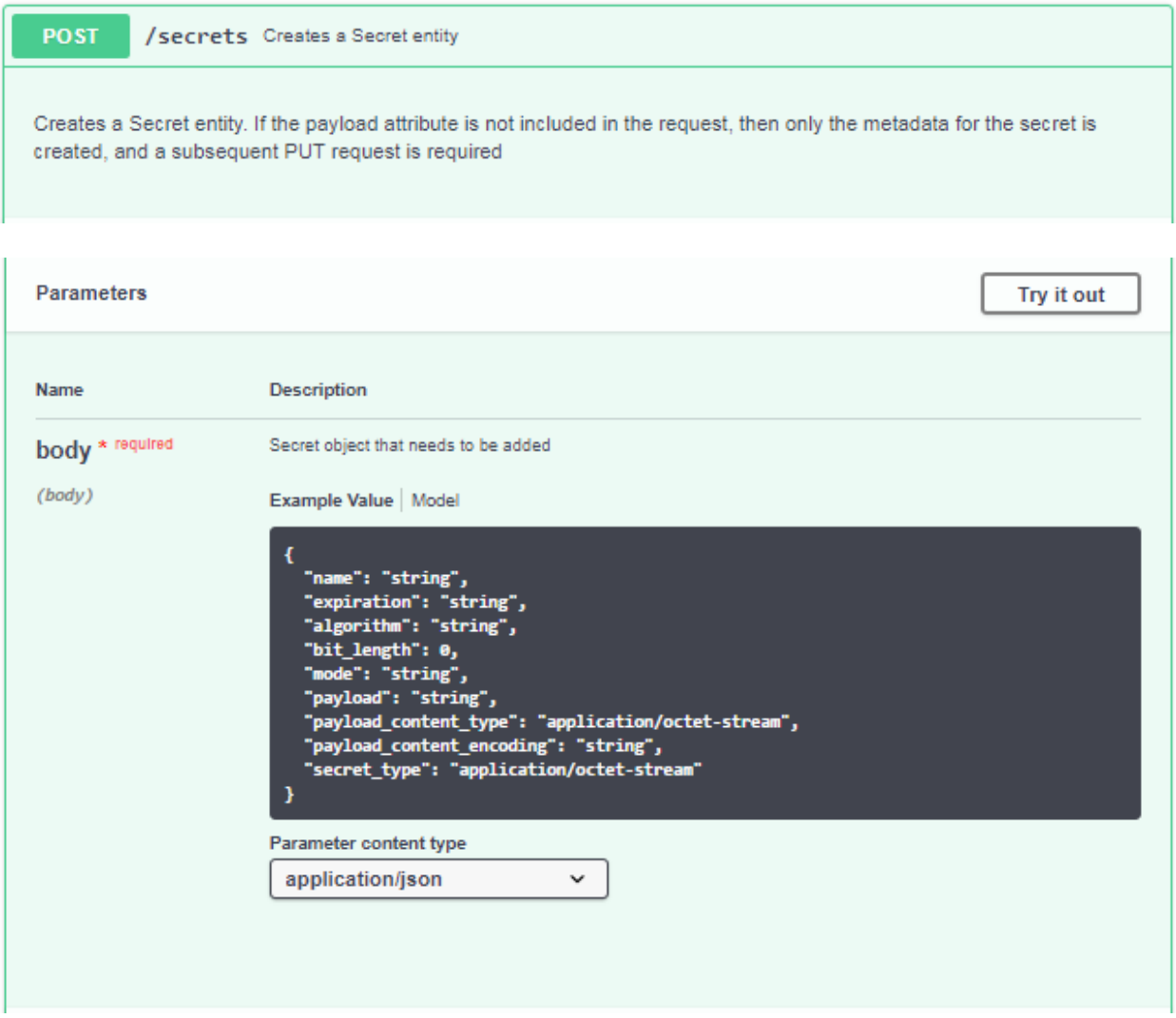
The cryptographic management tools selected to be applied into the PAPAYA project is bases in a wide known and used solution. Hence it is not foreseen any important constrains. Just in the case too many components will request the services of the KM Server, exceeding the KM capacities, it could concur on be a bottleneck effect. However, this eventuality has an easy solution by incrementing the number of nodes providing the KM Servers services.

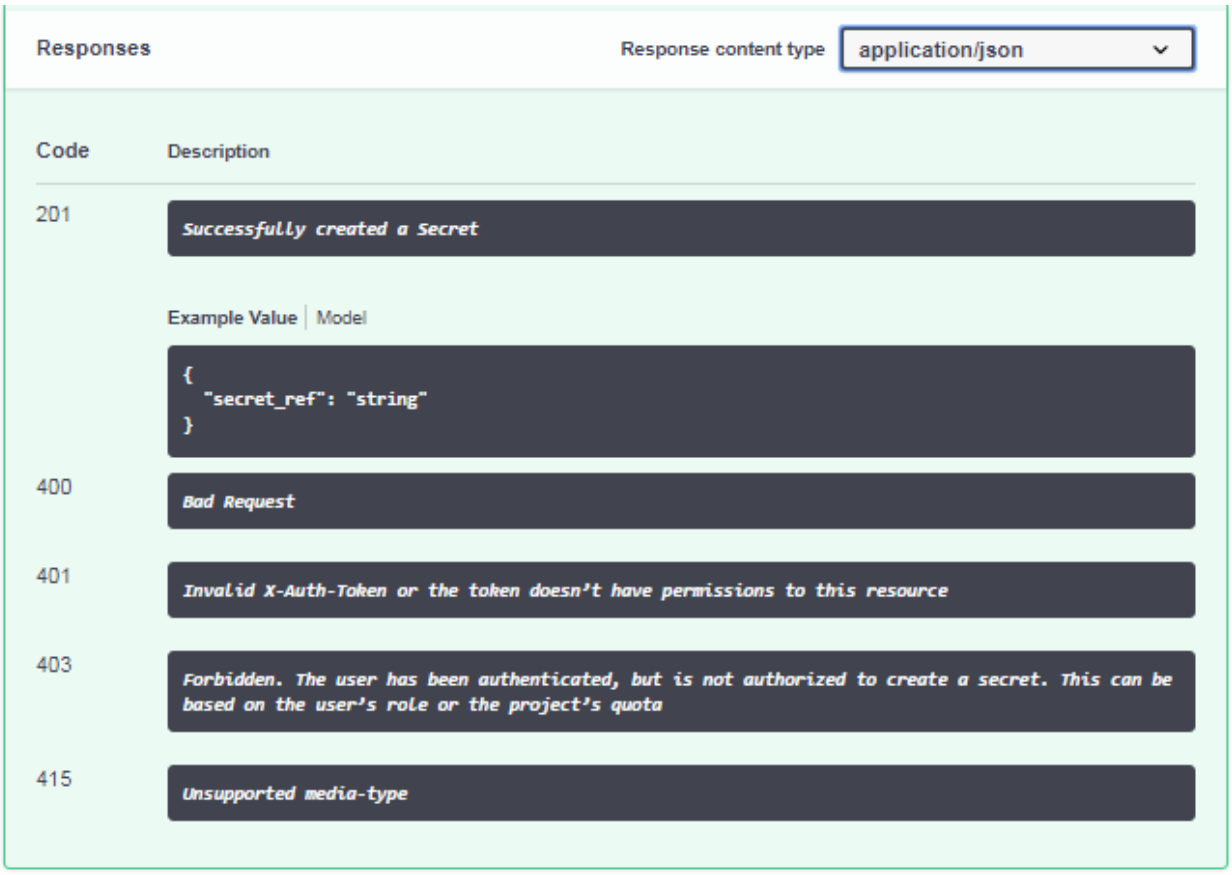
### 5.3.4 APIs

The Key Manager will provide a REST API that will allow to storage and to retrieve the cryptographic material.

The KM Server provides the following services to manage secrets:

## POST /secrets



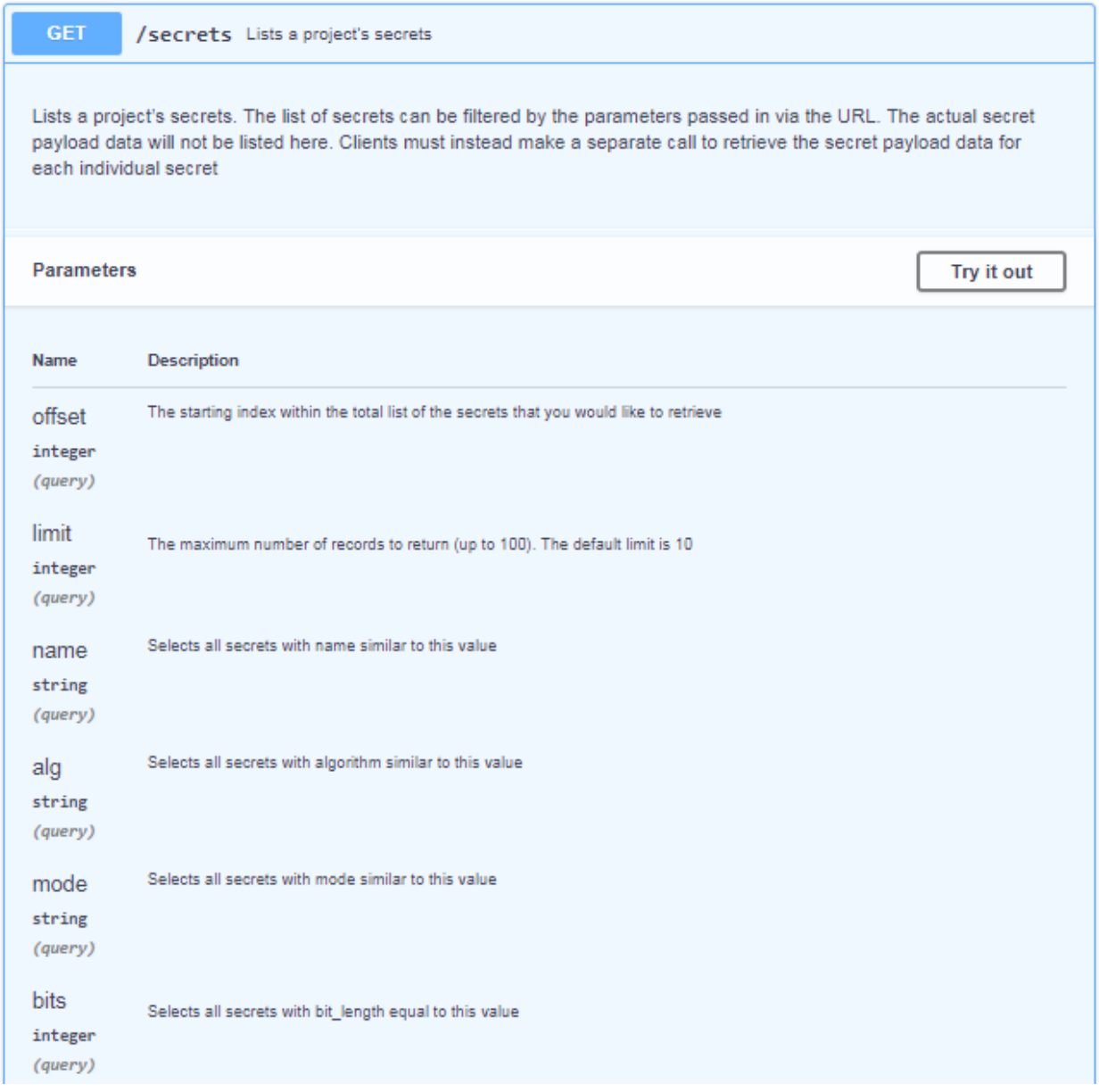


Responses

Response content type: application/json

| Code | Description |
|---|---|
| 201 | *Successfully created a Secret*<br>Example Value \| Model<br>`{ "secret_ref": "string" }` |
| 400 | *Bad Request* |
| 401 | *Invalid X-Auth-Token or the token doesn't have permissions to this resource* |
| 403 | *Forbidden. The user has been authenticated, but is not authorized to create a secret. This can be based on the user's role or the project's quota* |
| 415 | *Unsupported media-type* |

## GET /secrets



GET /secrets Lists a project's secrets

Lists a project's secrets. The list of secrets can be filtered by the parameters passed in via the URL. The actual secret payload data will not be listed here. Clients must instead make a separate call to retrieve the secret payload data for each individual secret

Parameters — Try it out

| Name | Description |
|---|---|
| offset<br>integer<br>*(query)* | The starting index within the total list of the secrets that you would like to retrieve |
| limit<br>integer<br>*(query)* | The maximum number of records to return (up to 100). The default limit is 10 |
| name<br>string<br>*(query)* | Selects all secrets with name similar to this value |
| alg<br>string<br>*(query)* | Selects all secrets with algorithm similar to this value |
| mode<br>string<br>*(query)* | Selects all secrets with mode similar to this value |
| bits<br>integer<br>*(query)* | Selects all secrets with bit_length equal to this value |

| | | |
|---|---|---|
| secret_type<br>string<br>*(query)* | Selects all secrets with secret_type equal to this value | |
| acl_only<br>boolean<br>*(query)* | Selects all secrets with an ACL that contains the user. Project scope is ignored | |
| created<br>string<br>*(query)* | Date filter to select all secrets with created matching the specified criteria. The values for this parameter are comma separated lists of time stamps in ISO 8601 format. The time stamps can be prefixed with any of these comparison operators: gt: (greater-than), gte: (greater-than-or-equal), lt: (less-than), lte: (less-than-or-equal). | |
| updated<br>string<br>*(query)* | Date filter to select all secrets with created matching the specified criteria. The values for this parameter are comma separated lists of time stamps in ISO 8601 format. The time stamps can be prefixed with any of these comparison operators: gt: (greater-than), gte: (greater-than-or-equal), lt: (less-than), lte: (less-than-or-equal). | |
| expiration<br>string<br>*(query)* | Date filter to select all secrets with expiration matching the specified criteria. The values for this parameter are comma separated lists of time stamps in ISO 8601 format. The time stamps can be prefixed with any of these comparison operators: gt: (greater-than), gte: (greater-than-or-equal), lt: (less-than), lte: (less-than-or-equal). | |
| sort<br>string<br>*(query)* | Determines the sorted order of the returned list. The value of the sort parameter is a comma-separated list of sort keys. Supported sort keys include created, expiration, mode, name, secret_type, status, and updated. Each sort key may also include a direction. Supported directions are :asc for ascending and :desc for descending. The service will use :asc for every key that does not include a direction. | |

Responses | Response content type: application/json

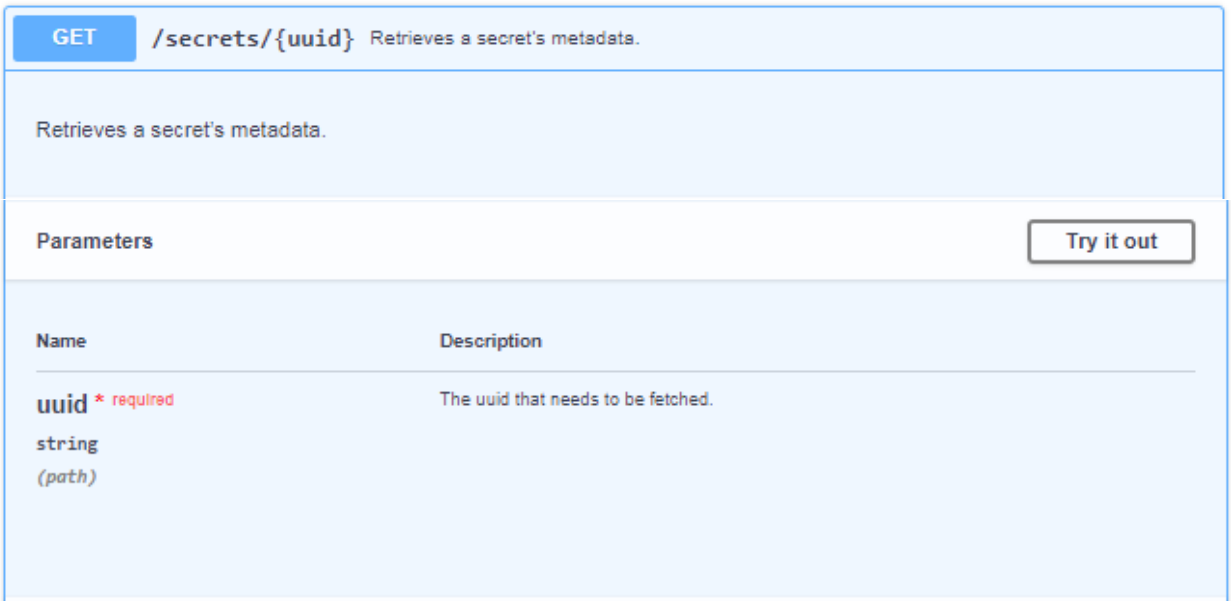
| Code | Description |
|---|---|
| 200 | *Successfull request* |

Example Value | Model


```
{
  "secrets": [
    {
      "name": "string",
      "expiration": "string",
      "algorithm": "string",
      "bit_length": 0,
      "mode": "string",
      "payload": "string",
      "payload_content_type": "application/octet-stream",
      "payload_content_encoding": "string",
      "secret_type": "application/octet-stream"
    }
  ],
  "total": 0,
  "next": "string",
  "previous": "string"
}
```


| | |
|---|---|
| 401 | *Invalid X-Auth-Token or the token doesn't have permissions to this resource* |

## GET /secrets/{uuid}



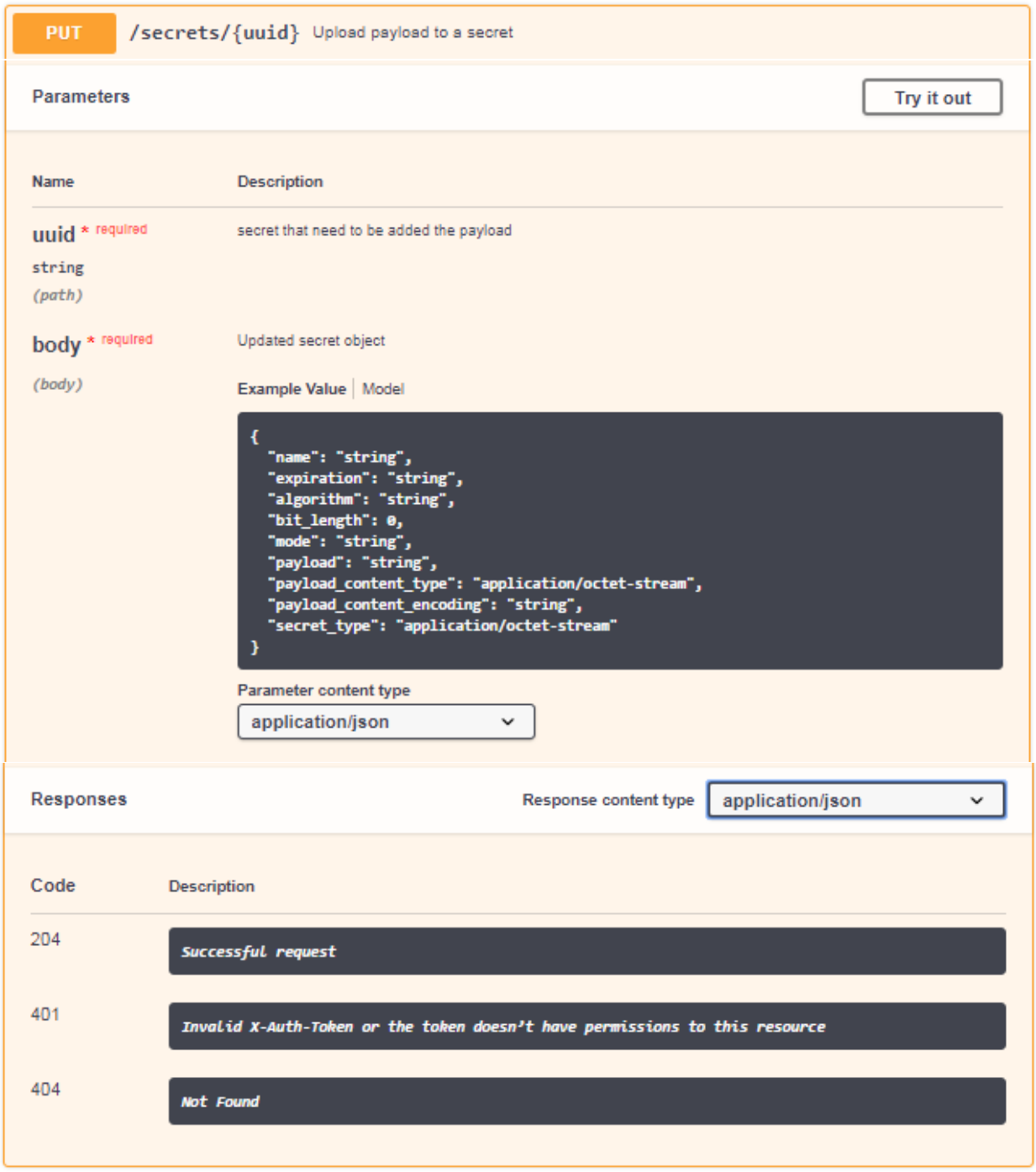
Responses | Response content type: application/json

| Code | Description |
|---|---|
| 200 | *Successful request* |
| 401 | *Invalid X-Auth-Token or the token doesn't have permissions to this resource* |
| 404 | *Not Found* |
| 406 | *Not Acceptable* |

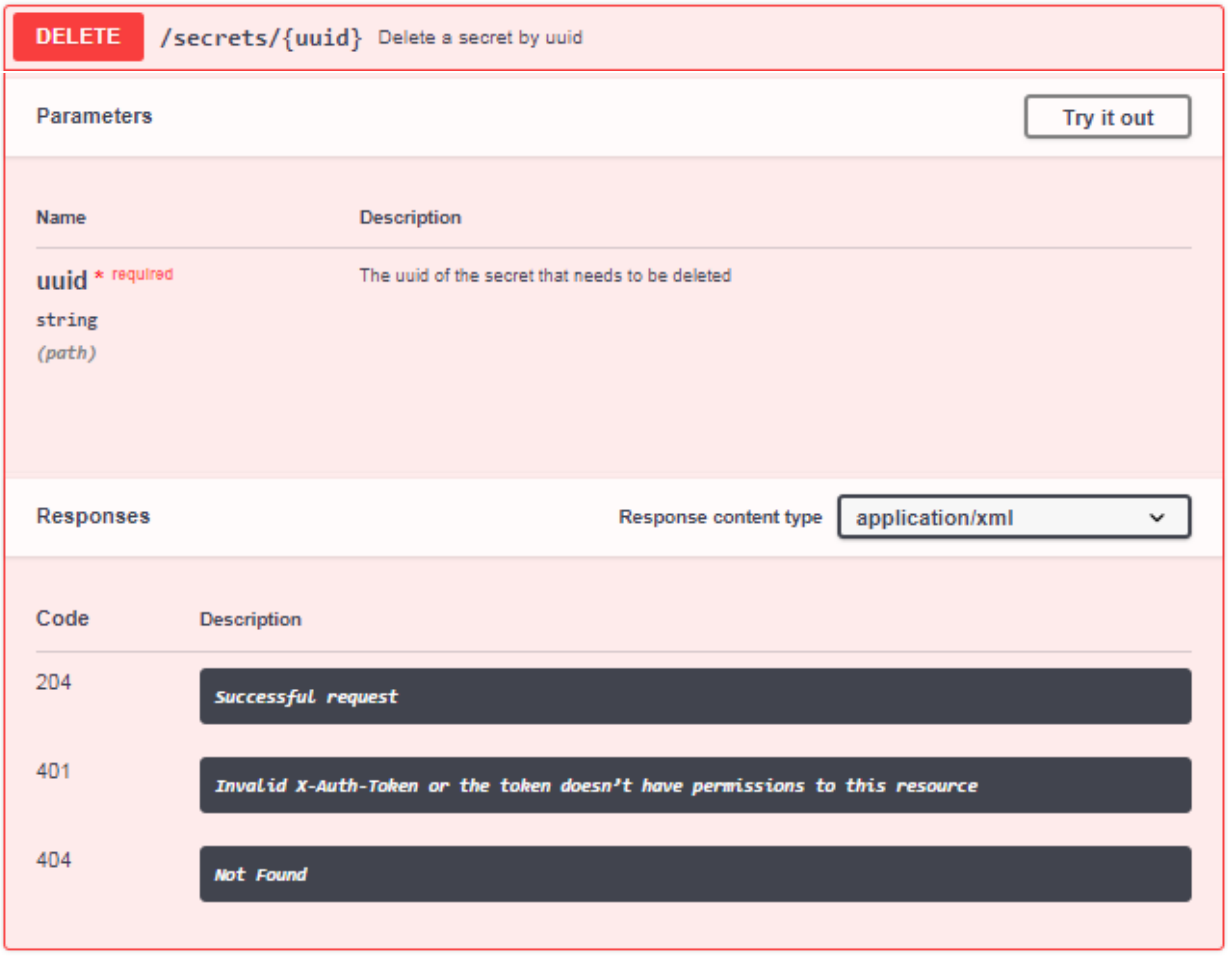
Example Value | Model

```
{
  "status": "string",
  "created": "string",
  "updated": "string",
  "expiration": "string",
  "algorithm": "string",
  "bit_length": 0,
  "mode": "string",
  "name": "string",
  "secret_ref": "string",
  "secret_type": "application/octet-stream",
  "content_types": "string"
}
```

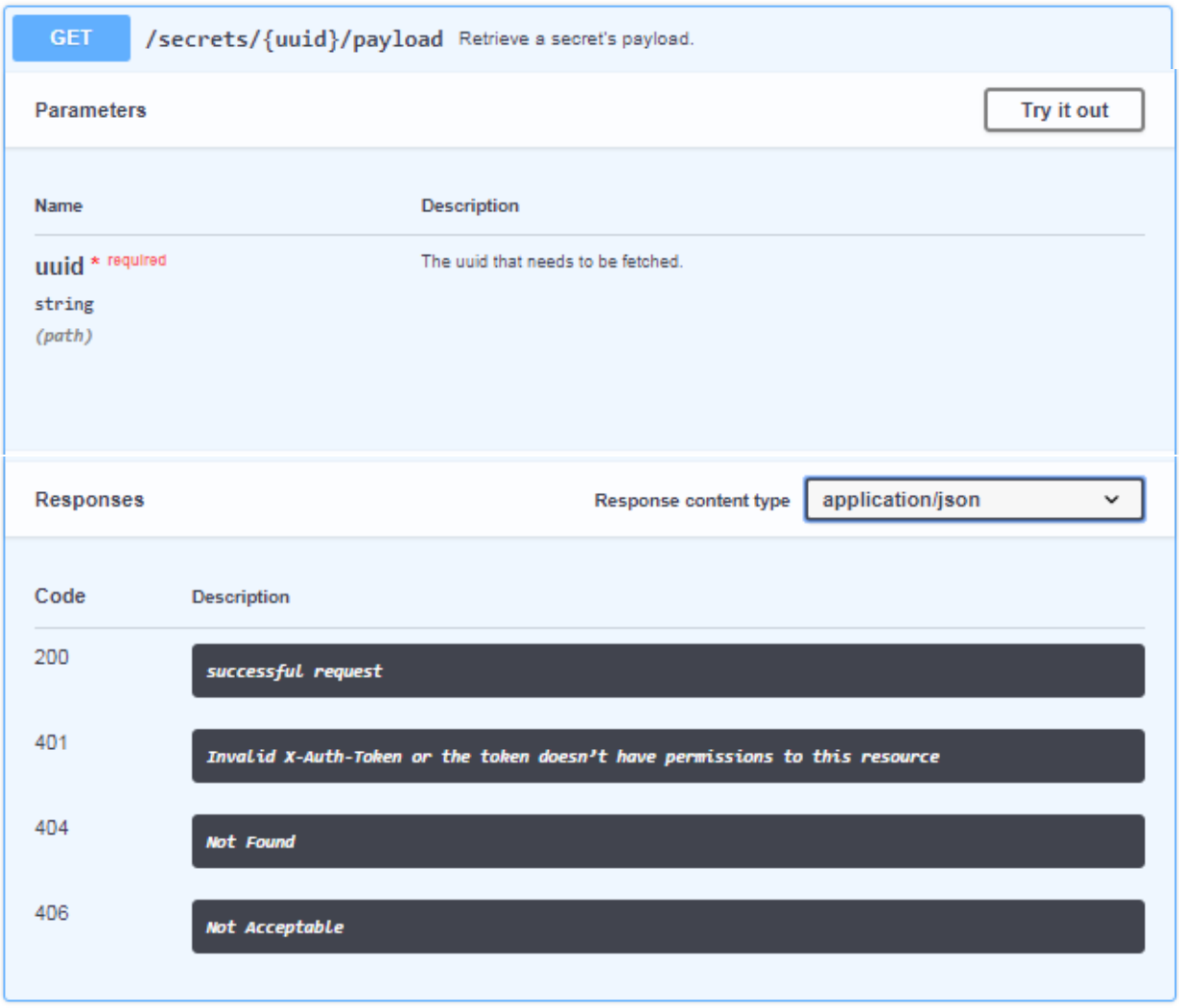
## PUT /secrets/{uuid}



## DELETE /secrets/{uuid}



## GET /secrets/{uuid}/payload



# 6 PAPAYA Dashboards

There are two dashboards in PAPAYA: the platform dashboard and the agent dashboard. The platform and agent dashboards are, as the names suggest, tied to the respective architectural components and their respective back-ends provide for the dashboards. The dashboards are accessed through web views in a web browser by their respective target users.

## 6.1 Platform Dashboard

Platform dashboard will be implemented as a Web application hosted in a container that will run on the PAPAYA's K8s cluster. The dashboard will provide the following functionality:

1. Present a list of services provided by the platform (Service's Catalog)
2. Add/Edit/Delete services.
3. Create/Deploy/Delete application (a dedicated instance of each of the provided services).
4. Allow application's owners to monitor the flow of the application by presenting operational logs.

### 6.1.1 Main components and their relationships

The Platform Dashboard consists of two main components (see Figure 45): the Web-UI and the backend.

**The Web-UI** will provide interface to service's catalog, applications list and application operational log.

**The backend** will be implemented as a python Flask[26] application. Flask is a microframework based on Werkzeug[27]. The Platform dashboard backend consists of five main modules.

- **The Service's Catalog** module is responsible of all actions related to service's management, such as adding a new service, editing or deleting the existing one. Editing and deletion will be allowed only to the service author (provider) or platform administrator. Each service should describe the required configuration and setup preferences. Prior to a service creation, the service provider will upload two images to PAPAYA's container registry, one image for the agent and the other for the server. The service provider will add the service in the platform dashboard (services catalog tab) and will provide all the required information such as images name, the ports exposed on both containers, service description etc.
- **The Application module** is responsible for application management. The platform client will choose a service of interest and provide all required configurations for the service. The server will create a dedicated application for the client. The client will be able to activate, terminate or delete the application. After application activation the dashboard will present an application URL, through which the communication with the application should be

[26] http://flask.pocoo.org/
[27] https://werkzeug.palletsprojects.com/en/0.14.x/

performed. In addition, the dashboard will allow the platform client to download a configuration file for the agent container, where the server_host and server_port are defined. The client will pull the agent container from the CR and deploy it. The deletion of application is not allowed while the application is running.

- **The Logger module** is responsible of the communication with the Loggingbeat component. Via the logger module, the dashboard will provide operational logs to the platform administrators and for the applications owners. The Loggingbeat is responsible for harvesting the application's operational logs from their standard output and standard error channels and shipping them to Elasticsearch[28] (ES) component and retrieve the logs from the ES for presentation in the Platform Dashboard. The Loggingbeat component will be described in details in Section 5.2.
- **K8s client module** allows the execution of Kubernetes commands. This component links between the platform dashboard and the Kubernetes cluster, where the applications and platform itself will run.
- **DB component** will communicate with the relational database where the users', services' and applications' data will be stored. The DB can be in one of two ways: provided as an external cloud service or DB running in a container using persistent volume.

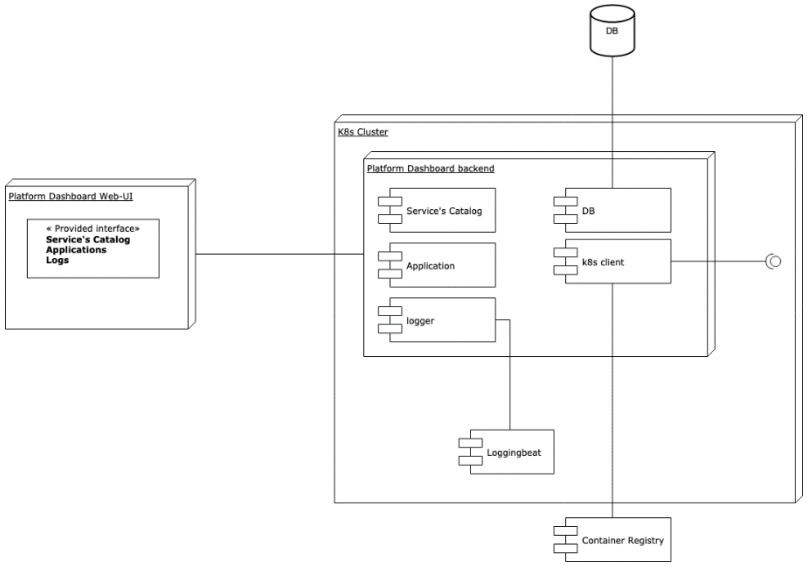



*Figure 45: Preliminary platform dashboard design*

### 6.1.2 Deployment and configuration

The platform administrator will deploy the platform dashboard on the Kubernetes cluster and provide all the required configurations and appropriate credentials, such as DB credentials, ES

[28] https://www.elastic.co/

credentials and container registry[29] (CR) credentials. The platform dashboard will receive all the relevant credentials using Kubernetes secrets objects[30]. Prior to dashboard deployment, the platform administrator will create a dedicated Kubernetes service account and provide cluster-admin privileges to that account. Platform dashboard will run with that service account name. Applications deployed by the platform dashboard will run under a default service account name (without cluster admin privileges).

### 6.1.3 Implementation constraints

The dashboard assumes that the user that sends request is already authenticated (by Identity and Access Management (IAM) mechanism described in Section 5.1).

### 6.1.4 APIs

The platform dashboard will provide the following views:

1. Application
2. Service (Services Catalog)
3. Logs

## 6.2 Agent Dashboard

The agent dashboard is intended for use by a client during development and operation of a PAPAYA agent. The dashboard provides two views, one for viewing the data processing logs from an agent and another view showing the configuration of an agent.

### 6.2.1 Main components and their relationships

The agent dashboard consists of a backend and a frontend component. The backend exposes a HTTP server on localhost that provides the two views as web-views using the frontend component running in a browser used by the client.

### 6.2.2 Deployment and configuration

The agent dashboard will be deployed as a standalone binary downloadable from the Platform dashboard. The binary is intended to be run on-demand from the command line by a client as part of development. We leave more advanced deployment as part of operations to clients. On launch, the binary takes two configuration options: the identifier for the container to directly read logs from (using Docker logs), and the filesystem path to the agent's configuration file. The configuration file is provided either by the agent when the container is run or downloaded from the platform dashboard.

[29] https://www.ibm.com/cloud/container-registry
[30] https://kubernetes.io/docs/concepts/configuration/secret/

### 6.2.3 Implementation constraints

The agent dashboard should be lightweight and quick to run for a single developer, support multiple architectures, and be possible to distribute as a standalone binary. We intend to use Go as the programming language for these reasons.

### 6.2.4 APIs

Only an internal API between backend and frontend. The input to the agent dashboard is provided as part of running the tool, as described in Section 6.2.2, and the tool provides no output beyond UIs.

# 7 Data Subject Toolbox

The data subject toolbox from PAPAYA consists of a number of independent or at most loosely coupled tools with the commonality that each tool provides a UI intended for data subjects. Using the toolbox, a client using the PAPAYA platform can integrate one or more of the tools in its data subject facing UIs to provide an integrated and seamless *data subject dashboard*. We opted for this design—instead of a monolithic dashboard targeting data subjects—ultimately for the sake of usability for data subjects, highlighted in requirements part of D2.2 [1]. Likely, most PAPAYA clients already have data subject facing applications that they would wish to augment and not completely replace with one or more of the functionalities provided by the data subject toolbox. Further, from D2.1 and D2.2 [1], we know that all data subject-facing UIs should be as part of mobile applications. Next, we describe the four tools part of the data subject toolbox.

## 7.1 Explaining Privacy-preserving Analytics

This tool explains to data subjects how the privacy-preserving data analytics using PAPAYA works and how the data controller has reasoned about the risks to the data subject due to this processing. The tool is integrated by the user of the PAPAYA platform (that is, the data controller and/or data processor) in its data subject facing UIs. The tool is appropriate to include as part of a consent or a privacy policy views in a UI. D3.2 provides more details on the purposes and technical design of the tool.

### 7.1.1 Main components and their relationships

The tool consists of two categories on independent components: one category for conveying how privacy-preserving analytics work and another for explaining risks. There will be at least one component per type of analytics developed in PAPAYA (see Platform Services in Section 4) and several components that explain different aspects of risks. Each component is designed to be completely standalone.

### 7.1.2 Deployment and configuration

For deployment, the components of the tool are intended to be completely deployed as a complied part of a mobile app provided by the platform user to data subjects.

Each component is configured by a JSON file, D3.2 contains further details on planned format (will be fully defined first in M24 as part of Task 3.3 in WP3).

For explaining how privacy-preserving analytics work the configuration file of the platform agent will be needed. We expect that the agent configuration specifies the type of analytic performed by the agent (e.g., collaborative training or apply neural network model) and relevant parameters for the analytics. We assume that this configuration file is generated by or provided together with each platform analytics agent as downloaded from the platform.

### 7.1.3 Implementation constraints

To make it as easy as possible to integrate the components into existing mobile apps we plan to use either Flutter or Angular for all components. Which of the two approaches to use will be determined during development based on consortium and developer preferences.

## 7.2 Data Disclosure Visualization Tool

The Disclosure Visualization Tool (DVT) visualizes what personal data a data subject has disclosed to different parties. This is particular important for privacy-preserving analytics, because what data is available to who might be counter-intuitive. DVT is therefore complementary to the tool for explaining privacy-preserving analytics. We have decided to split the tools to give as much flexibility as possible to PAPAYA platform users in choosing the most appropriate components.

### 7.2.1 Main components and their relationships

DVT consists of a single UI view that shows a playful and useful visualization of the provided data and their respective recipients.

### 7.2.2 Deployment and configuration

DVT is intended to be deployed as part of a mobile app, as for the explaining privacy-preserving analytics components. DVT is configured by either a JSON file or a couple of function pointers. The configuration consists of descriptions of disclosed data and their respective recipients. Support for function pointers enable a user of the DTV view to stream data, better supporting large datasets (e.g., for infinite scrolling of a timeline). The specific format will be based on a subset of the data model for the open source Data Track tool from the A4Cloud project[31].

### 7.2.3 Implementation constraints

DVT shares the same implementation constraints as Explaining Privacy-preserving Analytics.

## 7.3 Annotated Log View Tool

The Annotated Log Tool (ALT) provides an annotated log view for data subjects that aims to explain the actual data processing that have taken place on relevant personal data disclosed by the data subject.

### 7.3.1 Main components and their relationships

ALT consist of two components: an UI view component and an annotator component. The annotator component is to be called by a PAPAYA analytics user for each call an analytics agent. The component takes as input a list of data subject identifiers, the configuration of the analytics agent, and free-text descriptions of the performed processing and processed data. The component provides as output a map of data subject identifier-to-strings, where each string uses

[31] https://github.com/pylls/datatrack

an internal format. The output is intended to be stored, for each data subject identifier, together with a timestamp by the analytics user in whatever system it sees fit (e.g., most analytics users likely have a database of users already, or something similar). Section 4.3.4 provides further details on the input and output formats.

The UI view component takes as input a subsequence (sorted by time) of strings from the annotator component *for one data subject identifier* and provides as output a UI view that presents the recorded data processing.

### 7.3.2 Deployment and configuration

The UI view component of ALT is intended to be deployed in a mobile app just as the DVT and Explaining Privacy-preserving Analytics tools. The annotator component is intended to be integrated completely in the data processing software of the user of the PAPAYA platform. Our goal is to provide implementations of the annotator component in a number of programming languages to ease adoption.

There is no configuration for the annotator view—it is fully deterministic—therefore the UI view component is completely defined by its expected input generated by the annotator component.

### 7.3.3 Implementation constraints

The UI view component shares the same constraints as the DVT and Explaining Privacy-preserving Analytics tools. The annotator component must be possible to implement in a number of different programming languages. Likely, it is not motivated to distribute a simple function as a container.

### 7.3.4 APIs

ALT exposes one API call as part of the annotator component, intended to be called by the processing system for each use of an analytics agent. The API call will be implemented in a number of different programming language: hence, the data types described below are described in terms of expected requirements on the used type. The description of this API does not use the same format as the other APIs because it's not a RESTful API (which is what Swagger is intended for).
The API call takes as *mandatory* input:

- A *list of data subject identifiers*. The identifiers identify (within the system performing the data processing) each data subject whose data is being processed as part of one use of the analytics agent. The list type must be possible to enumerate and finite. Each identifier must be possible to use as a key in a key-value map provided as output (see below).
- The *configuration of the analytics agent*. The configuration MUST be in the format as provided by the analytics agent (likely a json string, to be defined).
- A *processing description*. The free-text processing description MUST be a string (or equivalent in the language) that succinctly describes the processing performed by using the analytics agent.

- A *data description*. The free-text data description MUST be a string that succinctly describes the type of processed data.

The API call also takes as optional input:

- A *list of purposes*. A list of integers, where each integer represents a purpose for the performed processing. A mapping between integers and purposes will be provided by the tool.

The API call provides as output for every call with the mandatory input where the list of data subject identifiers contains at least one identifier:
A *mapping of identifiers to UTF-8 strings*. For each identifier provided as input, the mapping contains an UTF-8 encoded string of variable length in an opaque format that describes the relevant data processing activity.

## 7.4 Privacy Engine

Ensures data subject access rights to their personal data (among others)

### 7.4.1 Main components and their relationships

The detailed design and architecture description of the PE and its subcomponents has been described within the deliverable *D3.2 Risk Management Artefacts for Increased Transparency*. So, the details of the services implemented by the Privacy-preserving Manager (PPM) and the Data Subject Rights Manager (DSRM) subcomponents can be consulted there. However, it will be necessary to extend the existing information detailed in order to define how this component will be integrated within the PAPAYA framework and the interaction between the PE and the other components of the system deployed within the Data Controller facilities. Although each use case will have a specific final architecture designed ad hoc, the Figure 46 shows the integration of the PE within the PAPAYA framework with certain level of abstraction that can be applied for any use case involved.

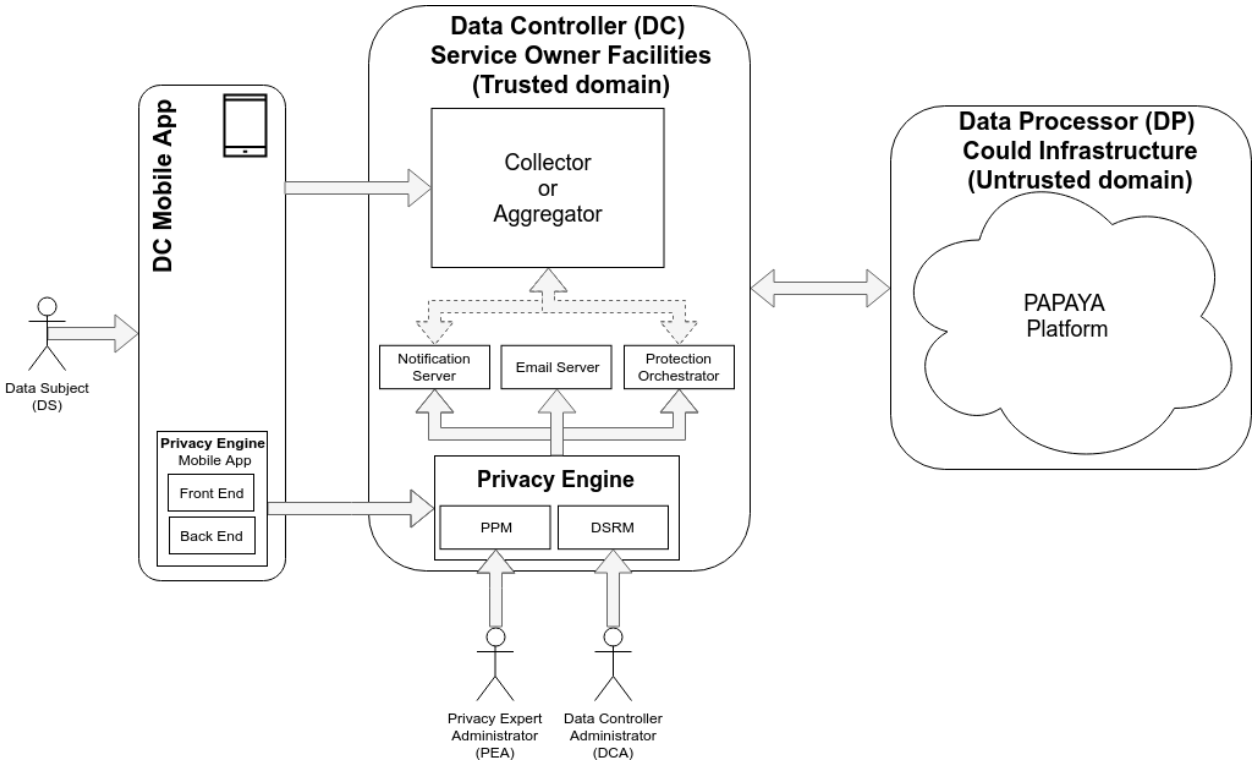



*Figure 46: Privacy Engine integration in the PAPAYA framework*

The diagram above shows three main domains: Data Controller (DC) mobile application, Data Controller Domain and Data Processor (DP) Domain. The main targets of each domain are as follows:

- *DC Mobile Application*: the DC will provide a mobile application to the Data Subject (DS). This mobile application will be the main interface of the DS with the system and will carry out the gathering of the data.

- *DC Domain:* The objective of this domain is to gather the data to be processed. Hence one of the main components of the whole business flow will be the Data Collector or Aggregator (some use cases will collect and some others will aggregate the data) devoted to harvest the data necessary for applying the selected analysis. Then it will pre-process the data using the corresponding PET in order it can be analyzed in the PAPAYA platform preserving the security and the privacy of the data of the DS. This domain is an environment controlled by the DC with the proper Privacy and Security measurements to protect the data, thereby it is considered as a trusted domain.

- *DP Domain:* This domain is considered as untrusted therefore all the data that it will access will be protected adequately. This domain will simulate an externalization of services on the

cloud by the DC. Therefore, the main target of this domain is taking advantage of the cloud infrastructure services to process a big amount of data.

As it can be seen on the previous figure, the PE will be integrated in two of those domains: in the mobile application and the DC Domain. It is worth highlighting that for preserving the security and the privacy of the data processed by underlying servers (Email Server, Notification Server and the Protection Orchestrator), the PE and its subcomponents will be deployed within the DC Domain in a controlled and trusted environment.

### 7.4.2 Deployment and configuration

As mentioned before, the different subcomponents of the PE and the underlying servers will be either part of a mobile application or deployed within the DC Domain.
With regards the subcomponents deployed within the mobile application, they have been designed taking into consideration the final integration on a root main application developed by the DC. For that purpose, the mobile subcomponents will be implemented using the following criteria:

- *Front End Services:* all the different interfaces for the DS as part of the PE will be developed based on HTML5+Javascript to easily be integrated with the root application just opening a web viewer on the native mobile implementation.

- *Back End Services:* The Consortium has agreed that the mobile application will be developed using the Android SO and framework. Therefore, in order to facilitate the integration of these services into the final mobile application, they will be provided in a Java library (JAR file) with a well define interface.

Regarding the services deployed within the DC facilities, the PE will be integrated within a Docker infrastructure, thereby facilitating the integration with the final deployment in the DC facilities (using Docker compose or Kubernetes technologies for the deployment). It is worth mentioning, that the PE will need the following relying services: the email server, notification server and the protection orchestrator. They will be deployed within the DC facilities to help on preserving the security and privacy of some the data processed. In order to ease the final deployment, these relying services will be encapsulated in Dockers as well.

### 7.4.3 Implementation constraints

As main implementation constraint of the design defined for the development of the PE and deployment plan described above is that the library developed for the back end mobile application (JAR file) is strongly tied to the use of an Android OS framework, been more difficult to integrate it on other mobile application frameworks.

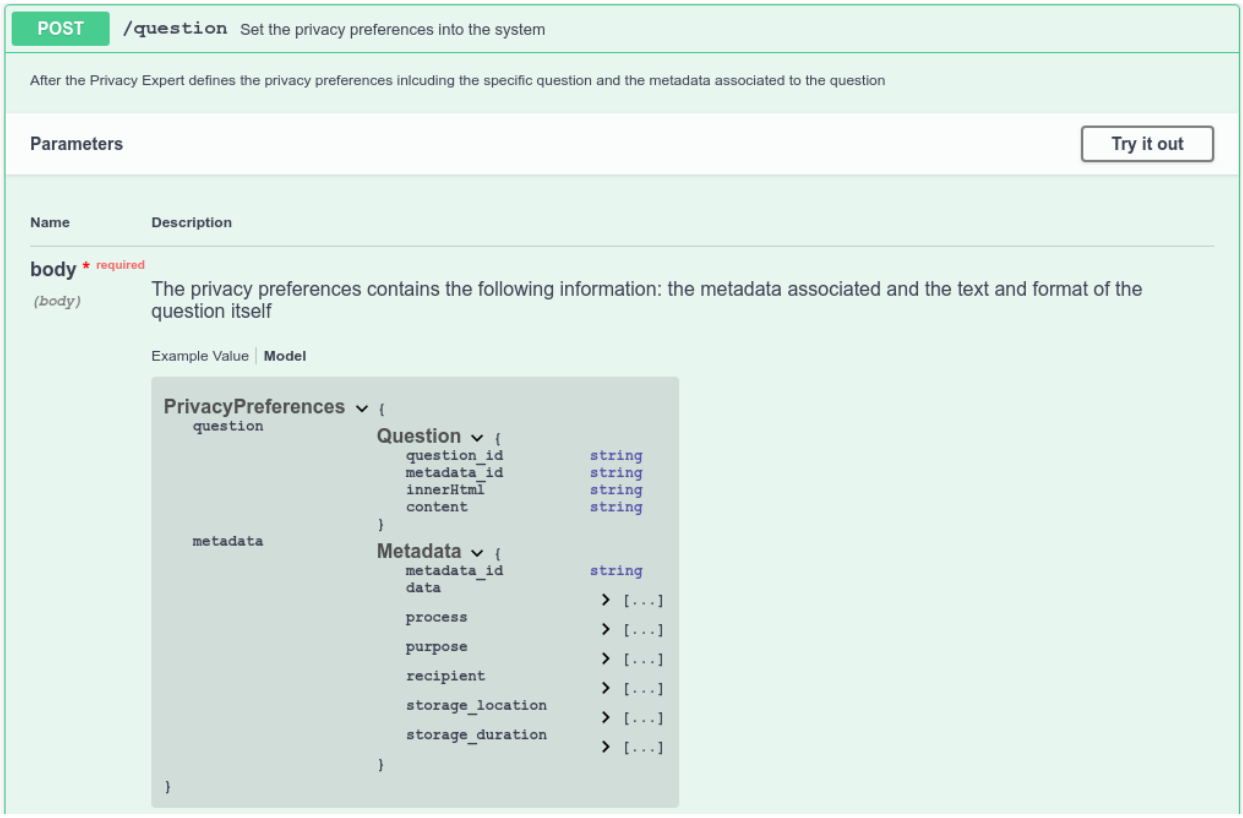
### 7.4.4 APIs

This section defines in detail the application programming interface of the both main functionalities that compose the Privacy Engine. The definition of the interfaces of the different subcomponents

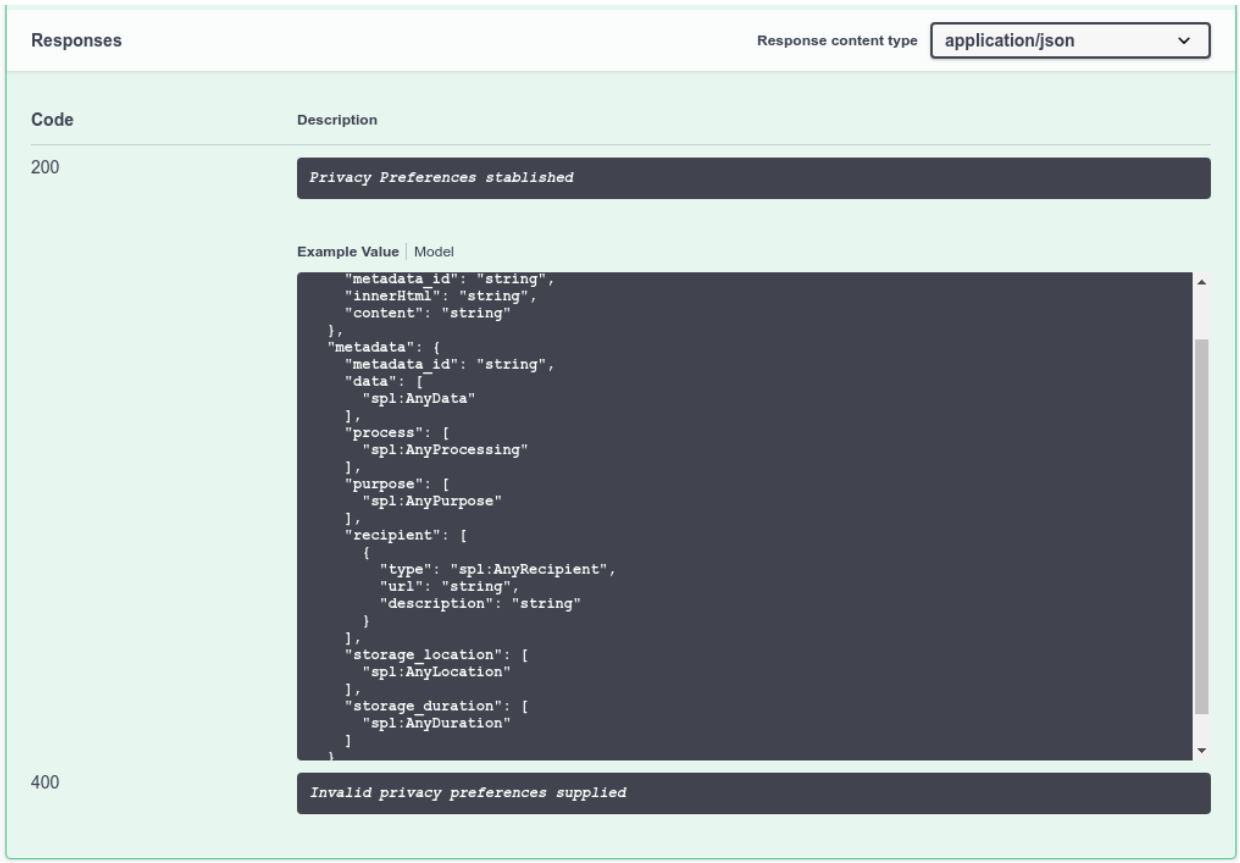
of the Privacy Engine has been developed using the open source tool Swagger[32]. The outputs obtained from this tool are as follows.

#### *7.4.4.1 Privacy Preferences Manager (PPM) API*

**POST /question**




[32] https://swagger.io/



Responses

Response content type: application/json

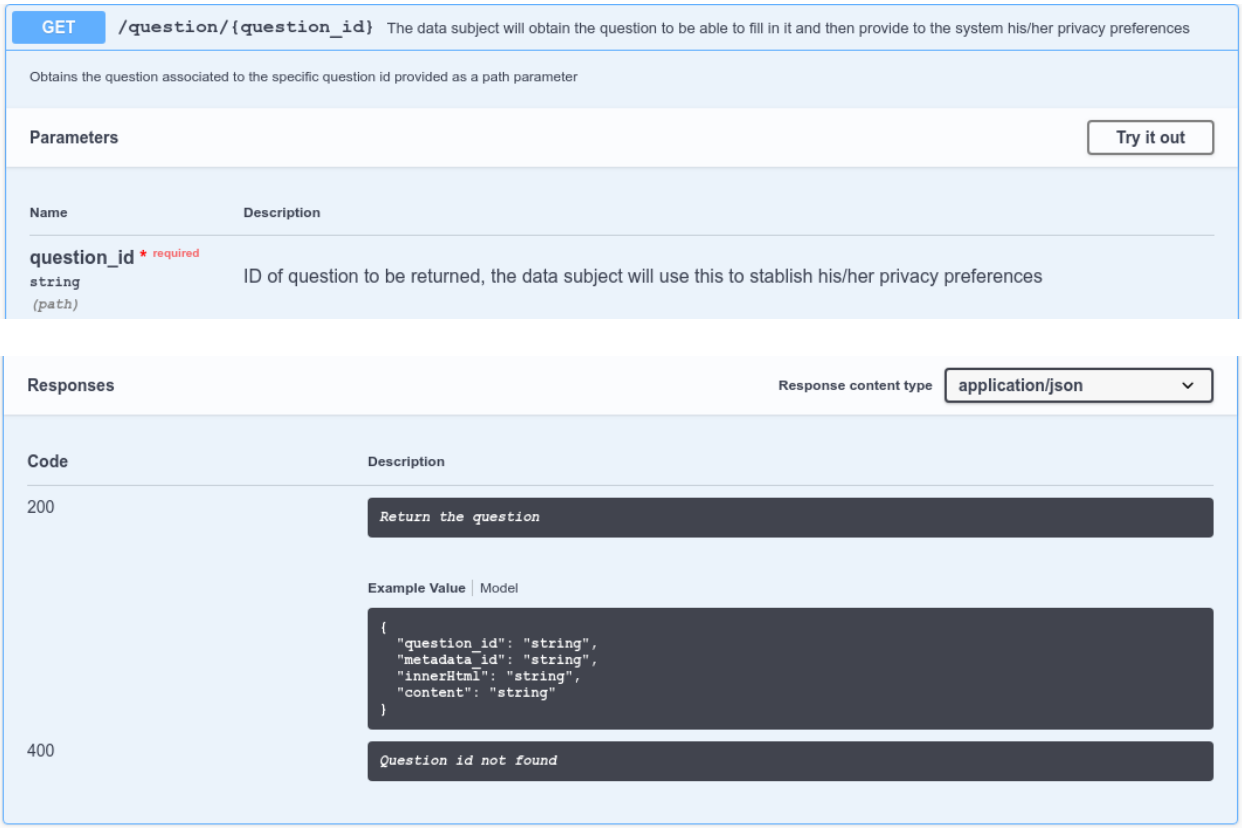
| Code | Description |
|---|---|
| 200 | *Privacy Preferences stablished* |

Example Value | Model

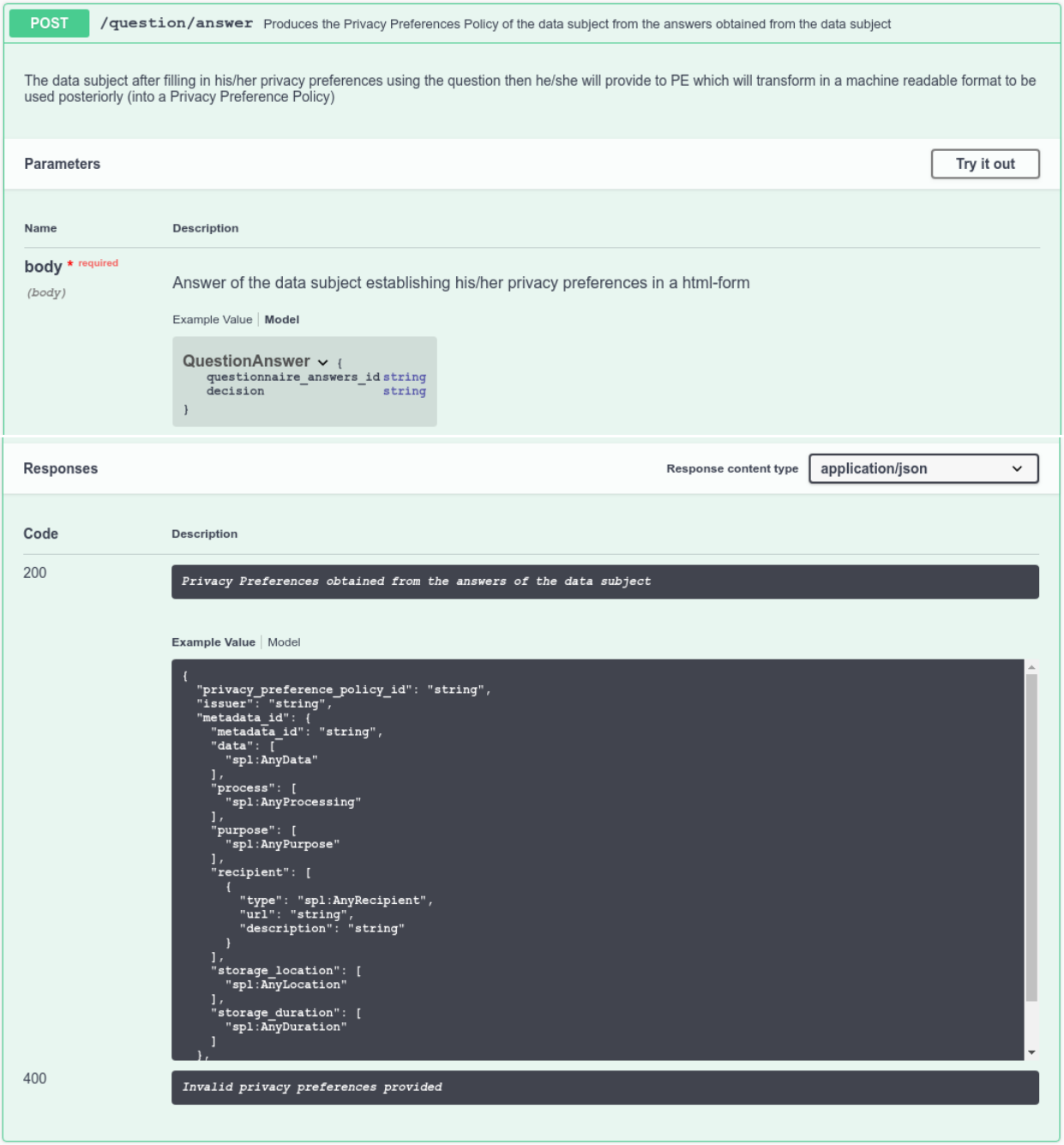

```
    "metadata_id": "string",
    "innerHtml": "string",
    "content": "string"
  },
  "metadata": {
    "metadata_id": "string",
    "data": [
      "spl:AnyData"
    ],
    "process": [
      "spl:AnyProcessing"
    ],
    "purpose": [
      "spl:AnyPurpose"
    ],
    "recipient": [
      {
        "type": "spl:AnyRecipient",
        "url": "string",
        "description": "string"
      }
    ],
    "storage_location": [
      "spl:AnyLocation"
    ],
    "storage_duration": [
      "spl:AnyDuration"
    ]
  }
```


| | |
|---|---|
| 400 | *Invalid privacy preferences supplied* |

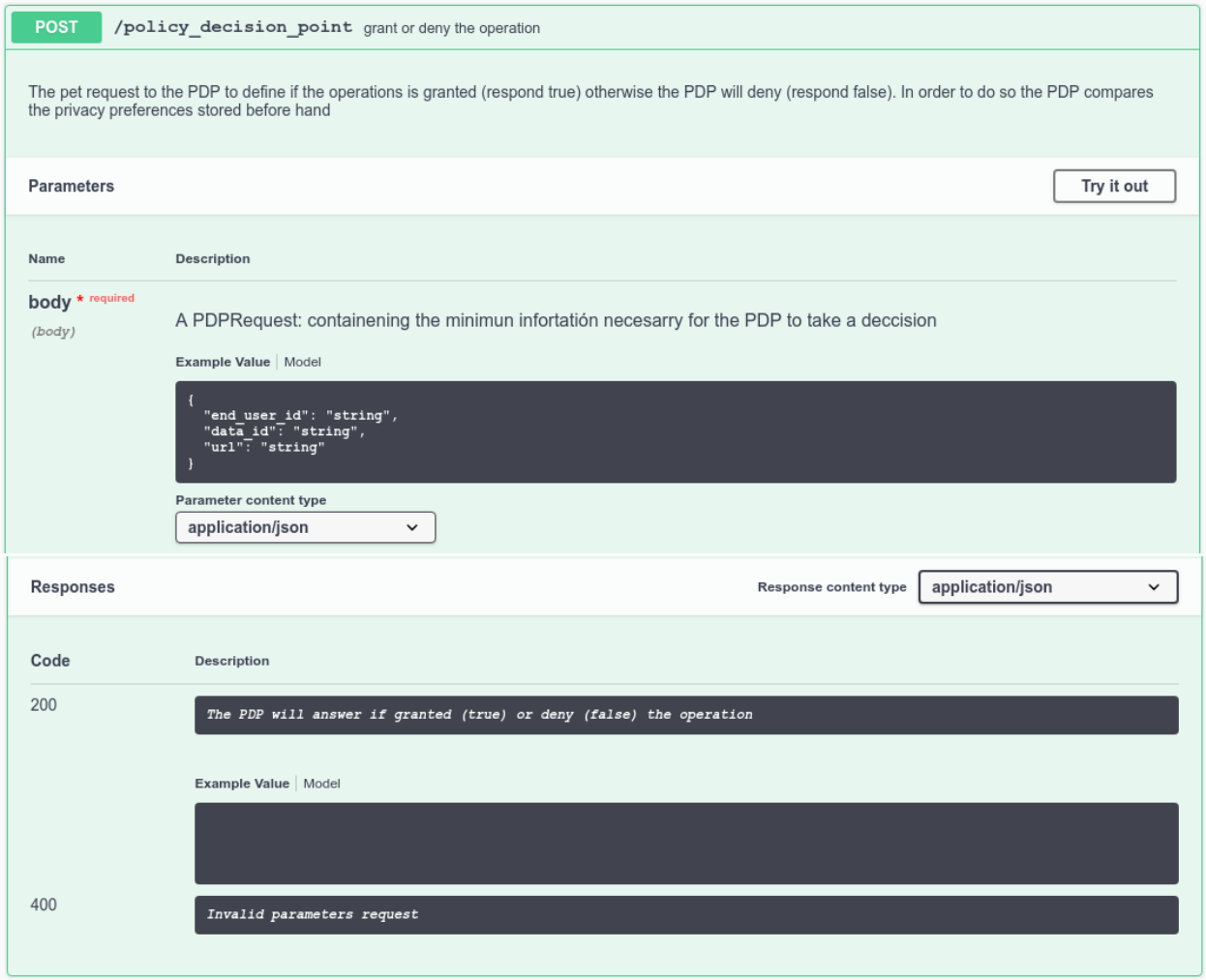
## GET /question/{question_id}



## POST /question/answer



## POST /policy_decision_point



## Models

**PrivacyPreferences** {
- question → Question {...}
- metadata → Metadata {...}

}

**Question** {

| | |
|---|---|
| question_id | string |
| metadata_id | string |
| innerHtml | string |
| content | string |

}

**PrivacyPreferencesPolicy** {

| | |
|---|---|
| privacy_preference_policy_id | string |
| issuer | string |
| metadata_id | Metadata { |
| | metadata_id: string |
| | data: [...] |
| | process: [...] |
| | purpose: [...] |
| | recipient: [...] |
| | storage_location: [...] |
| | storage_duration: [...] |
| | } |
| time_creation | string($date-time) |
| time_update | string($date-time) |

}

**QuestionAnswer** {

| | |
|---|---|
| questionnaire_answers_id | string |
| decision | string |

}

**PDPRequest** {

| | |
|---|---|
| end_user_id | string |
| data_id | string |
| url | string |

}

**Metadata** {

| | |
|---|---|
| metadata_id | string |
| data | [...] |
| process | [...] |
| purpose | [...] |
| recipient | [...] |
| storage_location | [...] |
| storage_duration | [...] |

}

**Purpose** string
Enum:

[ spl:AnyPurpose, svpu:Account, svpu:Admin, svpu:AnyContact, svpu:Arts, svpu:AuxPurpose, svpu:Browsing, svpu:Charity, svpu:Communicate, svpu:Current, svpu:Custom, svpu:Delivery, svpu:Develop, svpu:Downloads, svpu:Education, svpu:Feedback, svpu:Finmgt, svpu:Gambling, svpu:Gaming, svpu:Government, svpu:Health, svpu:Historical, svpu:Login, svpu:Marketing, svpu:News, svpu:OtherContact, svpu:Payment, svpu:Sales, svpu:Search, svpu:State, svpu:Tailoring, svpu:Telemarketing ]

**Process** string
Enum:

[ spl:AnyProcessing, svpr:Aggregate, svpr:Analyze, svpr:Anonymize, svpr:Collect, svpr:Copy, svpr:Derive, svpr:Move, svpr:Query, svpr:Transfer ]

**Data** string
Enum:

[ spl:AnyData, svd:Activity, svd:Anonymized, svd:AudiovisualActivity, svd:Computer, svd:Content, svd:Demographic, svd:Derived, svd:Financial, svd:Government, svd:Health, svd:Interactive, svd:Judicial, svd:Location, svd:Navigation, svd:Online, svd:OnlineActivity, svd:Physical, svd:PhysicalActivity, svd:Political, svd:Preference, svd:Profile, svd:Purchase, svd:Social, svd:State, svd:Statistical, svd:TelecomActivity, svd:UniqueId ]

**Recipient** {

| | |
|---|---|
| type | string<br>Enum:<br>[ spl:AnyRecipient, svr:Delivery, svr:OtherRecipient, svr:Ours, svr:Public, svr:Same, svr:Unrelated ] |
| url | string |
| description | string |

}

**StorageLocation** string
Enum:

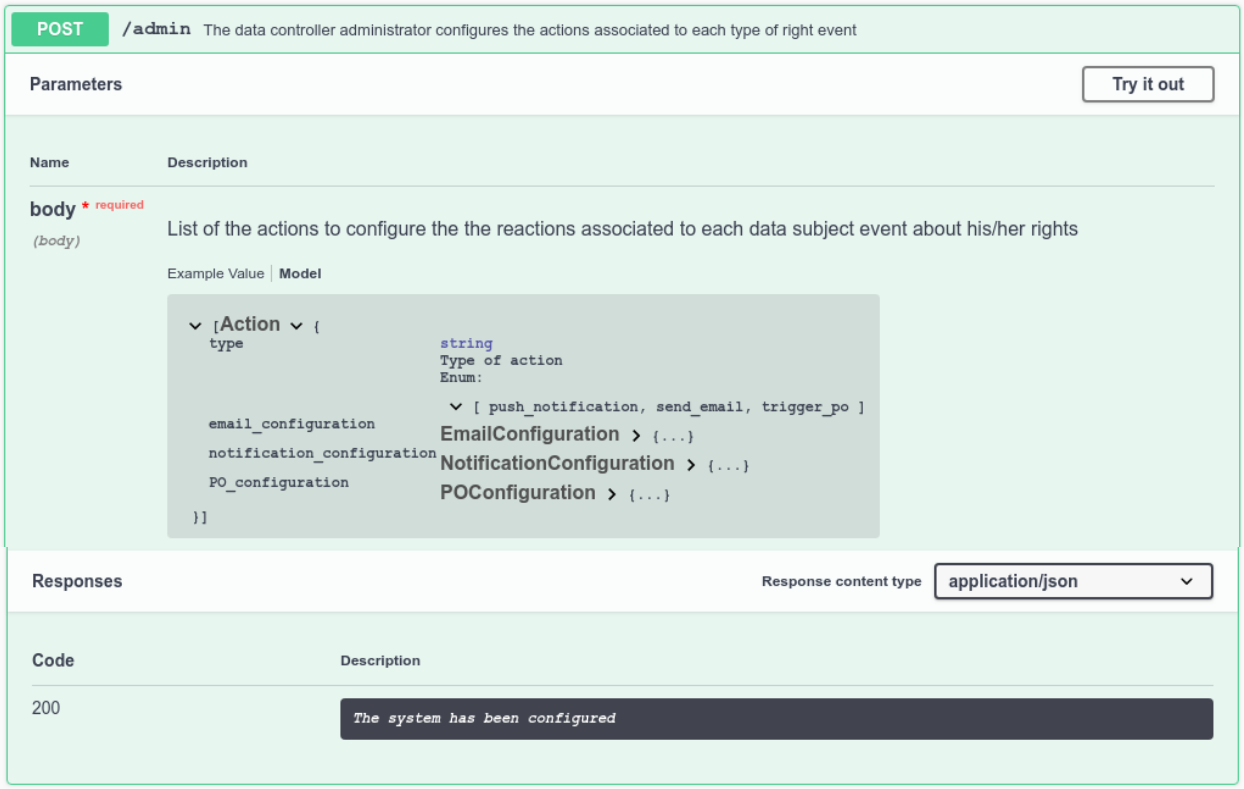
[ spl:AnyLocation, svl:ControllerServer, svl:EU, svl:EULike, svl:ThirdCountries, svl:OurServers, svl:ProcessorServers, svl:ThirdParty ]

**StorageDuration** string
Enum:

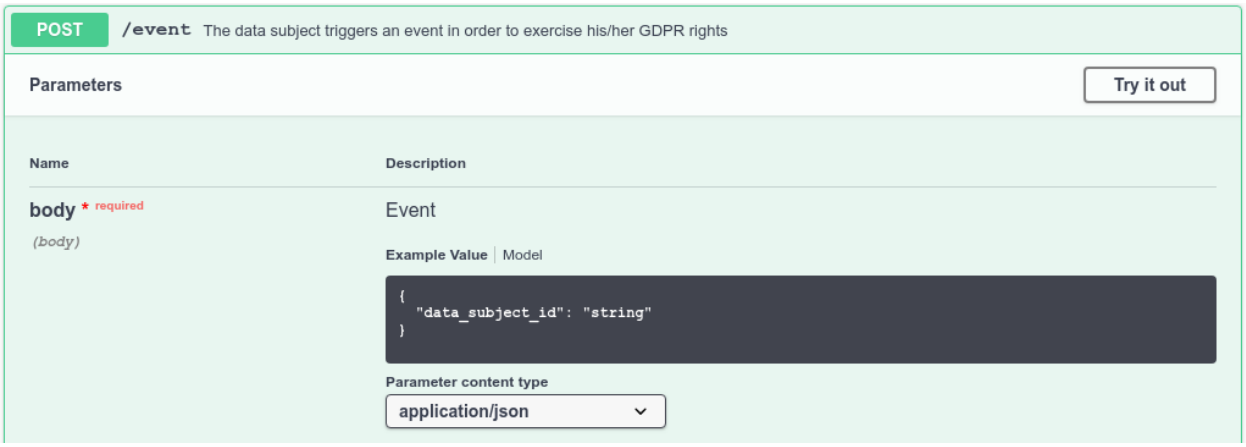
[ spl:AnyDuration, svdu:BusinessPractices, svdu:Indefinitely, svdu:LegalRequirement, svdu:StatedPurpose ]

### *7.4.4.2 Data Subject Rights Manager (DSRM) API*

**POST /admin**




**POST /event**



Responses | Response content type: application/json

| Code | Description |
|---|---|
| 200 | *Event trigger properly* |

## Models:

**Action** {
- type: string — Type of action — Enum: > Array [ 3 ]
- email_configuration: EmailConfiguration > {...}
- notification_configuration: NotificationConfiguration > {...}
- PO_configuration: POConfiguration > {...}

}

**EmailConfiguration** {
- address: string
- subject: string
- content: string

}

**NotificationConfiguration** {
- server: string
- topic: string

}

**POConfiguration** {
- file: string

}

**Event** {
- data_subject_id: string
- type: { *description:* *Type of event* }

}

# 8 Platform Deployment

The following section describes the deployment process of the PAPAYA platform, uploading and usage of privacy-preserving services provided by the platform (see Figure 47).

## 8.1 Platform initialization and platform dashboard deployment

For the PoC usage, the platform will be deployed on IBM's cloud service. All types of platform's users should have valid account on the IBM cloud. This can be done as follows:

1. Create free account on IBM cloud here (https://cloud.ibm.com/registration) and inform platform administrator when you done.
2. The platform administrator will invite you to the PAPAYA Kubernetes service and grant you appropriate privileges.

The platform administrator (IBM) will create an account of K8s service and allocate required resources for the Kubernetes cluster, such as number of nodes, RAM memory, CPUs, disk size etc. In addition, the platform administrator will allocate the following required external services:

1. CR. The platform administrator will create two namespaces, **papaya-client-side** and **papaya-server-side**. For each namespace the administrator will create a separate permission group. One group will be used for storing the server-side containers and will be exposed to administrators and service providers only. The second group is intended for storing the agent side containers and will be accessible for all three permission groups (administrators, service providers, and platform clients).
2. Cloud object Storage
3. DB – relational database
4. Provide permissions to the platform users
5. Elasticsearch account

All external services credentials will be saved using Kubernetes secrets mechanism. To obtain the required credentials (e.g. to use COS service), service providers will need to perform the following steps:

1. The platform administrator will create a secret.
2. The service provider will add the secret to the pods yaml file[33] .

The platform administrator will create the following permission groups:

1. Platform administrator
    a. Allocate resources on the Kubernetes cluster
    b. Allocate/Create external services
    c. Admin access to CR
2. Service providers
    a. Active access to the CR

[33] https://kubernetes.io/docs/tasks/inject-data-application/distribute-credentials-secure/

   b. If the service requires integration/communication with external service, appropriate accounts should be created or allocated.
3. Platform Clients
   a. Should be able to download only the agent side container from the CR

The platform administrator will deploy the Platform Dashboard using yaml deployment file. Once the dashboard is deployed, service providers and platform clients could use it to deploy/use the services.
To support auditing, the platform administrator will execute Loggingbeat as Deamonset on the Kubernetes cluster and connect it to the Elasticsearch instance.

## 8.2 Service upload and deployment

This section describes the steps needed in order to upload or use a service. For the PoC usage, we will use a Container Registry Provided by IBM Cloud. The steps are divided into two phases: (1) uploading phase; and (2) usage-execution phase.
The following steps are common for both phases:

1. Install the IBM Cloud CLI
2. Install the Docker CLI
3. Install the Container Registry plug-in.
   ibmcloud plugin install container-registry -r Bluemix
4. Log in to your IBM Cloud account.
   ibmcloud login -a https://cloud.ibm.com

The uploading phase will be done by the service provider user and it consists of the following steps:

1. Log your local Docker deamon into the IBM Cloud CR
   ibmcloud cr login
2. Create two images: server side and client-side agent.
   The server-side container will run on papaya's K8s cluster. The client-side agent will run on the client's side. Instructions on how to create the Docker image can be found in the following link:
   https://docs.docker.com/develop/develop-images/baseimages/
3. Upload each image to the appropriate namespace on the CR.
   for the server-side images use:
   docker tag hello-world de.icr.io/papaya-server-side/<my_repository>:<my_tag>
   docker push de.icr.io/papaya-server-side/<my_repository>:<my_tag>
   for the client-side agent images use:
   docker tag hello-world de.icr.io/papaya-client-side/<my_repository>:<my_tag>
   docker push de.icr.io/papaya-client-side/<my_repository>:<my_tag>

More details and CR commands can be found on the following link.

4. The service provider user (i.e. IBM, ORANGE, EUROCOM) will add a new service to the services catalog on the Platform dashboard (papaya.eu-de.containers.appdomain.cloud), while specifying all required fields and describing in details the service's flow and requirements.

The usage-execution flow will consist of the following steps:

1. The platform client will login to the Platform dashboard (papaya.eu-de.containers.appdomain.cloud).
2. The client will select the tab of services catalog and choose the service of interest.
3. The client will provide a configuration file if the service requires so.
4. The client will download the client-side agent image by using the following commands:

   *ibmcloud cr login*
   *docker pull <image name>*

   Where the <image name> is presented in the "services catalog" view on a platform dashboard under the "Agent Side Container" name and will be structured of the following
   *de.icr.io/papaya-client-side/<image>:<tag>*

5. After the services are deployed on the k8s cluster (by the platform dashboard application), the platform will present the url of the deployed service and provide the option to download the file (env file) that will contain SERVER_HOST_HTTP, SERVER_PORT_HTTP or SERVER_HOST_TCP, SERVER_PORT_TCP (accordingly to the communication protocol with the service) and other parameters that might be essential for communication with the client agent. The client administrator should be able to provide this file to the client agent, based on how the agent expects to receive these parameters. Possible ways of doing that are as follows:
   a. Provide the env file as env. variables for the client agent execution.

      *docker run with an --env-file <file name>*

   b. Read the env file by the client's application and provide it to the agent side within the initiation step. Meaning that the agent side will provide INIT API and will expect to receive all essential parameters in order to communicate with appropriate server side.

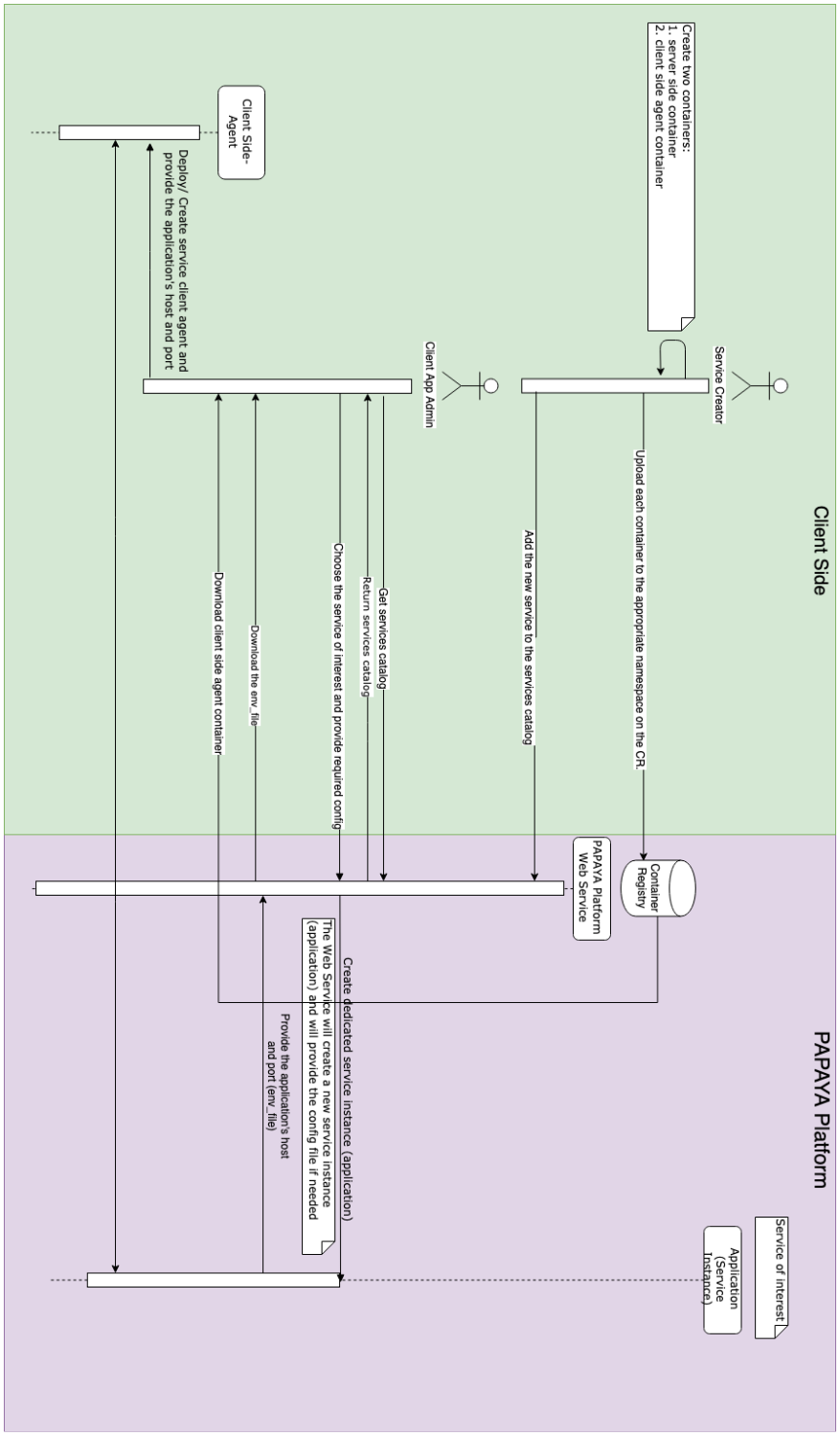



*Figure 47: Service upload and deployment diagram*

# 9 Platform Integration - Evaluation

In this section we describe briefly how we plan to evaluate the integrated platform. The goal is to verify that all required components could be deployed (either on the server-side or on the client-side), could run, could communicate with each other, and could provide required input/output. We want to stress here that evaluation of each service (e.g. in terms of accuracy, run time, etc.) is not in the scope of this document (it will be done in WP3). With this being mentioned, we plan to perform the following steps:

1. For each service described in section 4-7, create dummy service (both, the client-side and the server-side if required) - a pseudo service which only supports service APIs but doesn't have actual service functionality.
2. Deploy the service (both, the client-side and the server-side if required) as described in Section 8.
3. Run all the services and verify that expected functionality of each service (as described in D2.2 [1]) is supported.

# 10 Conclusions

In this deliverable, we presented the platform functional design, architecture and deployment. We described in detail the core platform services dealing with privacy-preserving computations as well as the services responsible to ensure data privacy, security, and transparency of all the processes while operating the platform. We explain how different services can be integrated into the platform in a way that they will be interoperable/compatible with each other and could work together in the integrated platform. We presented the design of platform dashboards that will provide UI, configuration, and visualization functionality. Finally, we described how all platform components could be deployed and how we plan to evaluate the integrated platform.

In next version of this document (D4.2 [21]), we plan to provide the full description of incomplete or not detailedly described services, evaluation results, and changes to the current design that maybe be required after the platform evaluation.

# 11 References

[1] S. Fischer-Hübner, B. Kane, J. S. Pettersson, T. Pulls, L. Iwaya, L. Fritsch, B. Rozenberg, R. Shmelkin, A. Palomares Perez, N. Ituarte Aranda and J. Carlos, *D2.2 - Requirements Specification,* 2019.

[2] B. Bozdemir, O. Ermis, M. Önen, M. Barham, M. Azraoui, S. Canard, B. Vialla, B. Rozenberg and R. Shmelkin, *D3.1 - Preliminary Design of Privacy Preserving Data Analytics,* 2019.

[3] B. Zvika, G. Craig and V. Vinod, "Fully Homomorphic Encryption without Bootstrapping".

[4] Z. Brakerski, "Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP," in *Annual Cryptology Conference*, 2012.

[5] J. Fan and F. Vercauteren, "Somewhat Practical Fully Homomorphic Encryption," *Cryptology ePrint Archive*, vol. Report 2012/144, 2012.

[6] J. H. Cheon, A. Kim, M. Kim and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *International Conference on the Theory and Application of Cryptology and Information Security*, Hong Kong, China, 2017.

[7] S. Canard, B. Vialla, B. Bozdemir, O. Ermis, M. Önen, M. Barham, M. Azraoui, B. Rozenberg and R. Shmelkin, *D3.3 - Complete Specification and Implementation of Privacy preserving Data Analytics,* 2020.

[8] C. M. Bishop, Pattern Recognition and Machine Learning (Information Science and Statistics), Springer, 2006.

[9] S. G. M. M. M. A. a. S. C. Eleonora Ciceri, *D2.1: Use Cases and Requirements. PAPAYA Deliverable D2.1,* 2019.

[10] Microsoft Research, Redmond, WA., *Microsoft SEAL,* : https://github.com/Microsoft/SEAL, 2018.

[11] S. Halevi, "HElib," [Online]. Available: https://github.com/homenc/HElib.

[12] wiki, "Garbled circuits," [Online]. Available: https://en.wikipedia.org/wiki/Garbled_circuit.

[13] "JustGarble," [Online]. Available: https://github.com/irdan/justGarble.

[14] C. r. a. O. S. University, "libOTe," [Online]. Available: https://github.com/osu-crypto/libOTe.

[15] wiki, "Oblivious transfer," [Online]. Available: https://en.wikipedia.org/wiki/Oblivious_transfer.

[16] N. Smart and F. Vercauteren, "fully Homomorphic SIMD Operations".

[17] chiraag, "Gazelle MPC," [Online]. Available: https://github.com/chiraag/gazelle_mpc/tree/master/src/lib.

[18] Y. Ishai, J. Kilian, K. Nissim and E. Petrank:, "Extending Oblivious Transfers Efficiently," in *CRYPTO*, 2003.

[19] F. Chollet, "Keras," [Online]. Available: https://keras.io/.

[20] J.-G. L. a. J. H. a. K.-Y. Whang, "Trajectory Clustering: A Partition-and-Group Framework," in *SIGMOD '07 Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, Beijing, China , 2007.

[21] B. Rozenberg, R. Shmelkin, B. Bozdemir, O. Ermis, M. Önen, M. Barham, M. Azraoui, S. Canard, B. Vialla and T. Pulls, *D4.2 - Progress report on platform implementation and PETs integration*, 2020.

[22] M. Abdalla, F. Bourse, A. De Caro and D. Poincheval, "Simple Functional Encryption Schemes for Inner Products," in *IACR International Workshop on Public Key Cryptography*, 2015.

[23] M. Abdalla, D. Catalano, D. Fiore, R. Gay and B. Ursu, "Multi-Input Functional Encryption for Inner Products: Function-Hiding Realizations and Constructions Without Pairings," in *Annual International Cryptology Conference*, 2018.

[24] S. Agrawal, B. Libert and D. Stehlé, "Fully Secure Functional Encryption for Inner Products, from Standard Assumptions," in *Annual International Cryptology Conference*, 2016.

[25] J. Chotard, E. D. Sans, R. Gay, D. H. Phan and D. Pointcheval, "Decentralized Multi-Client Functional Encryption for Inner Product," in *International Conference on the Theory and Application of Cryptology and Information Security*, 2018.

[26] E. D. Sans, R. Gay and D. Pointcheval, "Reading in the Dark: Classifying Encrypted Digits with Functional Encryption," IACR Cryptology ePrint Archive, 2018.

[27] C. E. Z. Baltico, D. Catalano, D. Fiore and R. Gay, "Practical Functional Encryption for Quadratic Functions with Applications to Predicate Encryption," in *Annual International Cryptology Conference*, 2017.