SHAPES

Smart and **H**ealthy **A**geing through **P**eople **E**ngaging in supporting **S**ystems

D4.2 SHAPES TP Development Tools and Capabilities Toolkit

Project Title	Smart and Healthy Ageing through People Engaging in Supportive Systems
Acronym	SHAPES
Grant Number	857159
Type of instrument	Innovation Action
Topic	DT-TDS-01-2019
Starting date	01/11/2019
Duration	48

Work package	WP4 – SHAPES Technological Platform
Lead author	Eleni ZAROGIANNI (ICOM) and Ilia PIETRI (ICOM)
Contributors	ICOM: Artur KRUKOWSKI EDGE : Marco Manso (EDGE), Jose Pires, Barbara Guerra GNO: Fotis Gonidis & Alexander Berler FINT: George Bogdos & Anargyros Sideris HMU: Yannis Nikoloudakis TREE: Tatiana Silva VICOM: Luis Unzueta, Manex Serras, Gorka Epelde, Jordi Torres, Naiara Muro, Jon Kerexata, Garazi Artola, Gorka Epelde & Eduardo Carrasco
Version	1.0 submission for internal peer review
Due date	30/04/2022 (M30)
Submission date	30/04/2022 (M30)
Dissemination Level	**PU** Public dissemination

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857159

Revision History

Rev. #	Date	Editor	Comments
0.1	2/03/2022	A. Krukowski (ICOM)	ToC
0.2	8/03/2022	E. Zarogianni (ICOM), I. Pietri (ICOM)	Updated ToC, Section 1 and 2
0.3	22/03/2022	E. Zarogianni (ICOM)	Section, 3.1.2 Sections 4.1 and 4.2, Section 5.
0.31	25/03/2022	Y. Nikoloudakis (HMU)	Section 3.1, 3.7 and 4.1
0.32	28/03/2022	F. Gonidis (GNO)	Section 3.5 and 4.4
0.33	30/03/2022	Tatiana Silva (TREE)	Section 3.6
0.34	30/03/2022	A. Sideris (FINT)	Section 3.3 and 3.4
0.4	30/03/2022	E. Zarogianni, I. Pietri, A. Krukowski (ICOM)	Section 3.2 and Section 4.2
0.5	1/04/2022	M. Serras (VICOM), T. Silva (TREE)	Section 3.8 Section 4.3
0.6	6/04/2022	I. Pietri (ICOM)	Sections 3.2, 4.1 and 4.2
0.7	6/04/2022	F. Gonidis (GNO)	Section 3.5.4
0.8	11/04/2022	V. Stamatiadis	First Review
0.9	29/04/2022	E. Zarogianni, I. Pietri, A. Krukowski (ICOM)	Post-PR corrections, incl. add-ons from partners, e.g. EDGE, TREE etc.
1.0	30/04/2022	E. Zarogianni (ICOM)	Final version submitted the coordinator and uploaded to EC

Keywords

Technological Platform, Design, Architecture, Implementation, Development Tools, Source Code.

Disclaimer

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857159.

Table of Contents

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857159.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857159.

List of Figures

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857159.

Table of Acronyms and Abbreviations

Acronym	Description
AIV	Assembly, Integration and Verifications
API	Application Programming Interface
App	Application
ASAPA	Authentication, Security and Privacy Assurance
CO	Consortium only dissemination level
PU	Public Dissemination level
DoA	Description of the Action
EC	European Commission
FHIR	Fast Healthcare Interoperability Resources
ICD	Interface Control Document
GDPR	General Data Protection Regulation
JSON	JavaScript Object Notation
MedSyn	MedicalSyn GmbH
OMN	Omnitor AB
PM	Person Month
QA	Quality Assurance
RAMS	Reliability, Availability, Maintainability, Safety of Means & People
RAP	Resource Access Proxy
REST	REpresentational State Transfer
RIA	Research and Innovation Action
SciFY	Science for You
STC	Scientific Technical Coordinator
TP	Technological Platform
TRD	Technical Requirements Document
UCLM	Universidad de Castilla - La Mancha
URL	Uniform Resource Locator
VICOM	Vicomtech
WP	Work Package

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857159.

Executive Summary

The purpose of this deliverable is to report on the work that has been performed in all the development tasks in Work Package 4 of the SHAPES project with respect to providing source-code and libraries such that 3rd-party developers would be able to take advantage of the SHAPES Technological Platform and be able to develop their value-added applications and services. Hence, the codes offered on GitHub repository of SHAPES as well as this report are intended for public dissemination.

Specifically, the sources codes and reference guides constituting this deliverable have been sources from the following tasks of WP4:

- **Task 4.3** Implementation of the Mediation Framework and Interoperability Services
  - SymbIoTe SDK and API
- **Task 4.4** Implementation & Deployment of the Secure Cloud and Big Data Platform
  - Big data platform API
- **Task 4.5** Human Interaction and Visual Mapping
  - voice, NLP and video features API
- **Task 4.6** SHAPES Authentication, Security and Privacy Assurance
  - authentication API
- **Task 4.7** SHAPES Gateway Reference Implementation
  - Gateway reference binary

The above mentioned source codes, libraries and SDKs are available on SHAPES GitHub (https://github.com/SHAPES-H2020) while this accompanying report provides the reference guide for those codes including API to platforms and services accessible remotely from each provider.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857159.

1 Introduction

Deliverable D4.2 “SHAPES TP Development Tools and Capabilities Toolkit” contains source codes, SDKs, libraries and APIs developed by all partners in WP4 and constitute public software support for 3rd-party developers and service providers wishing to take advantage of the SHAPES interoperability framework and supported Digital Solutions through the SHAPES Technological Platform. Since the actual software has been uploaded to a dedicated GitHub project (https://github.com/SHAPES-H2020), this report hence offers a reference guide to the list of repositories and code provided therein with reference guide how to use each of those.

1.1 Partners Involved in Relevant Tasks in WP4

The main task producing this deliverable was Task 4.8 (Integration and Testing of SHAPES TP) that focussed on the integration and validation of the integrated SHAPES technological platform.

A complete list of consortium partners contributing to deliverable D4.2 is as follows:

Table 3 – Deliverable Contributors.

ID	Short Name	Role
8	EDGE	Provider of the Front-end App
12	FINT	Leader of Task 4. Provider of FINoT IoT platform Provider of SHAPES Gateway
13	GNO	Leader of Task 4. Provider of FHIR interoperability framework
15	ICOM	Leader for deliverable D4.2 SHAPES Technical Manager Integration manager for SHAPES-TP Provider of symbIoTe and its API/SDK/plug-ins Provider of SHAPES messaging server
16	KOM	Provider of robot accessibility SDK
27	HMU	Leader of Task 4. Provider of ASAPA authentication framework
28	TREE	Leader of WP5 and Task 4. Provider of Data Lakehouse repository Provider of the Analytics Engine
35	VICOM	Leader of Task 4.5 Provider of o voice, NLP and video features API

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 857159.

1.2 Field of Application

This document is applicable to the remaining work within WP4 aiming at testing the integration of the core components into SHAPES TP, as well as the integration with the digital solutions that will be implemented in WP5. The various modules of the SHAPES TP will be integrated and tested in laboratory environment from a functional perspective. The integrated prototype will be ready to be deployed for the pilots and tested by the end users in accordance to the scenario uses cases.

1.3 Structure and Scope of the Document

The deliverable D4.4 "Integration Testing Results (preliminary version)" reports on the mid-phases of work produced in Task 4.8. "Integration and Testing of SHAPES Technological Platform" with respect to planned actions related to integration and deployments of the SHAPES platform for use case evaluations. The structure of the document is as follows:

Section 1: *Introduction and Methodologies*

Describes the methodologies for the integration activities.

Section 2: *SHAPES TP Prototype Integration*

Briefly describes the SHAPES Architecture, the integration approach followed for testing the SHAPES integrated prototype and the SHAPES repository structure, under which source codes are held

Section 3: *Project Build and Deployment of SHAPES Prototype Components*

Describes the structure of the developed component's software, the build and deployment requirements and the functionalities and dependencies of the individual components.

Section 4: *Communication Diagrams.*

Describes the flow of information between SHAPES component.

Section 5: Results and Conclusions.

1.4 Relation to other work in the project

This deliverable is based on results from:

- **D4.1** "*SHAPES TP Requirements and Architecture*" (due M18)
- **D4.3** "*Integration Plan and Test Cases*" (due M24)
- **D4.6** "*SHAPES Interoperability Reference Testing Environment*" (due M18)
- **D6.1** "*SHAPES Pan-European Pilot Campaign Plan*" (due M6)
- **D8.4** "*Ethics Framework for Shapes solution*" (due M6)

The outcomes from D4.2 feed to:

- **WP6**: "*SHAPES Pan-European Pilot Campaign*"
- **WP7**: "*Market Shaping, Scale-up Business Models and Socio-Economic Impact*"

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857159.

2 SHAPES TP Prototype Integration

2.1 SHAPES Architecture

The SHAPES Technological Platform (TP) brings a combination of devices, software, and accessible modes of interacting within the living environment that can adapt to the needs and priorities of older individuals, including those facing permanent or temporary reduced functionality and capabilities.

A number of established Digital Solutions (DS) that comprise the SHAPES ecosystem are expected to interconnect and integrate with the SHAPES core Technological Platform (TP), which is depicted in the lilac-hued area of the Figure 1 below.

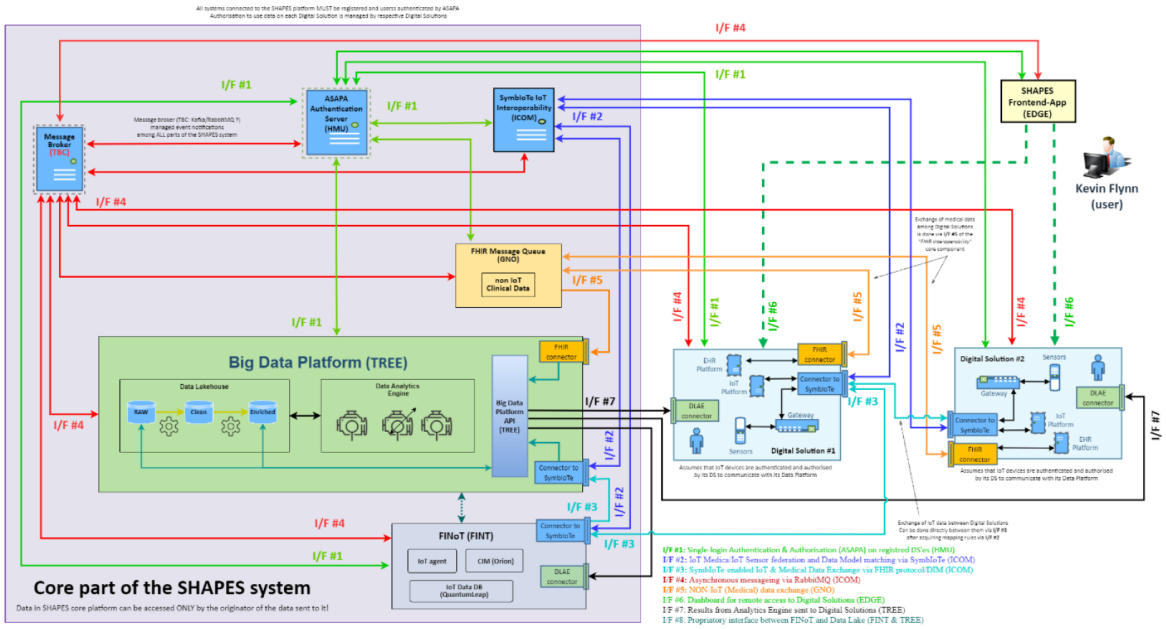

Figure 1- SHAPES TP architecture.

The SHAPES TP comprises of the following core components:

- *symbIoTe* IoT Interoperability Platform from ICOM is a mediation framework that facilitates the exchange of IoT Data between Digital Solutions and Platforms.
- *FINoT* IoT Data Management Platform from FINT is a FIWARE-based IoT framework, used to interconnect sensors, actuators and loggers. It acts as a central point for gathering IoT data, before these are fed to the Big Data Platform.
- *ASaPA* Single Sign-on Authentication engine from HMU offers authentication and authorization framework. Every user, digital solution, platform etc. is required to first register to the ASAPA component to get an authorization token in order to be able to interact within the SHAPES ecosystem.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857159.

- *Gateway* from FINT facilitates the interconnection of the edge IoT devices with the SHAPES Core cloud platform, enabling as such the accommodation of the IoT collected data to the FINoT IoT platform (part of the SHAPES core).
- *FHIR* Medical Data Interoperability from GNO. The FHIR medical interoperability component facilitates the interoperability and communication among digital solutions that exchange medical-related information with each other and/or other SHAPES core components. Its main component is the Message Queue (MQ), which allows the flow of medical-related resources among Digital Solutions or from Digital Solutions to the Data Lakehouse.
- *Big Data Platform* combining Data Lakehouse with the Analytics Engine from TREE. The Data Lakehouse along with the Analytics Engine form the so-called Big data Platform that allows Digital Solutions to send their data to the Data Lakehouse for advance processing, using an AI-based Analytics Engine. Results from the Data Analytics Engine (DAE) are sent back to relevant Digital Solutions.
- *SHAPES Front-end Application* from EDGE, brings a simple user interface providing a centralised access to the SHAPES Digital Solutions installed in the participant's mobile phone or tablet. It also provides a mechanism for the single authentication of the user in the device
- *Message Broker* from ICOM, enables the asynchronous notification mechanisms for all core components and interconnected Digital Solutions to be able to schedule exchange of information among them, without periodic checks.

2.2 Code Repository

Source code is available at GitHub: https://github.com/SHAPES-H2020, under which repositories per component have been created containing all the necessary information required for the project's build and deployment.

The structure of the GitHub project for SHAPES is as follows:

- *ASAPA*: ASAPA authentication system by HMU
- *Data Lakehouse*: Data Lakehouse by TREE
- *FHIR*: FHIR interoperability API by Gnomon
- *FINOT*: FINoT IoT platform by FINT
- *Front-end-App*: Front-end App support package by EDGE
- *Gateway*: Gateway integration support by FINT
- *Marketplace*: Marketplace integration support by HMU
- *Message-Queue*: SHAPES-TP message queue by ICOM
- *Robots*: Support package for Kompai robots
- *symbIoTe*: symbIoTe interoperability platform by ICOM
- VICOM: Human Interaction and Visual Mapping components by VICOM

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857159.

2.3 SHAPES Integration Approach

The project consortium uses the integration framework described in detail in D4.3 "Integration Plan and Test Cases" as the basis for implementation. During the implementation testing and lab deployment, each component (solution) is deployed and maintained at the premises of each relevant SHAPES partner. Communication between the different SHAPES TP core components is realised by providing RESTful APIs to expose their services, with JavaScript Object Notation (JSON) mainly being used for the format of the data. The relevant swagger documentation is provided by each component owner to describe their RESTful APIs. Asynchronous communication is, also, supported between the components through the deployment of a RabbitMQ server. Where possible, Docker is used for packaging the component's software and facilitating deployment of the components to other infrastructures. The respective container images are provided at Docker Hub[1], while SDKs, APIs or source code and respective configuration and deployment files such as Dockerfiles or Docker compose files are provided at each component's GitHub repo. Information and guidelines for the deployment of each SHAPES TP component in the integrated prototype is described in Section 3.

2.4 External Tools

The SHAPES TP has further dependencies on external tools, so that certain aspects of the platform prototype can be easily realised:

- RabbitMQ (version 3.6+): message queue server for the asynchronous communication between SHAPES TP core components. RabbitMQ's use is granted under a "Mozilla Public License". The RabbitMQ server is deployed at ICOM's premises to enable the exchange of notifications between SHAPES core components. A *Message-Queue* GitHub repo has been configured to add any accompanying material for the SHAPES-TP message queue component by ICOM.
- Amazon Web Services (AWS): used in the deployment of the Big Data Platform (Data Lakehouse and Analytics Platform) to provide scalable cloud computing services.
- Terraform: used to manage public cloud resources, such as AWS. Terraform Path specifically integrates the different projects, its documentation, and frameworks and infrastructure.
- MongoDB (version 3.6+): database used by symbIoTe components.
- Nginx (version 1.12+): for enabling access of platform components with the external world (i.e., applications, enablers, symbIoTe core).

[1] https://hub.docker.com/u/shapes2020

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857159.

3 Project Build and Deployment of SHAPES TP Prototype

This section provides information about the SHAPES TP core components integration and the deployment of the integrated SHAPES TP prototype. It provides information for the structure of each developed component's software, the build and deployment requirements and dependencies specific to them.

3.1 ASAPA Authentication and Authorisation (HMU)

3.1.1 Generic Information

URL of GitHub Repository	https://github.com/SHAPES-H2020/ASAPA
URL of Container Registry	https://hub.docker.com/r/shapes2020/asapa
URL of Deployment yaml file.	https://github.com/SHAPES-H2020/ASAPA/blob/main/docker-compose.yaml

3.1.2 Source Tree Information

The ASAPA component's source code contains the modules:

- /app_logger contains the functionality for logging
- /auth_module contains the authentication function and code
- /database_module contains the functions that manages the database connection and interaction
- /environment contains the variables for development and production environments
- /mq_client contains the functionality to register and interpret messages from a Rabbit MQ server, similar to consumer
- /policy_enforcer contains functionality for authorization of ASAPA
- /unit_tests contain the available unit tests
- /utils contains different function that are used in the project

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857159.

3.1.3 Tools and Project Dependencies

Tool	Version	Description
Docker-compose	3.9	Needed for deploying ASAPA Docker image
Docker-ce	stable	
git protocol	2.35.1	Needed for downloading the ASAPA

3.1.4 Build and Deployment Procedure

- git clone https://github.com/SHAPES-H2020/ASAPA
- cd /ASAPA
- Docker-compose up –d

The ports exposed are 8001 for the ASAPA service and 27018 for the database.

3.2 The symbIoTe orchestration middleware (ICOM)

3.2.1 Generic Information

URL of GitHub Repository	https://github.com/SHAPES-H2020/symbioteAPI
URL of Container Registry	https://hub.docker.com/r/shapes2020/symbiote-rap-websocket https://hub.docker.com/r/shapes2020/symbiote-api
URL of Deployment yaml file	https://github.com/SHAPES-H2020/symbiotecore https://github.com/SHAPES-H2020/symbiotecloud/tree/master/resources/docker/docker-compose

3.2.2 Source Tree Information

The symbIoTe project within the SHAPES repository has the following submodules:

- https://github.com/SHAPES-H2020/symbIoTeAPI,which includes the source code for the symbIoTeAPI service developed within the SHAPES project to offer a set of REST services designed to cover basic client requirements, without requiring applications or developers to download symbIoTe libraries.

- https://github.com/SHAPES-H2020/symbioteRAPNoticationAppExample, which includes the source code for an application example of subscribing to a resource to receive notifications, modified for the purposes of the SHAPES project.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857159.

- https://github.com/SHAPES-H2020/symbiotecore, which includes the configuration files and information required for the deployment of the symbIoTe core stack.
- https://github.com/SHAPES-H2020/symbiotecloud, which includes the configuration files and information required for the deployment of the symbIoTe cloud stack.
- https://github.com/SHAPES-H2020/symbiotesmartspace, which includes the source code for the smart space. Note that smart space is part of the SHAPES gateway and is described in detail later, in the relevant section for the SHAPES gateway (Section 3.4).

  Note that the symbIoTe software has only been extended for the purposes of the SHAPES project and original information is available at symbIoTe's GitHub repository[2].

3.2.3 Tools and Project Dependencies

Tool	Version	Description
Docker	18.03.x	symbIoTe software is provided in containerised form
Docker-compose	1.21.x	symbIoTe software is provided in containerised form
Docker-machine	0.14.x	symbIoTe software is provided in containerised form
MongoDB	3.6+	Containerised MongoDB database is used by symbIoTe Core/Cloud components for storing resource metadata
RabbitMQ	3.6+	Containerised RabbitMQ database is used by symbIoTe Core/Cloud components for internal communication between the subcomponents of symbIoTe.
NGINX	1.12+	NGINX is used for symbIoTe to enable access to components services from externally

3.2.4 Build and Deployment Procedure

As already mentioned, the symbIoTe software has been extended for the purposes of the SHAPES project and original instructions on how to deploy symbIoTe can be found at symbIoTe's GitHub repository[3]. This section provides the information and guidelines to deploy the services required for using symbIoTe within the SHAPES context.

[2] https://github.com/symbiote-h2020

[3] https://github.com/symbiote-h2020

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857159.

3.2.5 symbIoTe Core deployment

This section explains how to deploy the symbIoTe Core components stack using Docker directly in a VM running Linux. The instructions can also be found at the relevant wiki page. This is based on the online documentation for symbIoTe-core[4] available at the original symbIoTe's GitHub repository.

1. Preparation steps

1.1. Install prerequisites

- docker (18.03.x),
- docker-compose (1.21.x),
- bash,
- curl

1.2. Create deployment folder at file system

- Create a *symbiote-core* folder at the virtual machine's file system, e.g., `mkdir symbiote-core`
- Then inside the *symbiote-core* folders create the *configuration* folder:

```
cd symbiote-core
mkdir configuration
```

1.3. Create the Core AAM certificate required for your deployment

You need to create a PKCS12 key store containing a certificate and copy it inside the *configuration* folder; the steps to follow are described below. The key store will be used to self-initialize the *AuthenticationAuthorizationManager* (AAM) codes as Core AAM.

The key store is used by Core AAM micro service to authenticate third-party users and applications (i.e., users and applications that are not associated with any IoT platform). Core AAM micro service provides credentials required to access symbIoTe Services and also supports trust relationships between platforms, as it acts as the root certification authority.

The PKCS12 key store must have the following specifications:

- self-signed
- CA property enabled
- the following encryption parameters:
  - SIGNATURE_ALGORITHM=SHA256withECDSA
  - CURVE_NAME= prime256v1
  - KEY_PAIR_GEN_ALGORITHM=ECDSA
- Common Name (CN) value set according to AAMConstants.java field CORE_AAM_INSTANCE_ID value (currently equal to string "SymbIoTe_Core_AAM")
- certificate entry name: "*symbiote_core_aam*"

[4] https://github.com/symbiote-h2020/SymbioteCore

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857159.

The steps below need to be followed to create the PKCS12 key store core.p12:

- Create private key executing the following command:

```
openssl   ecparam -name prime256v1 -genkey   -out private-key.pem
```

- Create Certificate Signing Request (CSR) executing the following command:

```
openssl  req  -new  -sha256 -key private-key.pem -out CSR.csr
```

- Create the certification executing the following command:

```
openssl    x509 -req   -days 3600 -in   CSR.csr -signkey   private-key.pem -out
server.crt
```

Set ***CN = SymbIoTe_Core_AAM*** when is asked from the previous command.

- Creation of PKCS12 Keystore executing the following commands:

```
cat private-key.pem >server.pem
```

```
cat server.crt >> server.pem
```

```
openssl   pkcs12 -export  -in server.pem -out core.p12  -name paam  -noiter  -
nomaciter
```

Copy the generated *core.p12* keystore file inside the *configuration* folder.

1.4. Create necessary folders for your deployment

Copy from GitHub repo the following files in to *configuration* folder:

- *AuthenticationAuthorizationManager folder,*
- *CoreConfigProperties folder,*
- *CoreInterface folder,*
- *bootstrap.properties file,*
- *nginx-ngrok.conf,*
- *nginx-prod.conf,*
- *nginx.conf.*

Fill in all the fields marked with *FILL ME* in the *TODO* section of the following files:

- *CoreConfigProperties/application.properties*
- *AuthenticationAuthorizationManager/bootstrap.properties*
- *bootstrap.properties*

Enter the *CoreConfigProperties* folder, make any changes if necessary to the *.properties files (e.g. Mongodb, rabbitMQ credentials).

```
git          clone          https://github.com/symbiote-h2020/SymbioteCore.git
configuration/CloudConfigProperties
```

Enter the CoreConfigProperties folder:

```
cd CoreConfigProperties,
```

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857159.

to make any changes in the properties that are required e.g. the credentials and commit the changes:

```
git commit -am "SymbIoTe Core configuration"
```

```
cd ..
```

1.5. Create necessary docker volumes for your deployment

Change directory so that the current directory is the *configuration* folder.

Run from *configuration* folder the following command to create the necessary docker volume to hold the CoreconfigProperties:

```
docker container run --rm -v $PWD/CoreConfigProperties:/source -v {docker stack name}_symbiote-vol-config:/home/CoreConfigProperties -w /source alpine cp -r . /home/CoreConfigProperties/
```

As *{**docker stack name**}* use one that matches your project, thus a selected docker stack name such as *"**symbiote-core**".*

Copy in *configuration* folder the docker-compose files required (see GitHub):

- *docker-compose-swarm-core.yml*
- *docker-compose-prod-swarm-core.yml*

Uncomment and configure the proxy settings (JAVA_HTTP_PROXY, JAVA_HTTPS_PROXY, JAVA_SOCKS_PROXY, JAVA_NON_PROXY_HOSTS), in *docker-compose-swarm-core.yml* and *docker-compose-prod-swarm-core.yml* if you are behind proxy.

1.6. Obtain necessary certification files for your deployment

- Create **fullchain.pem** and **privkey.pem** certificate files as described in https://github.com/symbiote-h2020/SymbioteCloud/wiki/2.1-Configuration-of-NGINX#2111-obtaining-the-ssl-certificate or by some other provider.

Note that you can also use the certbot command to create a certificate manually with dns validation:

```
certbot --manual certonly --preferred-challenges dns -d <domain name> --email <admin email>
```

In that case you will be asked to deploy a DNS TXT record under a specific name. After the creation of the certificates, letsencrypt informs the user about the location of the new files. This is the location where the files have to be copied from. The target location has to be synchronized with the nginx configuration. Normally, it is the nginx-certificates folder as will be described next, where nginx is configured to find fullchain.pem and privkey.pem files.

- -Stay inside *configuration* folder and create the *nginx-certificates* folder.
- Copy the created **fullchain.pem** and **privkey.pem** certificate files to **nginx-certificates** folder (alternatively symbolic links can be used):

```
sudo cp /etc/letsencrypt/live/{your domain}/fullchain.pem nginx-certificates/
```

```
sudo cp /etc/letsencrypt/live/{your domain}/privkey.pem nginx-certificates/
```

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857159.

```
sudo chown -R {user}:{group} nginx-certificates
```

1.7. Start the deployment

- Run: `docker swarm init` if the node is not a swarm manager. Note this command is only done in the first deployment when setting the docker swarm and not when redeploying the stack. We use the swarm mode so that secrets are encrypted during transit and at rest. Docker secrets are only available to swarm services and not to standalone containers.
- Run `docker stack deploy -c docker-compose-swarm-core.yml -c docker-compose-prod-swarm-core.yml {docker stack name}` to deploy the service stack. The ***{docker stack name}*** is the name of the service stack. It is the same defined in $1.5 and in our case we use *"**symbiote-core**"*.
- Run `docker stack ls` to list the stack and check the number of services used.
- Run `docker image ls` to check that all images have been created. It may take a while to pull all the images from DockerHub for the first time.
- Run `docker service ls` to list the services and check their status. Wait until the actual number of tasks (replicas) for each service is not 0.
- Run `docker logs <service_name> -f` to get access to and follow the logs of a service. A service is ready when a message similar to 'Started <service_name> in 105.045 seconds (JVM running for 112.933)' appears in the logs of the service.
- Run `sudo service docker restart` to restart the docker service stack if needed and repeat the steps.
- Run `docker stack rm {docker stack name}` to stop the application and remove the service stack. Services, networks, and secrets associated with the stack will be removed. Note that *{docker stack name}* is equal to "*symbiote-core" string in SHAPES project.*
- You can run `docker swarm leave --force` to leave the swarm (note that this is not required for redeploying the stack but only if you stop using the node for docker swarm).

3.2.6 symbIoTe API deployment

The symbIoTe API can be deployed by using the following command:

```
docker compose up –d symbioteapi.yaml
```

The yaml file can be found at the symbIoTeAPI's GitHub repo. Swagger documentation for the endpoints exposed by the symbIoTe API to interact with symbIoTe in the context of SHAPES can be found at GitHub:

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857159.

3.2.6.1 symbIoTe Cloud deployment

The process of making an IoT platform symbIoTe-enabled consists of downloading and setting up ***symbIoTe Cloud*** components, integrating these components with your IoT platform, and registering your platform and resources to ***symbIoTe Core*** Services. The instructions can also be found at the relevant wiki page.

3.2.6.1.1 Prerequisites

Install the following on Linux:

- Docker (18.03.x)
- Docker-compose (1.21.x)
- Docker-machine (0.14.x)
- bash
- curl
- wget

3.2.6.1.2 Registration of platform owner

In this step it is necessary to register the platform owner (user) to symbIoTe Core through the symbIoTe Core Administration webpage. Visit the symbIoTe Core Administration web page at the following URL: https://symbiote-core.intracom-telecom.com/administration/, as shown in the figure below. Press the **Register** button.

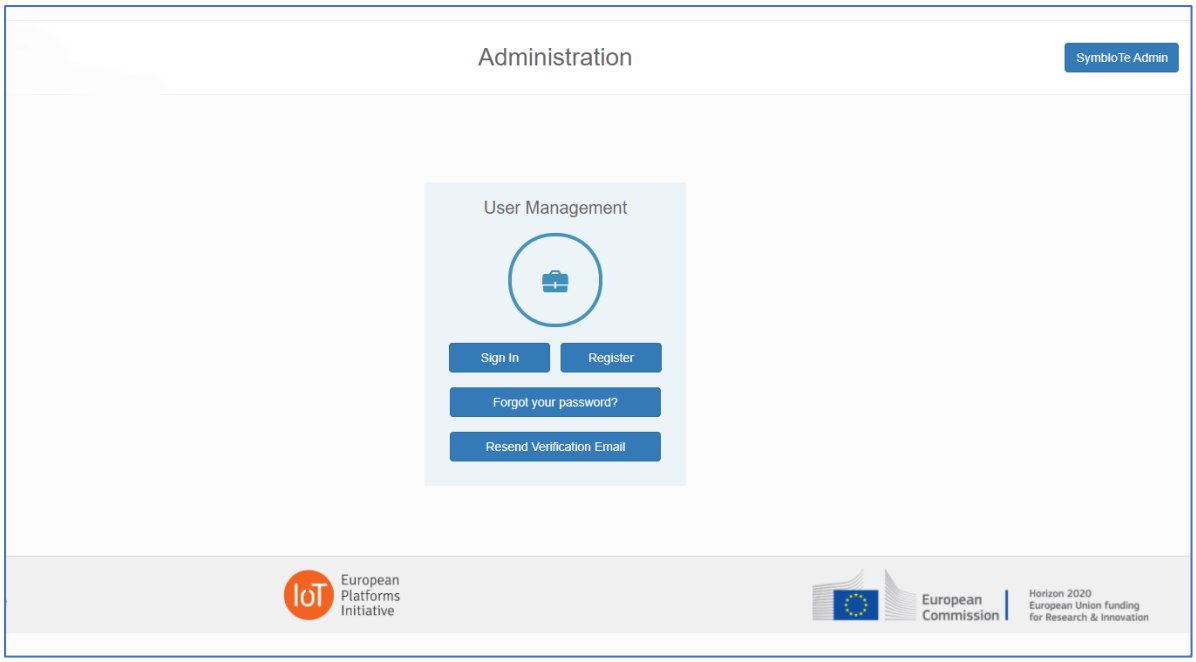

Figure 2 - Access to symbIoTe core administration GUI.

As shown in the following figure, during registration you must provide:

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857159.

- username
- password
- email
- user role (Service Owner in this case)

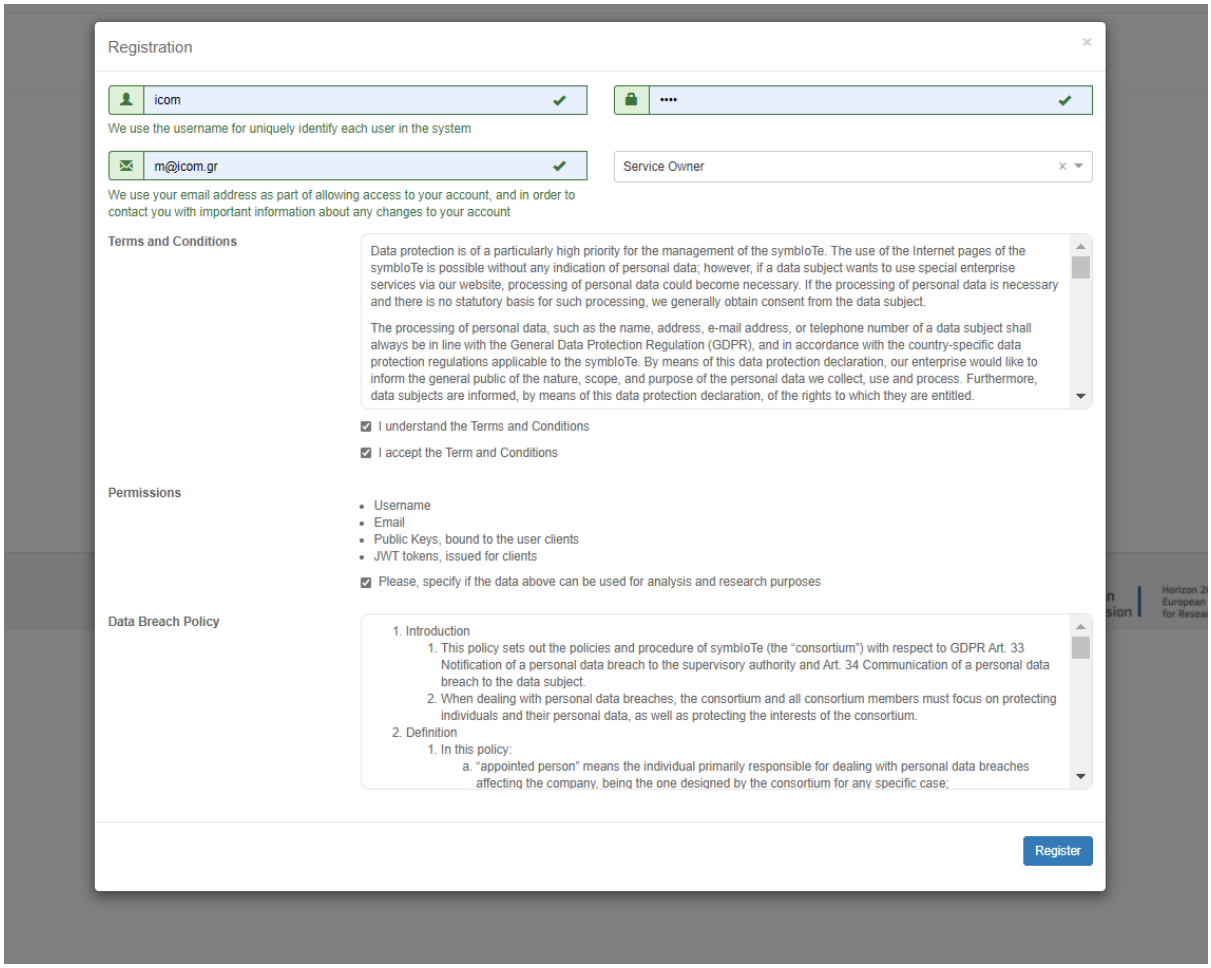

Figure 3 - Registration of platform owner to symbIoTe core.

3.2.6.1.3 Registration of platform's L2 Cloud services

Next, after successful registration you can log in (with the account details used for the registration of the platform owner) and register your platform (i.e., the symbIoTe Cloud Services platform). Click on the **Platform Details** panel and then click on **Register New Platform** button on the upper right corner.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857159.

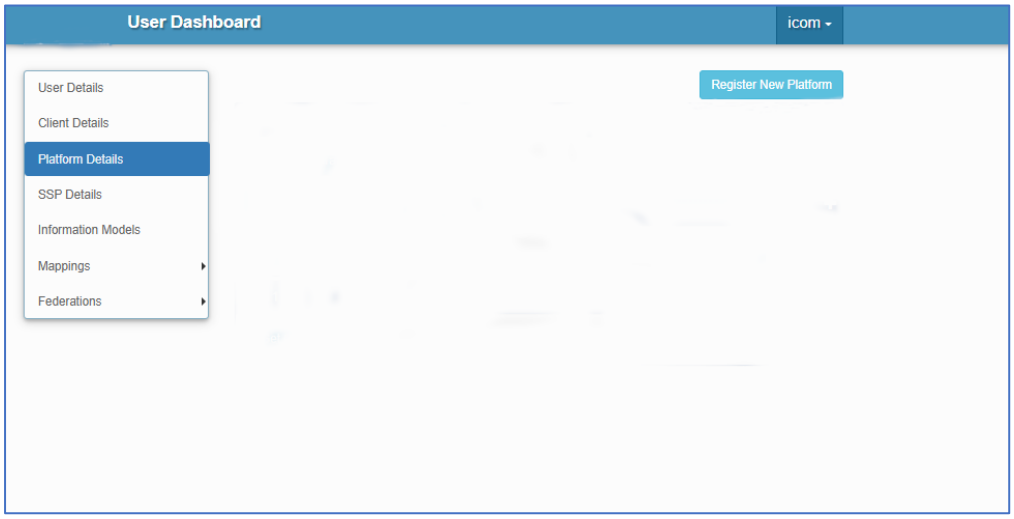

Figure 4 - Registration of the platform.

Then, the following details must be provided as shown in the figure below:

- **Preferable platform ID**
- **Platform Name**
- **Platform Description**
- **Interworking Services**: This is the valid URL to your Linux host where you will install L2 Cloud SHAPES services. It needs public IP address and DNS entry for that IP address. The alternative is to use ngrok tool which is good for experimentation but not for production.
- **Interworking Interface Information Model**: you can use BIM (Best practice Information Model) for out-of-the-box interoperability using the supported information model used in symbIoTe or PIM (Platform Information Model) to register a platform-specific information model that extends the supported information model. More details on creating an information model can be found at the wiki page.
- **Type** (i.e. Platform or Enabler): in this case use Platform.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857159.

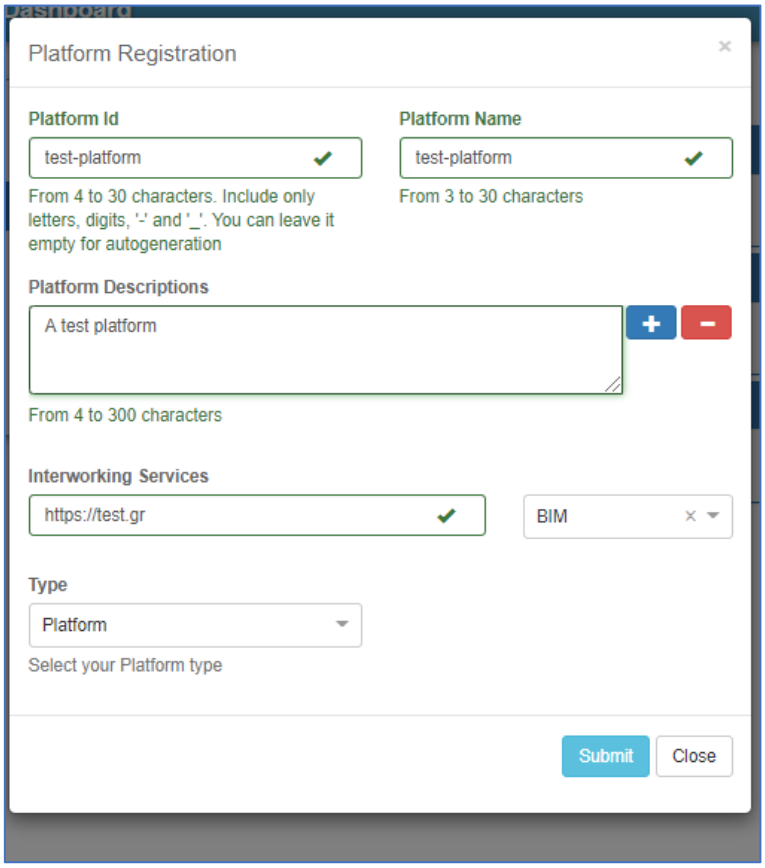

Figure 5 – Example of platform details in registration.

After completing this procedure, the IoT platform is now registered to the symbIoTe Core. It can be seen by selecting the **Platform Details** menu item.

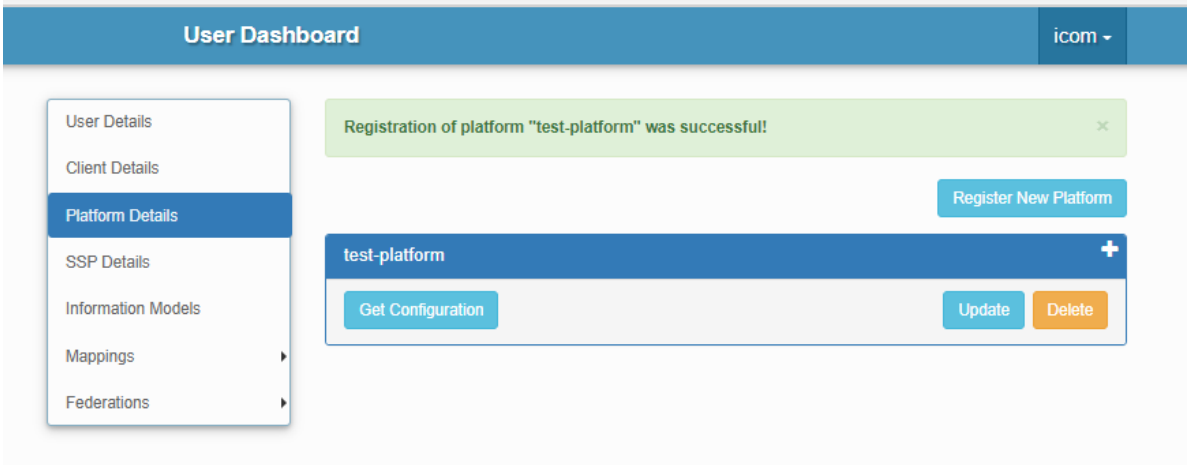

Figure 6 – Confirmation of Successful platform registration.

The panel of the newly registered platform and its details are available by clicking on its header or **+** sign.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857159.

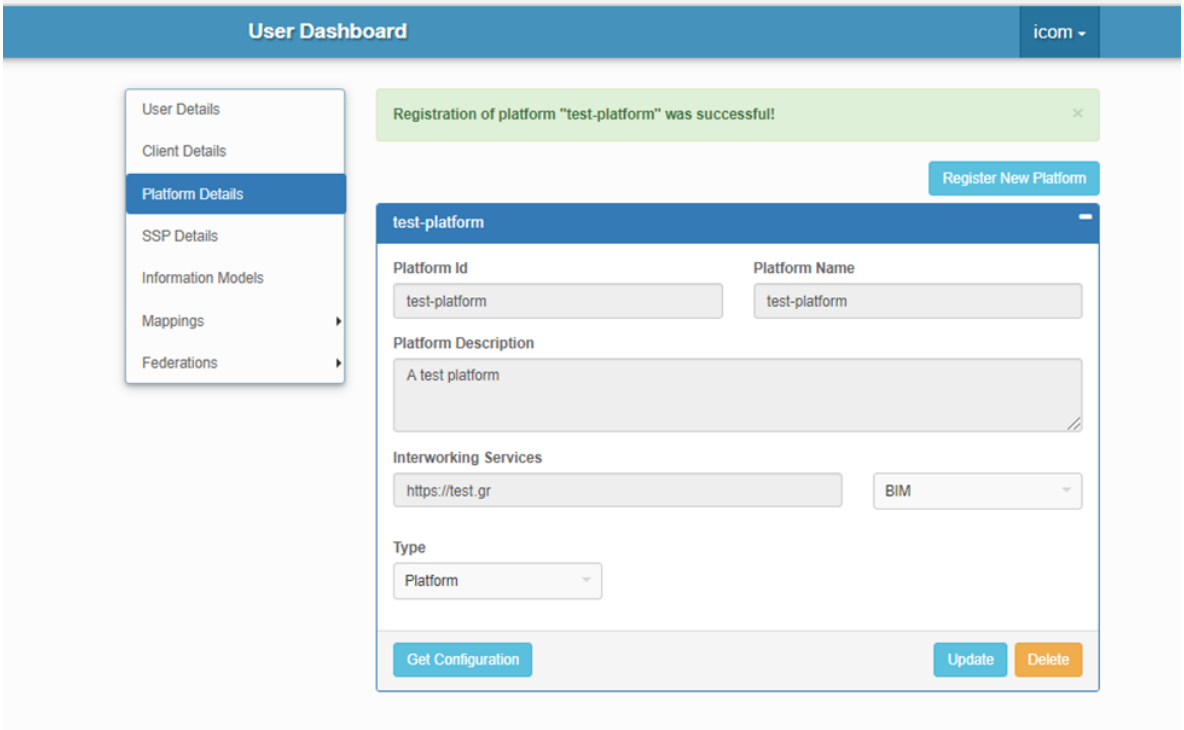

Figure 7 – Panel of the platform registered.

Download configuration files of registered platform's L2 Cloud services.

The next step is to download the necessary configuration for your deployment of the symbIoTe L2-compliance Cloud services. When opening the panel of the platform registered, as shown in the previous section, you can download the platform's configuration files by clicking on the **Get Configuration** button and entering some details as will be explained below. Hence, a .zip file will be downloaded that contains platform configuration properties which simplify the services configuration process.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857159.

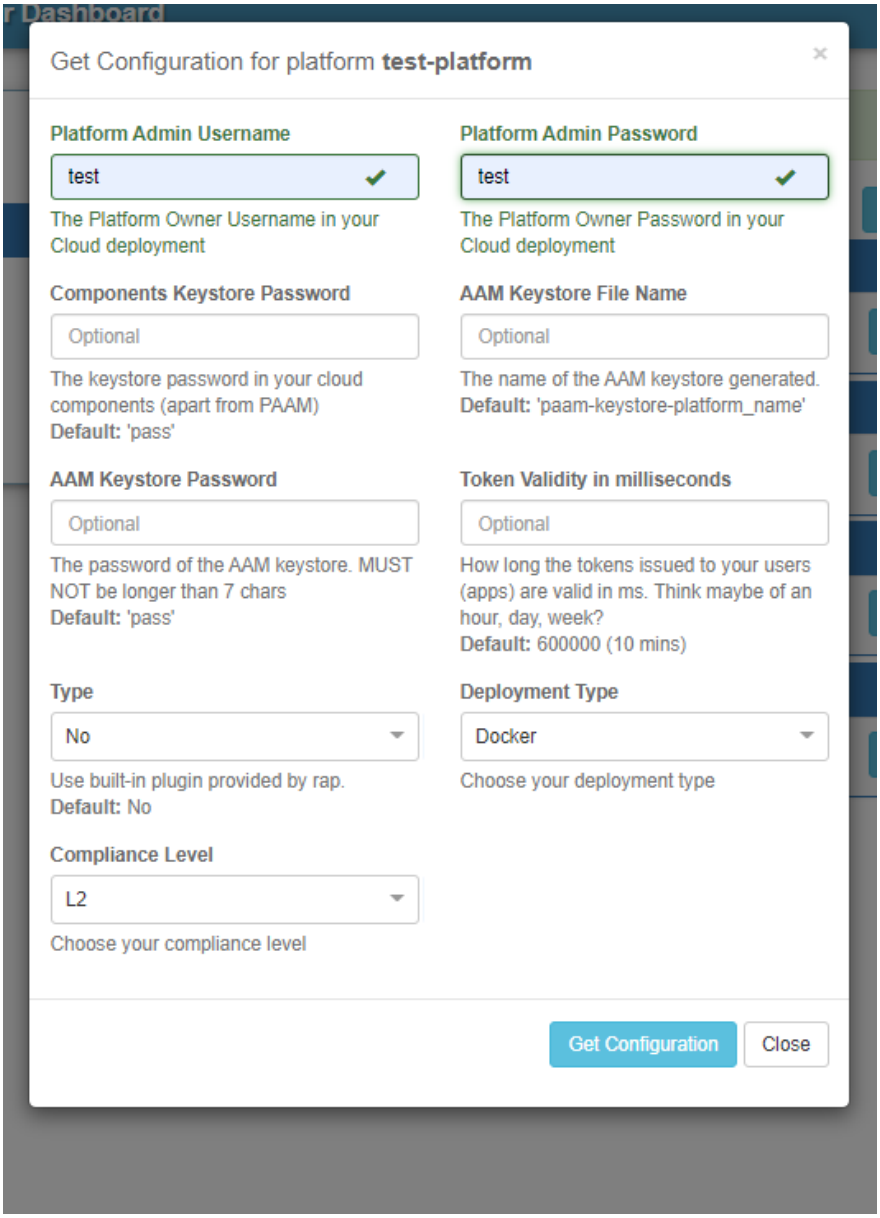

Figure 8- Platform configuration details.

When pressing the **Get Configuration** button, a configuration form is displayed (see the figure below).

Fill the mandatory fields:

- Platform Admin Username.
- Platform Admin Password.

And set

- Type: No.
- Deployment Type: Docker.
- Compliance Level: L2

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857159.

The optional fields can be left with their default values or you can set any other value that is necessary for the registered L2 cloud platform.

Press the **Get Configuration** button to download the **configuration.zip** file.

Create folder for your deployment and configure necessary property files

- Create a folder for your deployment, e.g. shapesL2-cloud:

  ```
  mkdir shapesL2-cloud
  ```

- Change to that directory: e.g.

  ```
  cd shapesL2-cloud
  ```

- Unzip in shapesL2-cloud folder the downloaded configuration.zip file.
- Enter the *CloudConfigProperties* directory.

  In *CloudConfigProperties/application.properties* file set URLs of the interworking interfaces. There are two kinds of interworking interfaces at symbIoTe Core:

  1. *coreInterface* serves northbound traffic coming from 3rd parties (e.g. applications searching for resources):

     symbIoTe.core.interface.url = https://symbiote-core.intracom-telecom.com

  2. *cloudCoreInterface* serves southbound traffic coming from IoT platforms (e.g. applications) running at the symbIoTe cloud:

     symbIoTe.core.cloud.interface.url = https://symbiote-core.intracom-telecom.com/cloudCoreInterface

- Enter the *shapesL2-cloud directory*.
- In *AuthenticationAuthorizationManager/cert.properties* set the address of the symbIoTe core AAM (available through the *coreInterface*):

  coreAAMAddress = https://symbiote-core.intracom-telecom.com/coreInterface

  Committing the changes

Inside the CloudConfigProperties directory execute the following commands:

```
rm -r .git
git init
git config user.email you@example.com
git config user.name "Your User Name"
git add .
git commit -m "Your platform's name"
```

Create a Docker volume to hold settings stored in CloudConfigProperties folder

- Enter to the shapesL2-cloud directory.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857159.

- Replace the ***{docker stack name}*** in the next Docker command with your selected Docker stack name (e.g., shapesL2_cloud).
- Inside the shapesL2-cloud directory execute the Docker volume command:

```
docker container run --rm -v $PWD/CloudConfigProperties:/source -v {docker stack name} _symbiote-vol-config:/home/CloudConfigProperties -w /source alpine cp -r . /home/CloudConfigProperties/
```

3.2.6.1.4 Configuring Docker compose files

Copy the following Docker compose files into the shapesL2-cloud directory (the Docker compose files can be found at the GitHub repo):

docker-compose-swarm-L2.yml

docker-compose-prod-swarm-L2.yml

In case your deployment (symbIoTe Cloud L2 services) runs behind a proxy then the **docker-compose-swarm-L2.yml** file must be configured to set the proxy settings. In case of proxy existence uncomment the lines beginning with:

- JAVA_HTTP_PROXY
- JAVA_HTTPS_PROXY
- JAVA_NON_PROXY_HOSTS

Set the content of lines (the respective environmental variables for the proxy settings) according to your proxy server setup.

3.2.6.1.5 Configuring NGINX micro-service

At the deployed symbIoTe Cloud L2 services docker stack, the NGINX microservice will also run. The NGINX microservice is configured using the **nginx.conf** and **nginx-prod.conf** files that are inside the shapesL2-cloud folder. The steps below need to be followed:

- Remove **nginx.conf**, **nginx-prod.conf** and **nginx-ngrok** stored in the **configuration.zip** file**.**
- Use **nginx.conf** and **nginx-prod.conf** files from provided documentation folder.
- Change the **server_name** setting in **nginx.conf** and **nginx-prod.conf** files: This is the platform name provided during registration such as test-platform**.**
- Create **fullchain.pem** and **privkey.pem** certificate files as described in https://github.com/symbIoTe-h2020/symbIoTeCloud/wiki/2.1-Configuration-of-NGINX or by some other provider.

  You can also use the certbot command to create a certificate manually with dns validation:

  ```
  certbot --manual certonly --preferred-challenges dns -d <domain name> --email <admin email>
  ```

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857159.

In that case you will be asked to deploy a DNS TXT record under a specific name. After the creation of the certificates, letsencrypt informs the user about the location of the new files.

This is the location where the files have to be copied from. The target location has to be synchronized with the nginx configuration. Normally, it is the nginx-certificates folder, which needs to be created as described next.

- In the **shapesL2-cloud** directory create the folder **nginx-certificates,** where nginx is configured to find **fullchain.pem** and **privkey.pem files,** executing the command**:**

```
mkdir nginx-certificates
```

Copy the created **fullchain.pem** and **privkey.pem** certificate files to **nginx-certificates** folder (alternatively symbolic links can be used):

```
sudo cp /etc/letsencrypt/live/{your domain}/fullchain.pem nginx-certificates/

sudo cp /etc/letsencrypt/live/{your domain}/privkey.pem nginx-certificates/

sudo chown -R {user}:{group} nginx-certificates
```

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857159.

3.2.6.1.6 Running Docker stack of the symbIoTe L2 Cloud services

Before deploying the L2 Cloud Docker stack (shapesL2-cloud) verify that the file structure under **shapesL2-cloud** folder is the same as shown in the next figure.

```
shapesL2-cloud
    docker-compose-prod-swarm-L2.yml
    docker-compose-swarm-L2.yml
    nginx-prod.conf
    nginx.conf

├──AuthenticationAuthorizationManager
        bootstrap.properties
        cert.properties

├──BarteringAndTrading
        bootstrap.properties

├──CloudConfigProperties
    │   application-cloud.properties
        application.properties
        auth-service.properties
        AuthenticationAuthorizationManager.properties
        BarteringAndTrading.properties
        EurekaService.properties
        FederationManager.properties
        InterworkingInterface.properties
        Monitoring.properties
        PlatformRegistry.properties
        RegistrationHandler.properties
        ResourceAccessProxy.properties
        SubscriptionManager.properties
        TrustManager.properties
        ZipkinService.properties

    └──.git
├──Eureka
        bootstrap.properties

├──FederationManager
        bootstrap.properties

├──Monitoring
        bootstrap.properties

├──nginx-certificates
        fullchain.pem
        privkey.pem

├──PlatformRegistry
        bootstrap.properties

├──RAPPluginStarter
        application.properties

├──RegistrationHandler
        bootstrap.properties

├──ResourceAccessProxy
        bootstrap.properties

├──SubscriptionManager
        bootstrap.properties

└──TrustManager
        bootstrap.properties
```

Figure 9- Structure of the deployment folder.

3.2.6.1.7 Running symbIoTe L2 Cloud services as a Docker stack

Follow the next steps to deploy symbIoTe Cloud L2 as a Docker stack.

- Enter to the **shapesL2-cloud** directory.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857159.

- Run **`docker swarm init`** if the node is not a swarm manager. We use the swarm mode so that secrets are encrypted during transit and at rest. Docker secrets are only available to swarm services and not to standalone containers.
- Run **`docker stack deploy -c docker-compose-swarm-L2.yml -c docker-compose-prod-swarm-L2.yml {docker stack name}`**

  `{docker stack name}` is the name of the service stack to be paused and is assigned in the step described in 1.2.5 paragraph e.g. shapesL2-cloud.
- Now you can run **`docker stack ls`** to list the stack and check the number of services used.
- After a few seconds run **`docker ps`** to check the list of micro services running in the ***`{docker stack name}`*** stack.
- Run **`docker image ls`** to check that all images have been created. It may take a while to pull all the images from Docker Hub for the first time. Docker service ls to list the services and check their status. Wait until the actual number of tasks (replicas) for each service is not 0.
- Run **`docker logs <container id> -f`** to get access to and follow the logs of a service. A component is ready when a message similar to 'Started in 105.045 seconds (JVM running for 112.933)' appears in the logs of the container.
- Run **`docker stack rm {docker stack name}`** to stop the application and remove the service stack. Services, networks, and secrets associated with the stack will be removed.
- You can run **`docker swarm leave --force`** to leave the swarm.

3.2.6.1.8 Creating the RAP plugin

Platform providers are required to implement the code needed to acquire resource data from their IoT platform. The RAP plugin acts as a micro service, interfacing to the RAP micro service (which is deployed in the L1/L2 symbIoTe Cloud) and the IoT platform itself to enable L1- or L2-compliant access to resources data within IoT platforms. Even though symbIoTe is entirely developed in Java, a standard communication mechanism is provided to all symbIoTe components (e.g., https and message queues) and therefore, the platform-specific RAP plugin is language independent and does not need necessarily to be developed in Java. Instructions on creating a RAP plugin can be found at the available online. Finally, notification mechanism can be enabled by implementing the relevant interfaces. An example of an application subscribing to a resource can be found at GitHub and relevant instructions at the page.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857159.

3.3 FINoT IoT platform (FINT)

3.3.1 Generic Information

URL of GitHub Repository	git@github.com:SHAPES-H2020/FINOT.git (provides the FINoT API documentation)
URL of Container Registry	NA (FINoT is a cloud base platform, containerized form is not available
URL of Deployment yaml file	NA (FINoT is deployed to FINT's cloud, no yaml is required)

3.3.2 Build and Deployment Procedure

Not applicable because FINoT is deployed and available via FINT's cloud.

3.4 SHAPES Gateway (FINT)

The SHAPES gateway consists of FINT's IoT gateway and the symbIoTe's Smart Space (SSP) middleware which enables the integration between smart objects (IoT gateway, smart devices and IoT platforms) forming a local IoT environment (smart space).

3.4.1 Generic Information

URL of GitHub Repository	git@github.com:SHAPES-H2020/Gateway.git
URL of Container Registry	N/A for IoT Gateway https://hub.docker.com/repository/docker/shapes2020/smart_space_middleware, for SSP
URL of Deployment yaml file	N/A (deployment via shell command or bash script), for IoT Gateway https://github.com/SHAPES-H2020/symbioteSmartSpace/tree/master/resources, for SSP

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857159.

3.4.2 Source Tree Information

The source tree information for symbIoTe's Smart Space[5] used as part of the SHAPES gateway in the context of SHAPES project is the following:

- \src\main\java\eu\h2020\symbiote\ssp\innkeeper which include the code for the Innkeeper component of the smart space that is responsible for the registration of the smart devices, IoT gateways and platforms.
- \src\main\java\eu\h2020\symbiote\ssp\localaam which includes the code for the Authorization Authentacation Manager (AAM) component of the smart space.
- \src\main\java\eu\h2020\symbiote\ssp\rap which includes the code for the Resource Access Proxy (RAP) component of the smart space that handles the access to the resources registered to the SSP.

3.4.3 Tools and Project Dependencies

Tool	Version	Description
Docker Engine	>= 20.10.11	The L3 symbi]IoTe integration middleware and the Shapes GW – Smartplug API are provided in a containerized form
Docker	18.03.x	symbIoTe software is provided in containerised form
Docker-compose	1.21.x	symbIoTe software is provided in containerised form
Docker-machine	0.14.x	symbIoTe software is provided in containerised form
MongoDB	3.6+	Containerised MongoDB database is used by symbIoTe SSP middleware

3.4.4 Build and Deployment Procedure

3.4.4.1 Smart space middleware

This section describes the procedure to deploy symbIoTe's smart space (SSP) middleware as part of the SHAPES gateway in order to integrate smart objects (IoT gateway, smart devices and IoT platforms) forming a local IoT environment (smart

[5] https://github.com/SHAPES-H2020/symbiotesmartspace

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857159.

space). The smart space middleware is based on the original symbIoTe software[6] adjusted and dockerised for the purposes of the SHAPES project. The steps to deploy the smart space middleware within the SHAPES project (also available at the wiki page) are the following:

3.4.4.1.1 Register user and configure symbIoTe Smart Space

The first step is to create a service owner user who is an ASAPA user in the symbIoTe Core Admin web page[7]. During registration, the following information needs to be provided:

- username
- password
- email
- user role (Service Owner in this case)

Figure 10 - Registration of the smart space admin at symbIoTe's GUI.

Afterwards, the user can log in to the administration GUI and register their symbIoTe Smart Space (SSP), by clicking on the *SSP Details* panel and then on *Register New SSP* button on the upper right corner.

[6] https://github.com/symbiote-h2020/SymbioteSmartSpace

[7] https://symbiote-core.intracom-telecom.com/administration/

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857159.

Then, the following details need to be provided:

- Preferable SSP id.
- SSP Name.
- External Address: a valid https URL for the address where the SSP is available from the Internet.
- Site Local Address: a valid https URL for the address where the SSP is available for clients residing in the same network.
- Choose if the site local address should be exposed.

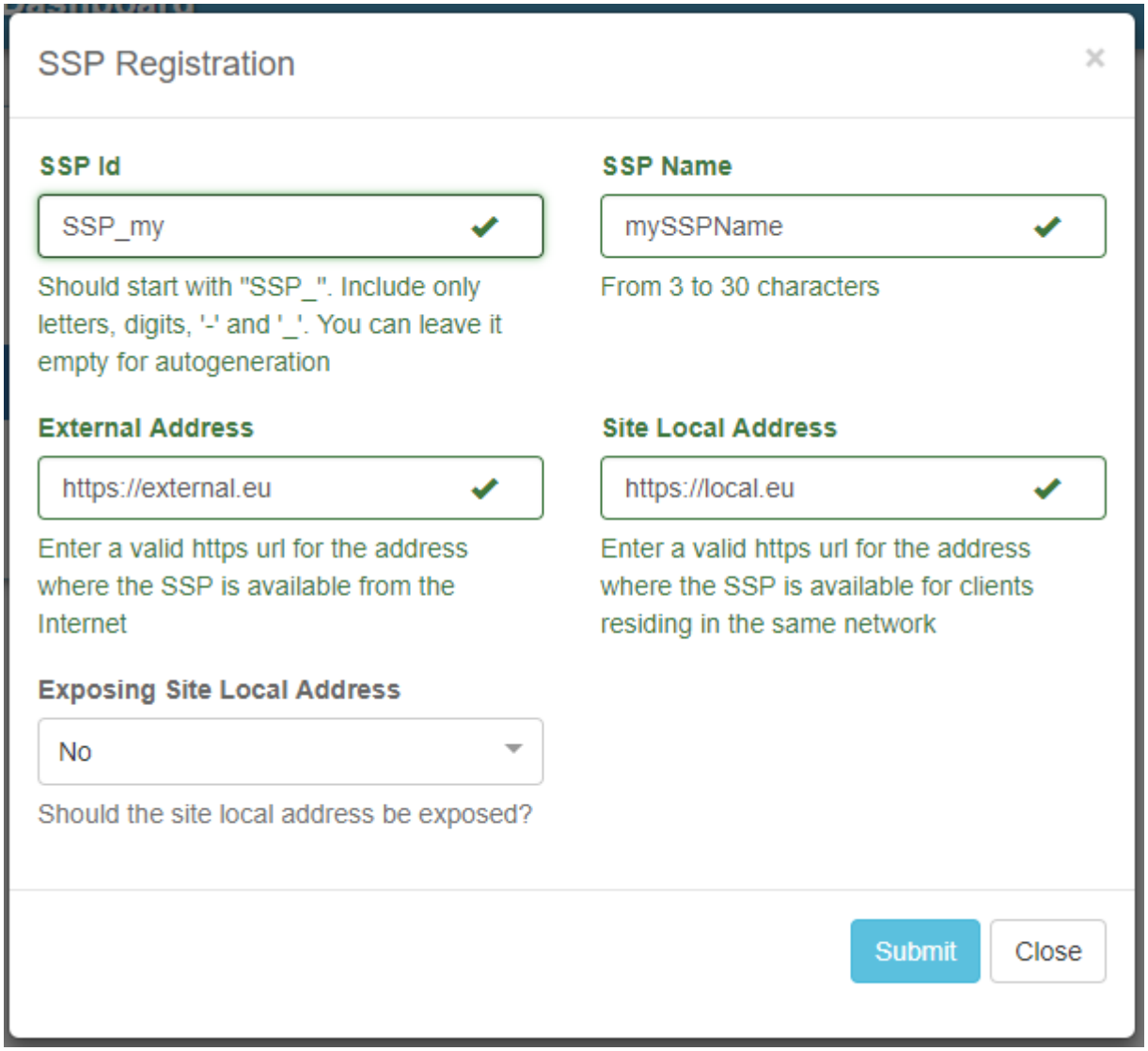

Figure 11 – Symbiote Smart space registration.

In this step the symbIoTe Smart Space (SSP) is registered to the symbIoTe Core services. The user can see the panel of the newly registered symbIoTe Smart Space (SSP) and check its details by clicking on its header.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857159.

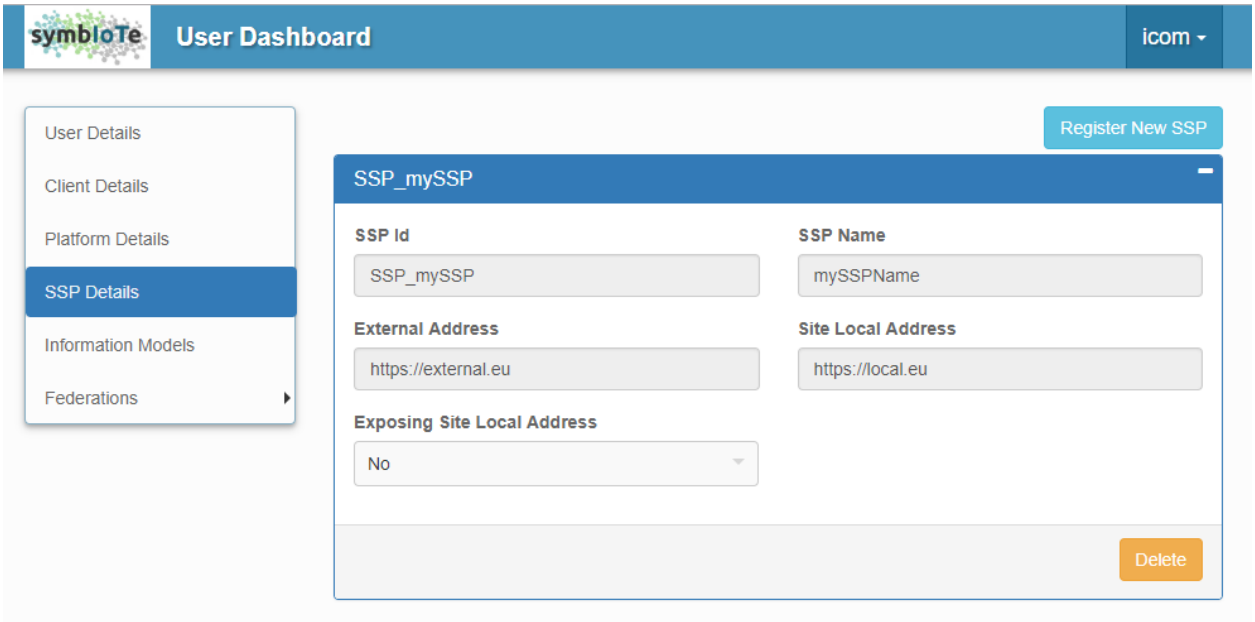

Figure 12 - Deletion of a smart space.

The registered SSP can be deleted by clicking the delete button on the bottom right corner of the SSP details.

3.4.4.1.2 Installing prerequisites in the virtual machine.

In the virtual machine where the smart space is to be deployed the following needs to be installed:

- Docker
- Docker-compose

The virtual machine running the Docker stack of symbIoTe's Smart Space must expose the HTTPS port 443 to internet with a public IP address. Also, the virtual machine must have the ability to access the symbIoTe Core Services through port 443.

Create the folder and the necessary subfolders for your deployment.

In virtual machine create the *SymbioteSmartSpace* folder:

```
mkdir SymbioteSmartSpace
```

Inside the SymbioteSmartSpace folder create the *SmartSpaceMiddleware* folder and copy the *application.properties* file.

Update the *application.properties* file with the correct configuration:

- Set to *ssp.id* the value assigned to SSP Id during the SSP registration phase described in the previous step.
- Set to *ssp.url* the public IP URL of your virtual machine.
- Update the *symbIoTe.core.interface.url*, *symbIoTe.cloud.interface.url* and *symbIoTe.rap.cram.url* with the correct symbIoTe Core Services URL.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857159.

Inside the *SymbioteSmartSpace* folder create an *AuthenticationAuthorizationManager* folder and copy the *bootstrap.properties* and *cert.properties* files.

Update the *bootstrap.properties* file with the correct configuration:

- Update the *symbIoTe.core.interface.url* with the correct symbIoTe Core Services URL.
- Update the *symbIoTe.interworking.interface.url* with the public IP of your virtual machine.

Update the *cert.properties* file with the correct configuration:

- Set to *serviceId* the value assigned to SSP Id during the SSP registration phase (described in the previous step).
- Set to *serviceOwnerUsername* and *serviceOwnerPassword* the credentials of the user registered through the administration web page to the symbIoTe Core services.
- Update the *coreAAMAddress* with the URL of symbIoTe's Core Services AAM.

Obtaining the certificate files.

To secure communication of symbIoTe Smart Space, an SSL certificate is needed[8] to obtain the certificate files *fullchain.pem* and *privkey.pem*:

Inside the *SymbioteSmartSpace* working folder create the *nginx-certificates* folder.

Copy to the *nginx-certificates* folder the certificate files *fullchain.pem* and *privkey.pem*:

```
$ sudo cp /etc/letsencrypt/live/{your domain}/fullchain.pem nginx-certificates/
$ sudo cp /etc/letsencrypt/live/{your domain}/privkey.pem nginx-certificates/
```

Configuration of NGINX with HTTPS.

Nginx microservice is used for redirecting external requests to symbIoTe's Smart Space Docker microservices (such as, from IoT platforms, Smart Devices, symbIoTe-enabled applications). Nginx needs to be configured so that it redirects correctly to the appropriate microservice endpoints. Copy the Nginx configuration file *nginx-prod.conf* inside *SymbioteSmartSpace* folder.

Configuration of docker compose files.

Inside the *SymbioteSmartSpace* folder, copy the *docker-compose-prod-swarm-L3.yml* and *docker-compose-swarm-L3.yml* files.

In case the virtual machine runs behind a proxy, update the proxy settings according to your proxy server. If no proxy is used, then comment the proxy settings parts

[8] https://github.com/symbiote-h2020/SymbioteCloud/wiki/2.1-Configuration-of-NGINX#2111-obtaining-the-ssl-certificate

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857159.

(JAVA_HTTP_PROXY, JAVA_HTTPS_PROXY, JAVA_SOCKS_PROXY and JAVA_NON_PROXY_HOSTS).

Run the docker stack.

To deploy the Symbiote Smart Space docker stack run:

- **`docker swarm init`**
- **`docker stack deploy -c docker-compose-swarm-L3.yml -c docker-compose-prod-swarm-L3.yml`** ***`nameOfYourStack`***

Note that you can select any preferable name for the stack to be deployed, *nameOfYourStack.*

To check that the docker stack is up and running execute the following command:

`docker ps` that lists the services of the stack.

To remove the stack execute the following command:

`docker stack rm nameOfYourStack.`

To ensure the deployment is working the following URLs can be checked from the browser:

- https://yourdomain/innkeeper/public_resources, where the correct response at the beginning of the deployment is an empty json array []. https://yourdomain/aam/get_available_aams, which returns the information about all available AAMs in the system from the symbIoTe Core Services.

After successful deployment of the SSP the registration of the resources can be followed (see wiki).

3.4.4.2 Gateway's symbIoTe Client (L3-compliant GW)

This section describes the deployment of the symbIoTe client installed at FINT's gateway (GW) to enable its interconnection to the smart space middleware (L3-compliance).

The following Dockerfile contains the configuration for building the Docker image of the GW's client.

```
FROM python:3
MAINTAINER Anargyros Sideris (FINT)

# ARG srcdir=dist/finot_symbiote_main
RUN apt install bash
ENV appdir=/usr/src/app
ENV appname=gw-l3-rap-plugin
RUN mkdir -p $appdir/$appname
```

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857159.

```
WORKDIR $appdir/$appname

COPY requirements.txt ./

RUN pip install --no-cache-dir -r requirements.txt

COPY l3mdw.py  .
COPY modules  modules

# CMD [ "/bin/bash", "-c", "ls -l" ]
CMD [ "python", "l3mdw.py"
```

The following are the contents of the middleware's deployment bash script; it is assumed that the Dockerfile is in the same directory with the deployment script. The deployment script is to be executed logged in the GW as a shell command with root privileges (e.g., bash deploy.sh).

```
#!/bin/bash
name=gw-l3-rap-plugin
cd ~/l3-mdw/
docker stop $name
docker rm $name
docker build -t $name .
docker run --network host --name $name -d $name
```

3.5 FHIR and eHealthPass Medical Interoperability (GNO)

3.5.1 Generic Information

URL of GitHub Repository	https://github.com/SHAPES-H2020/FHIR_MQ/tree/master
URL of Container Registry	N/A
URL of Deployment yaml file	N/A

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857159.

3.5.2 Source Tree Information

The FHIR MQ component is provided as a web service and is accessible by making requests to the API, as described in the previous deliverable D4.3 “Integration Plan and Test Cases”.

In GitHub, FHIR component is available as follows:

- *FHIR*: FHIR interoperability API by Gnomon.
- *FHIR_MQ*: Message queue for FHIR by Gnomon.

3.5.3 Build and Deployment Procedure

The FHIR MQ is offered as a web service. As such developers do not need to build and deploy the service on their premises but can consume directly it using the API described in the D4.3.

3.6 Data Lakehouse and Analytics Engine (TREE)

3.6.1 Generic Information

URL of GitHub Repository	https://github.com/orgs/SHAPES-H2020/repositories?q=Data Lakehouse_&type=all&language=&sort= Data lake integration: https://github.com/SHAPES-H2020/Data Lakehouse_integration.git API: https://github.com/SHAPES-H2020/Data Lakehouse_bigdata-api.git Data Lakehouse_ DS environment: https://github.com/SHAPES-H2020/Data Lakehouse_ds-env.git Data Lakehouse_FHIR integrator: https://github.com/SHAPES-H2020/Data Lakehouse_fhir-integrator.git Data Lakehouse_symbiote conector: https://github.com/SHAPES-H2020/Data Lakehouse_symbiote-connector.git Data Lakehouse_ASAPA authenticator: https://github.com/SHAPES-H2020/Data Lakehouse_asapa-authenticator.git
Terraform Path	*integrates the different projects, its documentation, and frameworks and infrastructure* https://github.com/SHAPES-H2020/Data Lakehouse_integration/tree/main/infr Data Analytics: https://github.com/SHAPES-H2020/Data Lakehouse_ds-env/tree/main/infr Internal API (entrance to TREE’s API): https://github.com/SHAPES-H2020/Data Lakehouse_fhir-integrator/tree/main/api_2level

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 857159.

	Integration with FHIR: https://github.com/SHAPES-H2020/Data Lakehouse_fhir-integrator/tree/main/api_1level

3.6.2 Source Tree Information

- *Data Lakehouse_integration*: **"integration" project:** *integrates the different projects, its documentation, and frameworks and infrastructure.*
- *Data Lakehouse*: **"data-lakehouse" project:** *core component that defines and orchestrates the data workflows.*
- *Data Lakehouse_ds-env*: **"ds-environment" project:** *infrastructure configuration modules for a data scientist team environment.*
- *Data Lakehouse_asapa-authenticator* **"asapa-authenticator" project:** *Data Lakehouse ASAPA integrated authenticator.*
- *Data Lakehouse_bigdata-api* **"bigdata-api" project:** outbound data and control interfaces.
- *Data Lakehouse_symbiote-connector*: **"symbiote-connector" project:** *Symbiote – Inbound API integration project.*
- *Data Lakehouse_fhir-integrator:* **"fhir-integrator" project**: *Phir – Inbound API integration project.*
- *Data Lakehouse_fhir-integrator:* **"fhir-integrator" project**: *Phir – Inbound API integration project*.

3.6.3 Tools and Project Dependencies

The list of tools to be deployed is still under investigation; however a preliminary list of the tools to be used are:

Tool	Version	Description
AWS		Under development
Terraform		Under development
Docker/K8s		Under development

3.7 SHAPES Front-end Application (EDGE)

3.7.1 Generic Information

URL of GitHub Repository	https://github.com/SHAPES-H2020/Front-end-App
URL of Container Registry	N/A

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857159.

URL of Deployment yaml file.	N/A

SHAPES Front-end Application (FA) is a new SHAPES development, proposed by EDGENEERING, with the clear purpose of facilitating the users' interaction with the different SHAPES Digital Solutions, namely those running on Apps either on smartphones or tablets (devices).

The SHAPES Front-end App uses Android Intent to start other installed Apps. When an intent is launched, it sends authentication data using Bundles.

The SHAPES Front-end App is integrated with the ASAPA (SHAPES Authentication mechanism) and several SHAPES digital solutions.

3.7.2 Source Tree Information

The FA source-code follows a structure common in Android Studio projects (https://developer.android.com/studio). As documented in Android Studio's project overview (https://developer.android.com/studio/projects) as follows.

Within each Android app module, files are shown in the following groups:

- **manifests:** Contains the *AndroidManifest.xml* file.
- **Java:** Contains the Java source code files, separated by package names, including JUnit test code.
- **res:** Contains all non-code resources, such as XML layouts, UI strings, and bitmap images, divided into corresponding sub-directories.

Within each module name, the following directories are used:

- build/ Contains build outputs.
- libs/ Contains private libraries.
- src/ Contains all code and resource files for the module

Project and module specific build configurations are defined in *build.gradle* files.

3.7.3 Tools and Project Dependencies

Tool	Version	Description
AndroidStudio	4.1 or above	https://developer.android.com/studio
Gradle	7.0.3 or above	Build automation tool for software development. Used for compilation, testing, deployment, and publishing.
Android version / API	Android 6.0 or above / API Level 23 or above	Android operating system and runtime system.
Android SDK Build Tools	31 or above	Software development kit including a debugger, libraries and an android device emulator.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857159.

Git tools	2.25 or above	Tool for downloading FA from the SHAPES repository.

3.7.4 Build and Deployment Procedure

- `git clone https://github.com/SHAPES-H2020/Front-end-App`
- `cd /Front-end-App`
- `./gradlew assemble` (in unix-based command line)

3.8 Marketplace (HMU)

3.8.1 Generic Information

URL of GitHub Repository	https://github.com/SHAPES-H2020/Marketplace
URL of Container Registry	https://hub.docker.com/r/shapes2020/marketplace-backend https://hub.docker.com/r/shapes2020/marketplace-frontend
URL of Deployment yaml file	https://github.com/SHAPES-H2020/Marketplace/blob/main/docker-compose.yaml

3.8.2 Tools and Project Dependencies

Tool	Version	Description
Docker-compose	3.9	Needed for deploying Marketplace Docker image
Docker-ce	stable	-
git protocol	2.35.1	Needed for downloading the Marketplace

3.8.3 Build and Deployment Procedure

In order to deploy the Marketplace, there are two main components to be deployed, the Front-end and the Back-end. Both components are compiled and deployed by utilizing the Docker-compose functionalities.

- Git clone https://github.com/SHAPES-H2020/Marketplace
- Cd Marketplace
- Docker-compose up -d

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857159.

3.9 Human Interaction and Visual Mapping (VICOM)

3.9.1 Generic Information

URL of GitHub Repository	https://github.com/SHAPES-H2020/VICOM

As the modules listed above, rather than being provided as a software with integration capabilities, are provided as standalone/pluggable solutions that can be employed to enrich the SHAPES ecosystem of DS.

3.9.2 Source Tree Information

The previously defined repository contains the information of the Human Interaction and Visual Mapping components of SHAPES. The main tools provided in this section are:

- Adilib and the Adilib-Skills
  - Reminders
  - Tutorials
  - Questionnaires
  - Agenda
- Emotion detection
- Oroface
- Facocog

The GitHub repository is under development and the following information will be ultimately added:

- URLs of the current deployments
- User Manuals and Documentation
- Demonstrators (if applicable)
- URLs / Documentations of the intercommunication APIs (if applicable)
- Other relevant information for the acquisition of these solutions

3.9.3 Tools and Project Dependencies

Not applicable OR to be provided in D4.6

3.9.4 Build and Deployment Procedure

To be provided in D4.6.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857159.

4 Communication Diagrams

This section describes the communication diagrams between the SHAPES core components regarding main functionalities of the SHAPES TP, as well as the exchange of messages required between SHAPES core components and digital solutions to be able to interact within the SHAPES ecosystem.

4.1 User Initialisation Workflow

This section describes the communication diagram for a DS to integrate their user registration procedure with the SHAPES authentication mechanism provided by the ASAPA component in order provide secure and authorised access to users. A user, DS or client application that wants to interact with the SHAPES ecosystem needs to register to the ASAPA component in order to be able to login to the SHAPES ecosystem and authenticate themselves. The authentication token is then needed to be presented for accessing and using any SHAPES service. The detailed user initialisation workflow is described in the figure below.

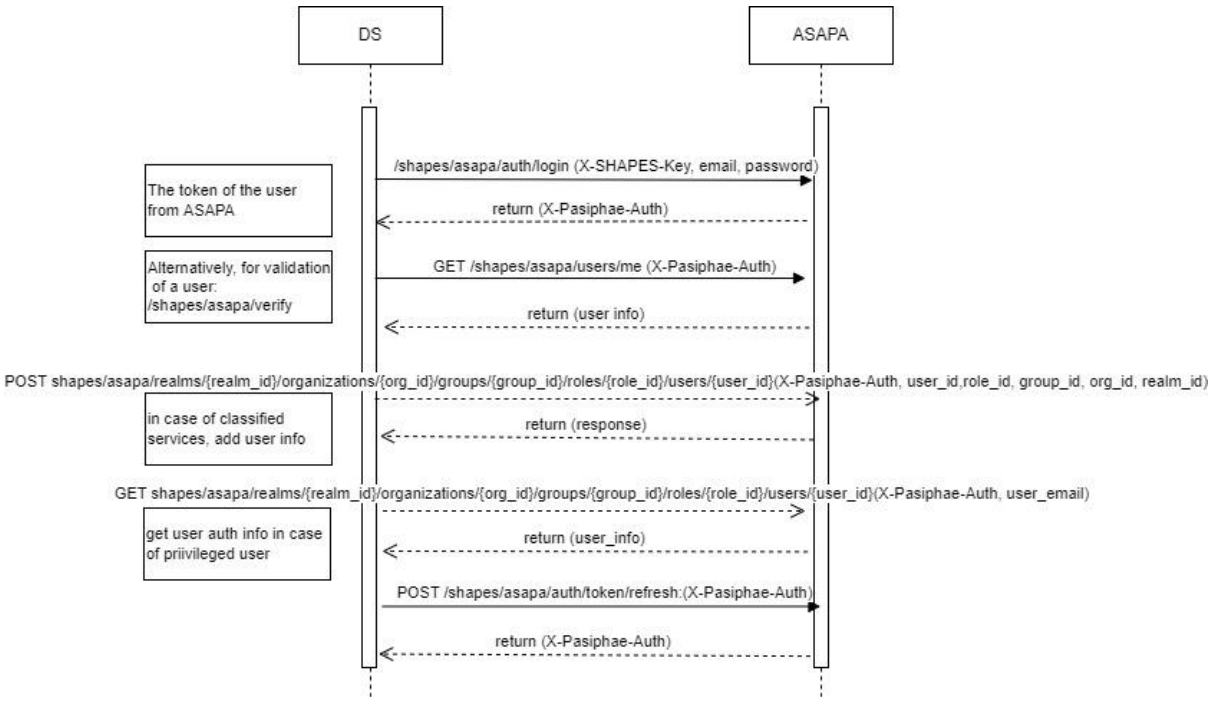

Figure 13 --User initialization workflow.

- In case the DS provides classified services (services that require different authorization rights), an initial configuration is required to be followed by the DS before any user access to the system so that the required roles for their authorization mechanism are created. A request to the following ASAPA endpoint can be made by the DS to create the different groups/roles/permissions: POST */shapes/asapa/realms/{realm_id}/organizations/{org_id}/groups/{group_id}/roles/{role_id}/permissions*.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857159.

- A user registered to ASAPA can be considered as a guest user giving the minimum set of user privileges and rights for a DS. A guest user can be created by utilising the following ASAPA endpoint: POST *shapes/asapa/auth/register*.
- After the successful registration of the user, the DS can authenticate a user using the login endpoint of the ASAPA component in order to receive a valid token which can then be used for authorization purposes: POST */shapes/asapa/auth/login*.
- To validate the user token, the */shapes/asapa/users/me* endpoint can be used. Alternatively, the GET */shapes/asapa/auth/token/verify* ASAPA endpoint can be used to verify the validity of a token, but it does not return the user authorisation information as the aforementioned endpoint.
- If a user accesses a classified service of a DS, then additional information is needed. In order to collect this information the DS needs to provide a special endpoint for the first time in order to collect the additional user profiling information or user role required. This endpoint after the collection of these data communicates with the ASAPA component to add the user info obtained by making the following POST request: *shapes/asapa/realms/{realm_id}/organizations/{org_id}/groups/{group_id}/roles/{role_id}/users/{user_id}*.
- In case of the classified service and the need of a privileged user, user's authorisation information can be received by making a GET request to the following ASAPA endpoint: *shapes/asapa/realms/{realm_id}/organizations/{org_id}/groups/{group_id}/roles/{role_id}/users/{user_id}*, which requires a valid token of the user to be provided.
- The token of a user is valid for a configurable time period. The validity period of the token is obtained in the response of the login endpoint. During this validity period the client could refresh the token as many times as they like utilising the POST */shapes/asapa/auth/token/refresh* endpoint. The token can also be verified using the GET */shapes/asapa/auth/token/verify* endpoint. If the token is not valid, the user will need to login again providing the user credentials.

The endpoints are described in details in the respective swagger documentation available at GitHub.

4.2 IoT data interoperability workflows

This section describes the main communication diagrams to interoperate, share and access IoT data within the SHAPES ecosystem, utilising the symbIoTeAPI service of the symbIoTe component, developed for the SHAPES purposes. The respective symbIoTeAPI endpoints are described in detail in the swagger documentation available at GitHub.

4.2.1 User registration to symbIoTe

A guest user, in order to access symbIoTe services for searching IoT devices and accessing their data, firstly needs to login to ASAPA to receive an authentication token

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857159.

and then, register to symbIoTe core services by making a POST request to the following symbIoTeAPI's endpoint: /symbIoTe/admin/registerUser/me/toSymbIoTe.

A PAAM user for a symbIoTe-enabled DS that needs to register the DS's IoT resources needs to register to symbIoTe, using the following symbIoTeAPI's endpoint: /symbIoTe/admin/registerpaam.

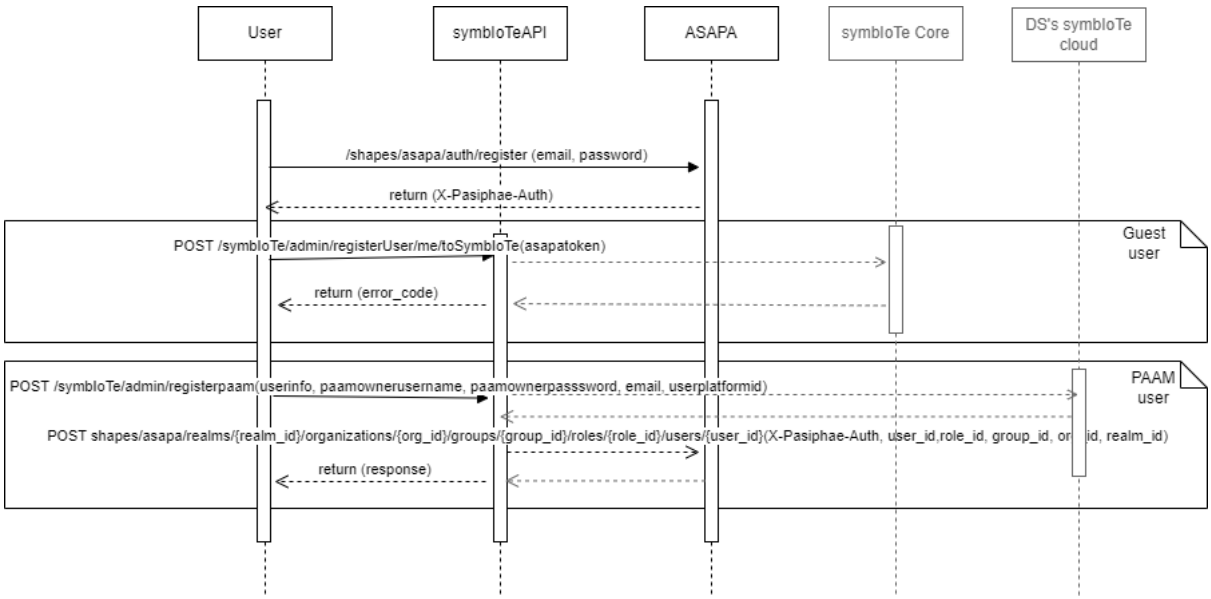

Figure 14 - User registration to symbIoTe.

Note that a PAAM user is a user registered as a platform owner and has privileged rights compared to a simple symbIoTe user.

4.2.2 Registration of IoT Resources

A PAAM user of a symbIoTe-enabled DS can register its platform's resources to symbIoTe's core services so that they can be exposed to other DSs for L1-compliance, using the following symbIoTe's API endpoint: POST /symbIoTe/resource/register/L1Res. The PAAM user firstly needs to log successfully to the ASAPA component.

Finally, the PAAM user can share the registered resource to a joined federation in order to expose its IoT resource to the federated platforms (or symbIoTe-enabled DSs) and achieve L2-compliance.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857159.

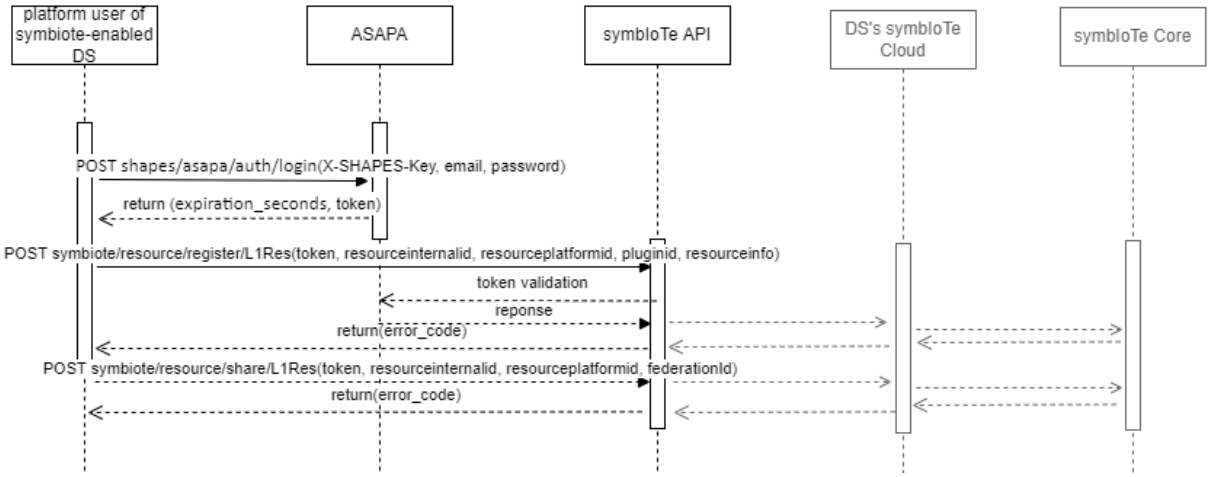

Figure 15 – Registration of IoT resources to symbIoTe.

4.2.3 Search for IoT Resources

Resource metadata of symbIoTe Core (L1-compliant)-registered resources can be discovered though a POST request to the following symbIoTeAPI endpoint: /symbIoTe/resource/get/ListOfL1. In the request made, the application client (or DS) can define the search criteria to find the resources it needs, after logging to the ASAPA component. Finally, the resource metadata for a specific resource can be obtained through a POST request to the /symbIoTe/resource/get/ListOfL1 symbIoTeAPI endpoint. The procedure described is depicted in the figure below.

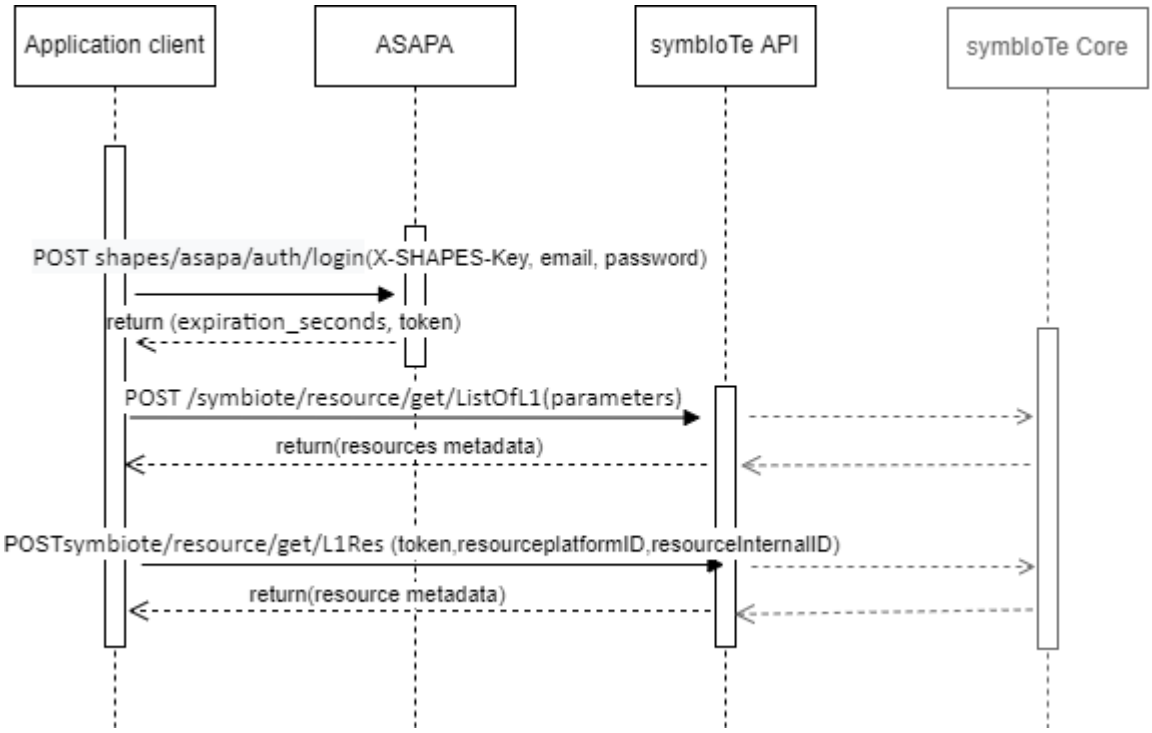

Figure 16 – Search L1 resource.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857159.

The workflow is similar for the case of federated resources (L2 compliance), utilising the respective L2 symbIoTeAPI endpoints, which query the symbIoTe Cloud services of the symbIoTe-enabled DS, as shown in the figure below.

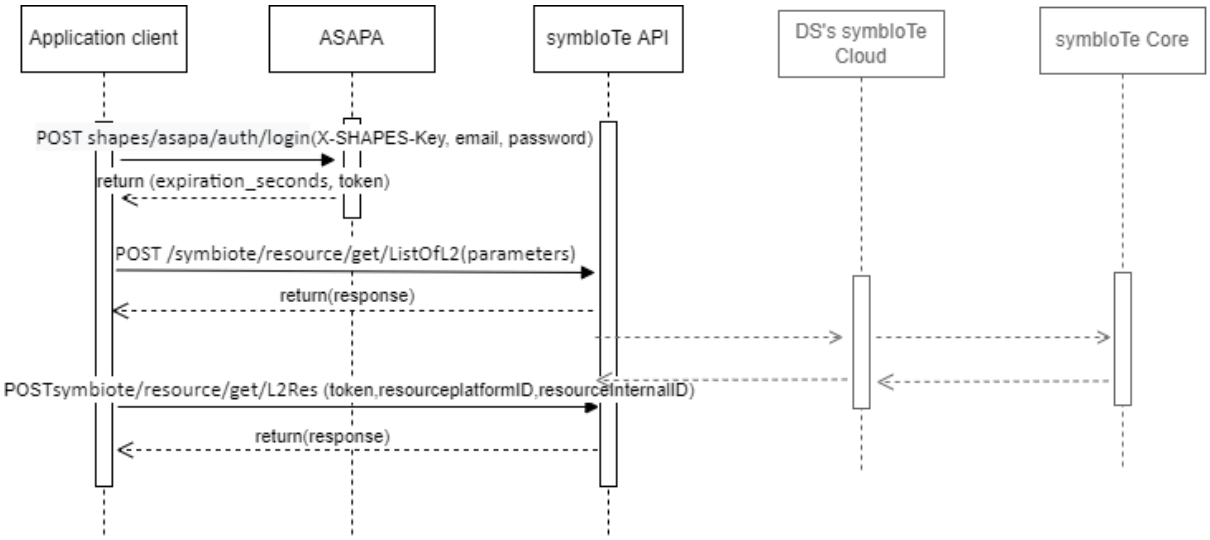

Figure 17 – Search L2 resource.

4.2.4 Access to IoT Resources

The procedure to access a resource registered to symbIoTe core (L1 compliant) and obtain the resource's observations is described in the figure below. An application client (or DS) that wants to access the resource makes a POST request to the /symbIoTe/resource/access/ symbIoTeAPI endpoint, after logging to the ASAPA component. The symbIoTeAPI forwards the request to the Resource Access Proxy (RAP) service of the symbIoTe Cloud deployed by the DS/IoT platform (e.g. FINoT platform) in order to access the resource and get the observation data. The RAP service forwards the request to the platform specific RAP plugin (the connector of symbIoTe with the IoT platform/DS developed by the platform owner) to make any adaptations required, regarding the data format and acquire the data of the platform's resource. Then, the RAP plugin forwards them to the generic part of the RAP which then handles all communication with the upper layers to send the observation data to the client. The resource metadata (resource ID) required for the request made can be obtained, if not known, using the searchL1Resource procedure, explained in the previous section.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857159.

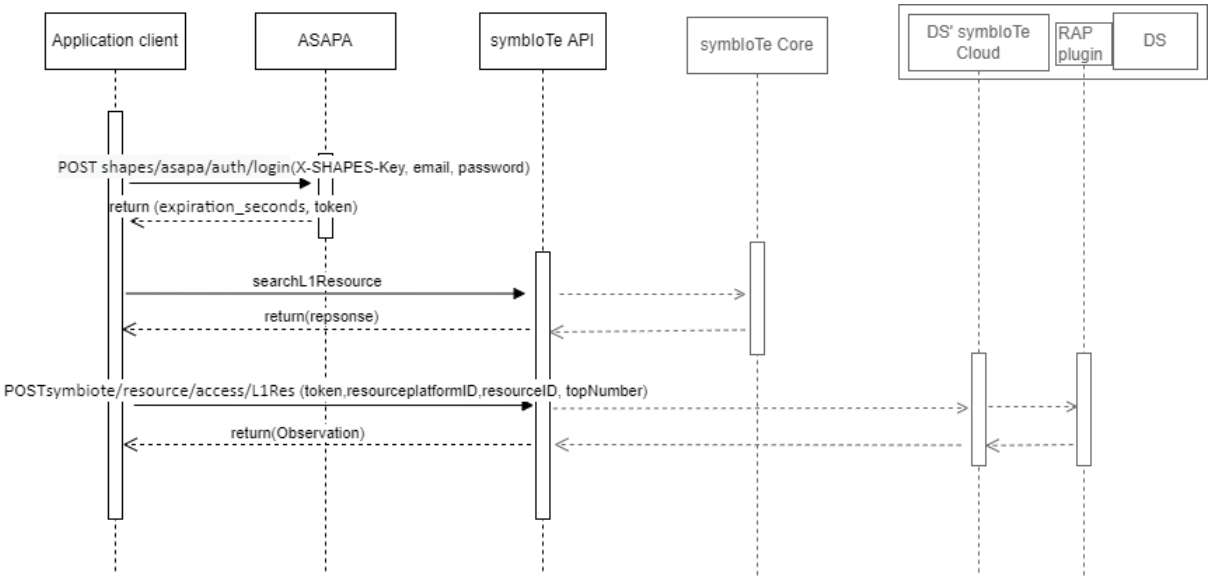

Figure 18 – Access L1 resources.

The procedure is similar for the case of federated resources (L2 compliance) utilising the corresponding L2 symbIoTeAPI components to reach the symbIoTe cloud services of the respective IoT platform (or DS) and obtain the resource observations as presented in the following figure.

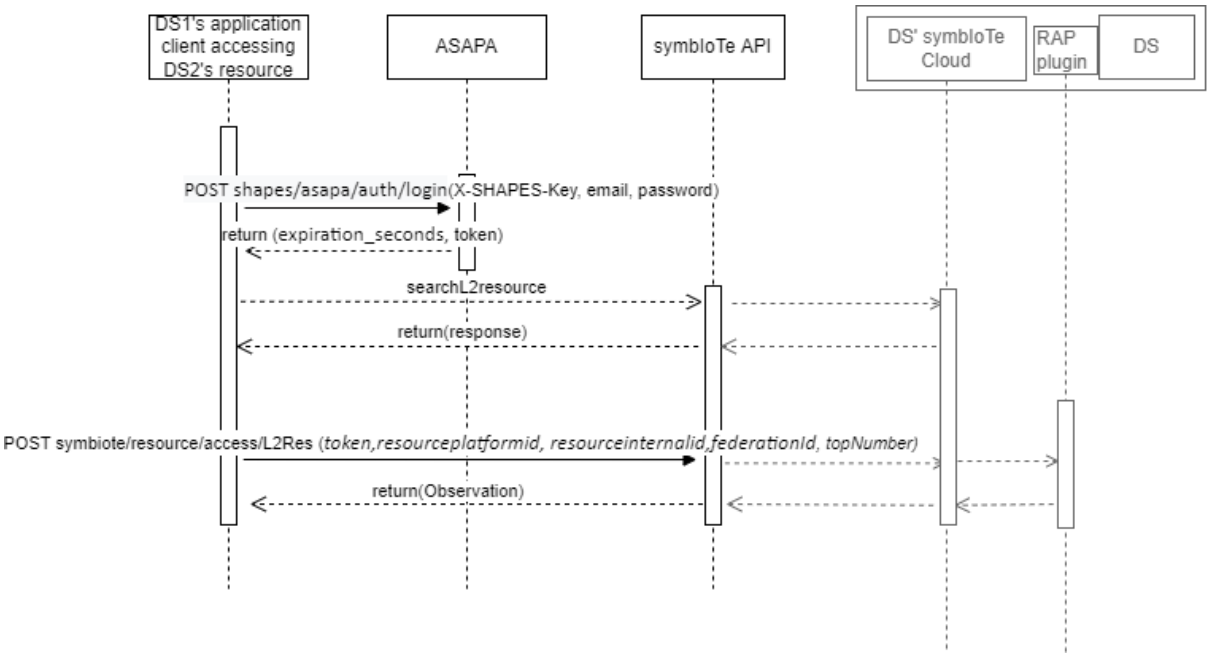

Figure 19 - Access L2 resource.

Finally, symbIoTe's mechanism for handling resource subscriptions can be utilised by a DS that wants to support resource subscriptions through the use of web sockets.

Any application client that acts as a subscriber, such as the data Lakehouse, or a DS needs to handle the web socket management in order to create the required web socket sessions and monitor their status as shown in the figure below. The application

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857159.

client sends a subscriber request to the DS that provides the resource (acting as the publisher) and starts receiving the resource observations. Finally, the subscriber needs to listen to the keep alive message sent by the publisher to keep the session alive. An unsubscribe request to the DS that is acting as publisher, notifies it that the subscriber does not want to receive data anymore and the session is closed. Finally, optional steps can be utilised, in case the application client (subscriber) is the Data Lakehouse component. The Data Lakehouse component may implement a *NotifyForData* endpoint in order to be informed by a DS to subscribe to its resource for starting receiving observations, while a *NotifyForUnsubscription* endpoint can be implemented to receive notifications to stop its subscription to the resource, when the DS does not want to send it data anymore.

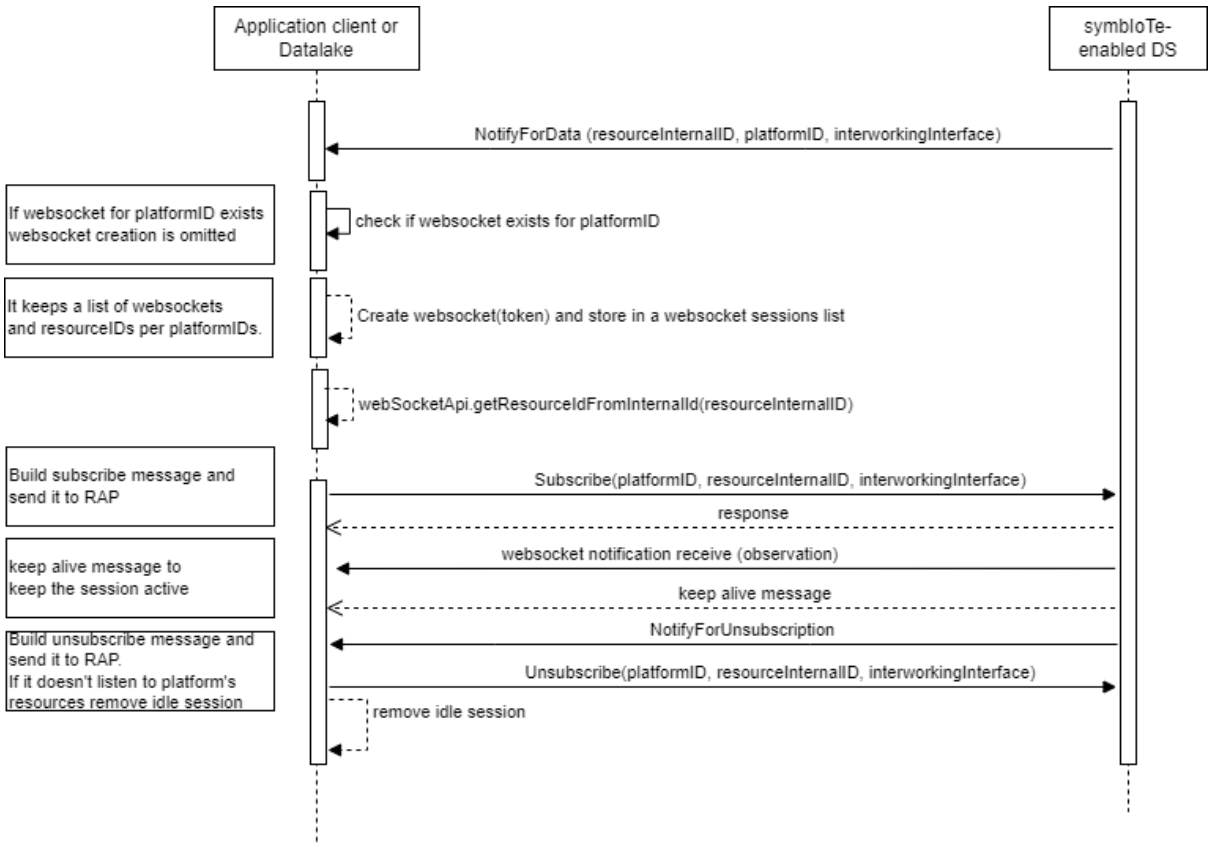

Figure 20 – Subscription to a resource.

4.2.5 Registration and access to SHAPES Gateway

This section describes the workflows for the connection to the Smart Space (SSP). The SSP is part of the SHAPES Gateway component and enables the connection between IoT platforms, IoT gateways and smart devices within a local environment to communicate and be accessible through symbIoTe.

In order for n IoT platform or an IoT gateway to be registered to the SSP through the SSP gateway, it needs to send a request to the *Inkeeper* subcomponent of the SSP in order to register and receive its unique identifiers, namely the *symId* and *sspId*. Resources (devices) are connected to the gateway to broadcast their data. The IoT gateway or IoT platform registers its resource(s), by sending a join request to the SSP

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857159.

RAP, a subcomponent of the SSP and receive the unique identifier for each registered resource. After the successful registration of the resources, the same procedure described in earlier sections for searching and accessing an L1 resource, can be followed to search and access the resources registered to the SSP, by an IoT gateway/IoT platform from outside the SSP. The aforementioned procedure is also used for the connection of FINT's IoT gateway to symbIoTe's smart space, which comprise the SHAPES gateway.

Finally, symbIoTe L3, also, allows a smart device (a device where symbIoTe software can be installed) directly to the SSP. The procedure is similar to the case of an IoT platform or IoT gateway registering to the SSP with the device endpoints now being used (/innkeeper/sdev instead of /innkeeper/platform at the figure below).

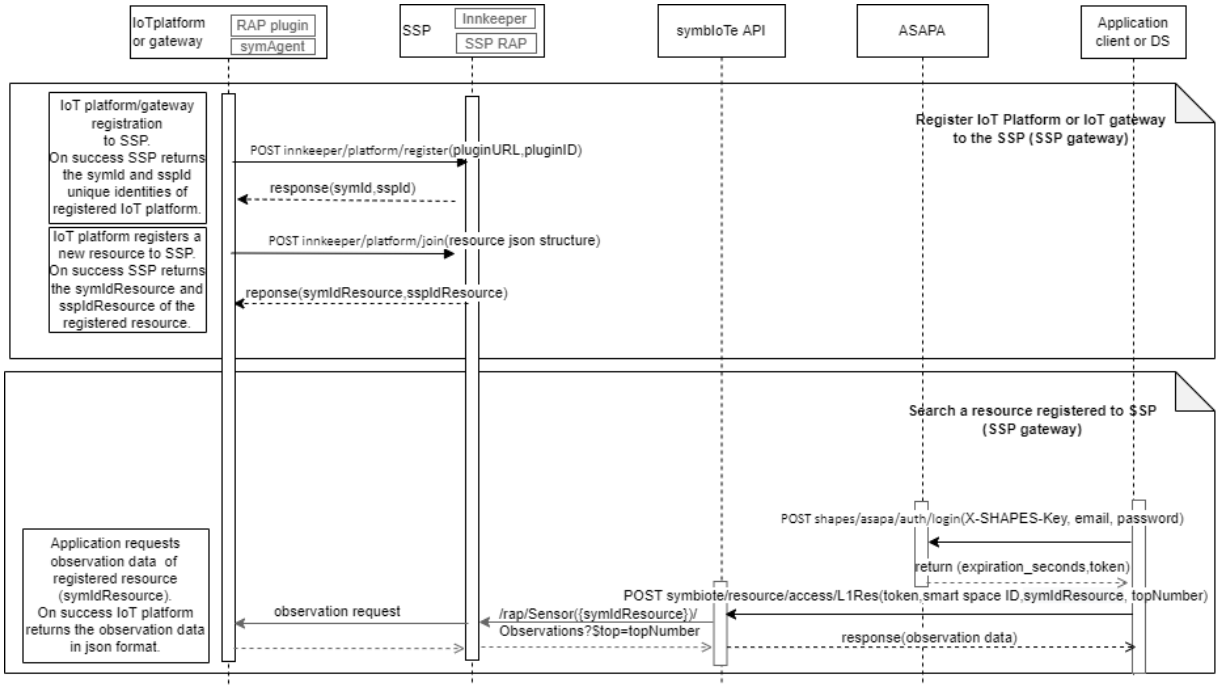

Figure 21: Registration to SSP and access to resources (L3).

The following figure describes the workflow where the FINoT platform is used for storing resource data of DSs that do not have storage for keeping and sharing historical data. The differentiation from the previous diagram is that the SHAPES gateway (FINT IoT Gateway connected to the SSP) now sends the data of the resources to the FINoT platform to store them. When a request for obtaining historical data is made, the SHAPES gateway forwards the request to the FINoT platform to acquire the data stored.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857159.

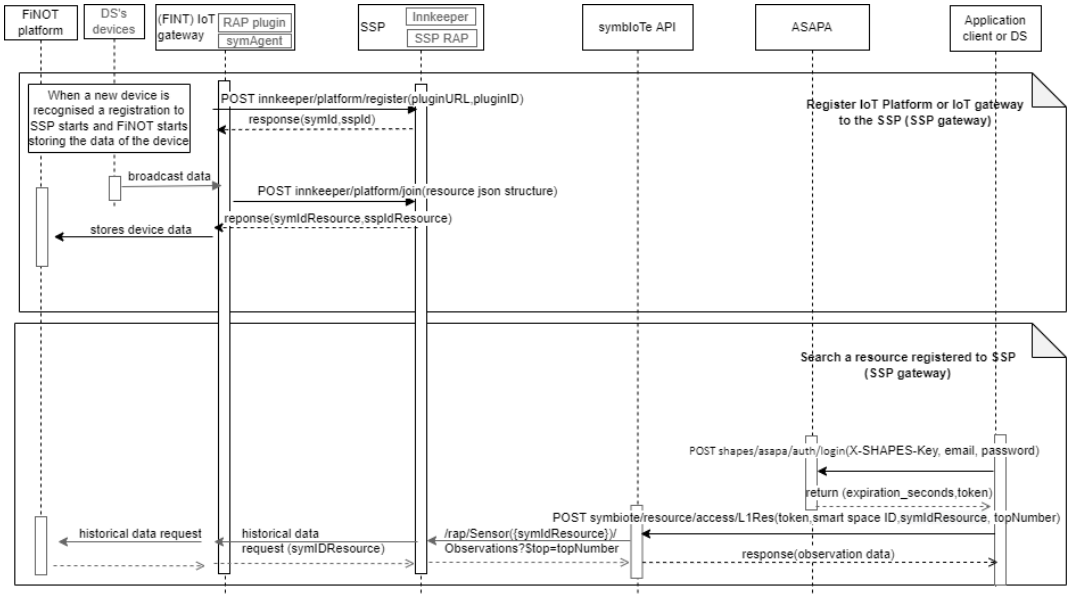

Figure 22: Storing of IoT data to FINOT platform.

4.3 Exchange of FHIR data

The diagram below describes the communication required for exchanging FHIR messages through the use of the FHIR message broker. Any Digital Solution (DS) that wants to send or receive FHIR messages needs to be a SHAPES user registered to ASAPA. Therefore, a DS that wants to send FHIR data through the FHIR component firstly needs to login to the ASAPA component and obtain a valid token. Then, the DS needs to register to the FHIR MQ by providing the ASAPA token. The FHIR MQ validates the token and confirms the registration. Subsequently, the DS can create a new topic and send a message.

Likewise, any DS that wants to obtain data specific to a topic, needs to login to ASAPA, obtain a valid token and register to FHIR MQ using the token. After that, the DS can request the list of the data providers and the available topics in order to choose the ones they are interested in subscribing to. In order to subscribe to a topic, the DS needs to provide the endpoint to which the FHIR MQ will forward the messages. A DS can, also, stop receiving messages by deleting the respective subscription.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857159.

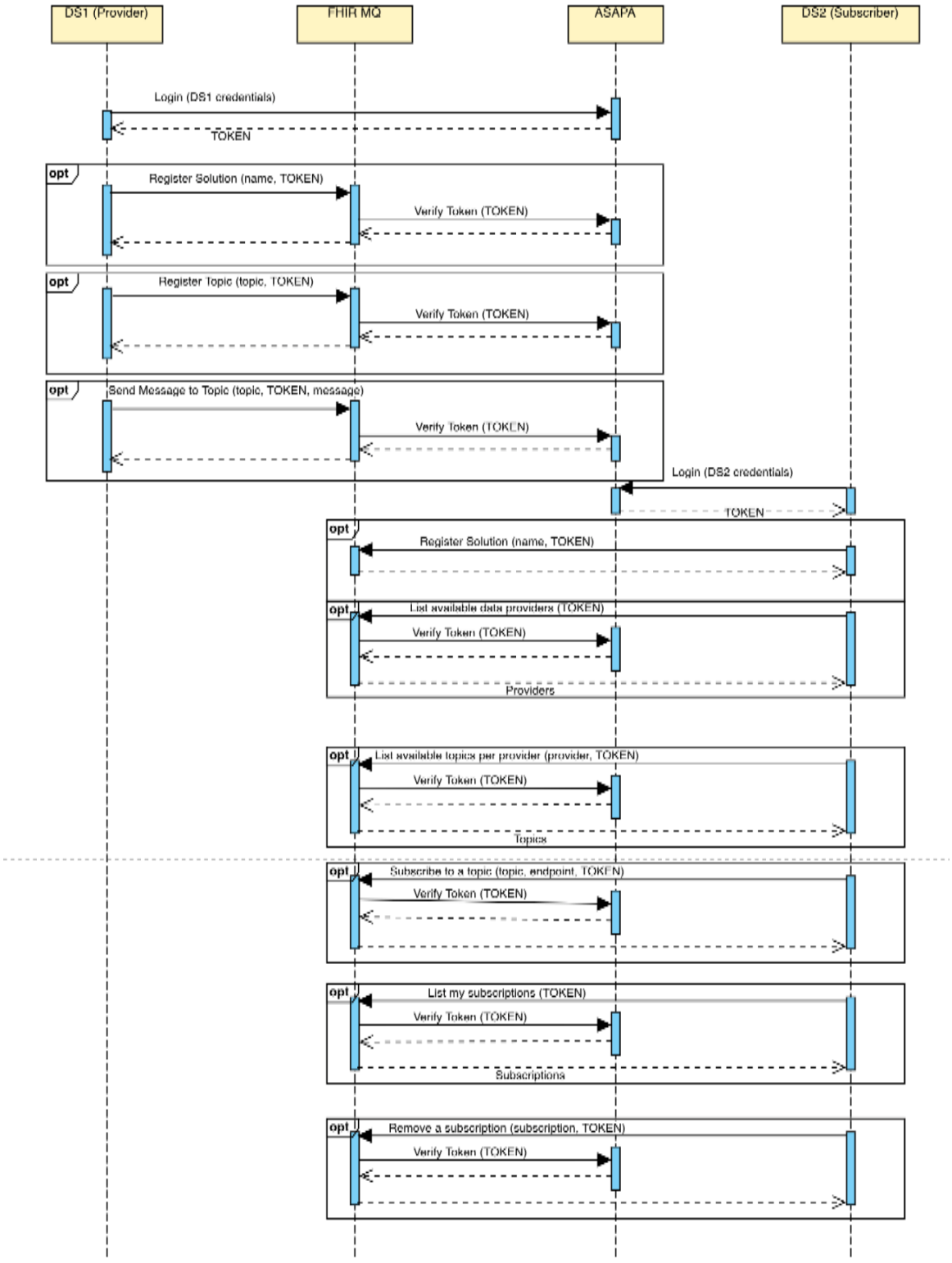

Figure 23: FHIR Workflow.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857159.

4.4 Data Lakehouse Workflow

Figure 24 provides a general overview of the next represented workflows, regarding the interaction workflow of the Data Lakehouse. The platform allows data ingestion, execution and transformation of the analysis deployed in the platform and data query. Although these processes are represented in a sequential mode for better visualization and understanding, these phases are done in parallel.

The involved agents are: the Digital Solution- DS (ds1: DS) that ingest the data, the DataLake, the DS that consume the data (ds2:DS) and the Analytics Controller, that is responsible for launching the analysis.

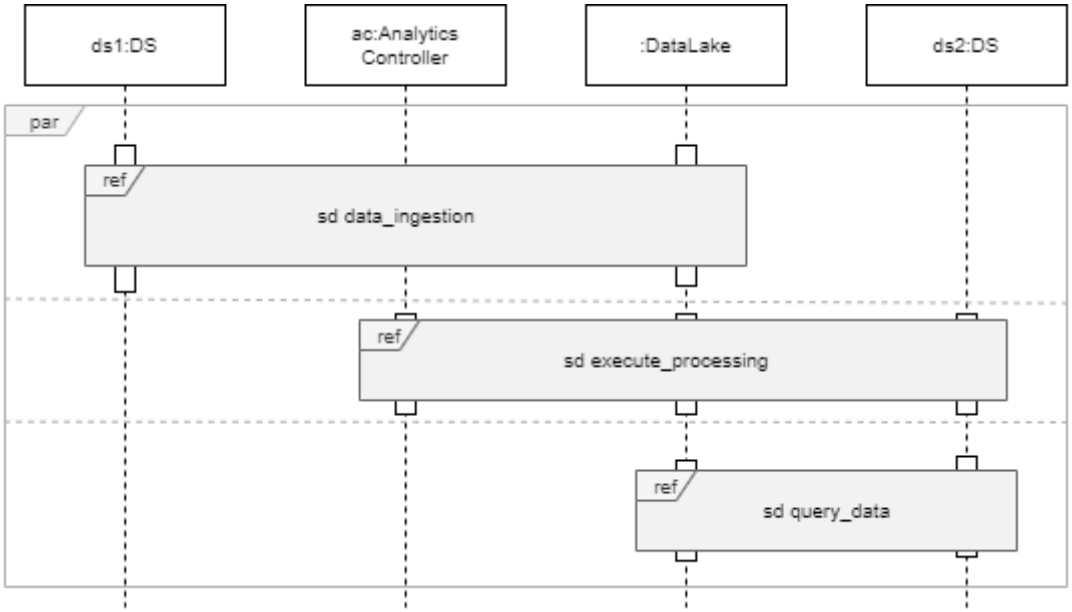

Figure 24 – Data Lakehouse Workflow.

4.4.1 Data Ingestion Workflow

We have two data types: IoT data (sent by DS1a) and non-IoT data (sent by DS1b). Within the IoT data scheme, the DS sends data through symbIoTe (Section 4.4.1.1). Regarding non-IoT data, more information can be seen in Section 4.4.1.2.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857159.

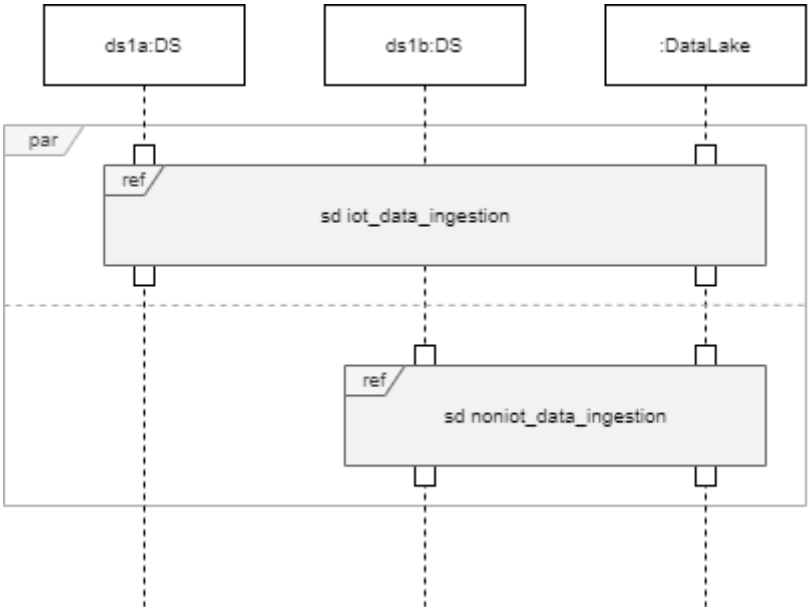

Figure 25 – Data Ingestion Workflow.

4.4.1.1 IoT Data Ingestion Workflow

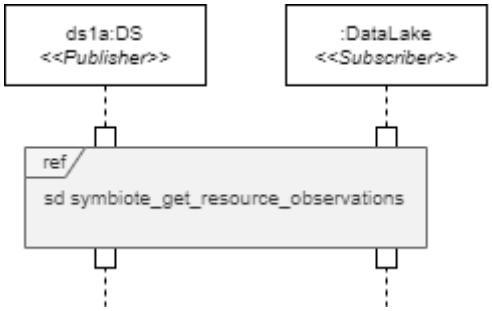

Figure 26 – Data Ingestion for IoT data

4.4.1.2 Non-IoT Data Ingestion Workflow

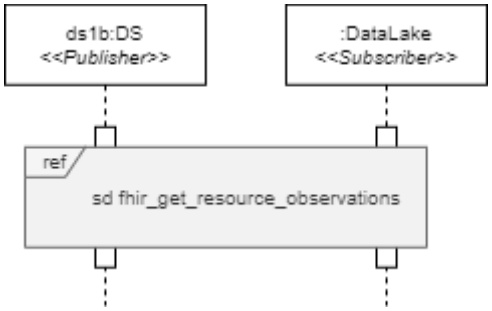

Figure 27 – Data Ingestion for non-IoT data.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857159.

4.4.2 Data Processing Execution

In this schema, the “Analytics Controller” is responsible for requesting the execution of the processing. The “Message Broker” is the communication mechanism between the “Analytics Controller” and the Data Lakehouse, which executes the processing. It is the message queue the DS2 listens to when an analysis finishes.

The process is: the Analytics Controller publishes an analysis execution request, through the message broker; the Data Lakehouse is listening to the message broker (message queue) and once the execution request is received, it then performs the analysis. Once the execution finishes it also publishes a message information that the results are available. The DS2 that listens to it receives that information.

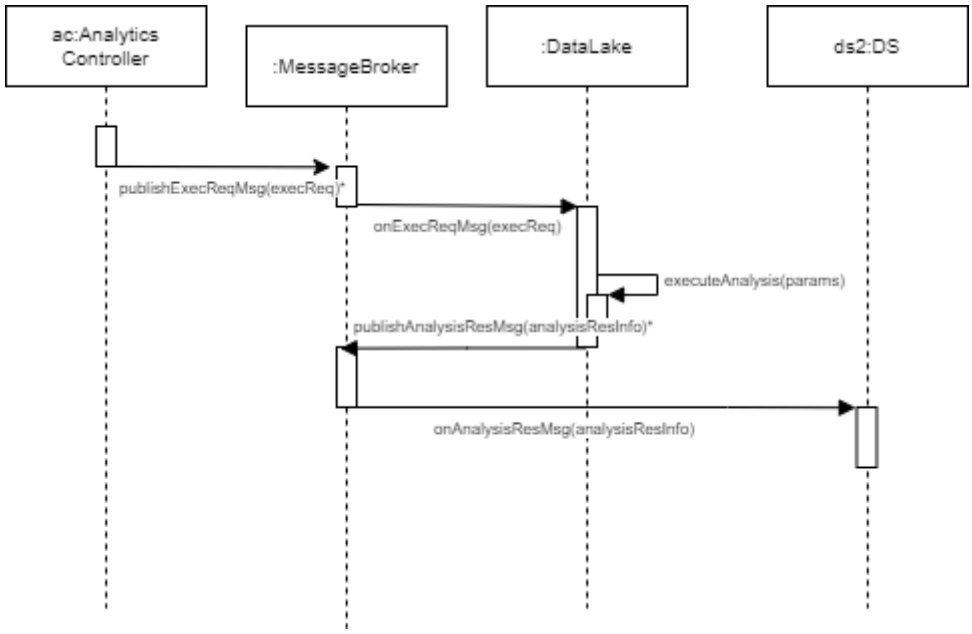

Figure 28 – Data Processing

4.4.3 Data Query

To consult the data, first the ASAPA login needs to be executed (Section 4.1). Once the DS (DS2) has the ASAPA token, it invokes the operation query from the Data Lakehouse. If the token is valid, the Data Lakehouse validates if the user has access to these data (permissions) and executes the query to return the result. Otherwise, it informs the DS with the respective error (e.g., invalid token or unauthorised access).

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 857159.

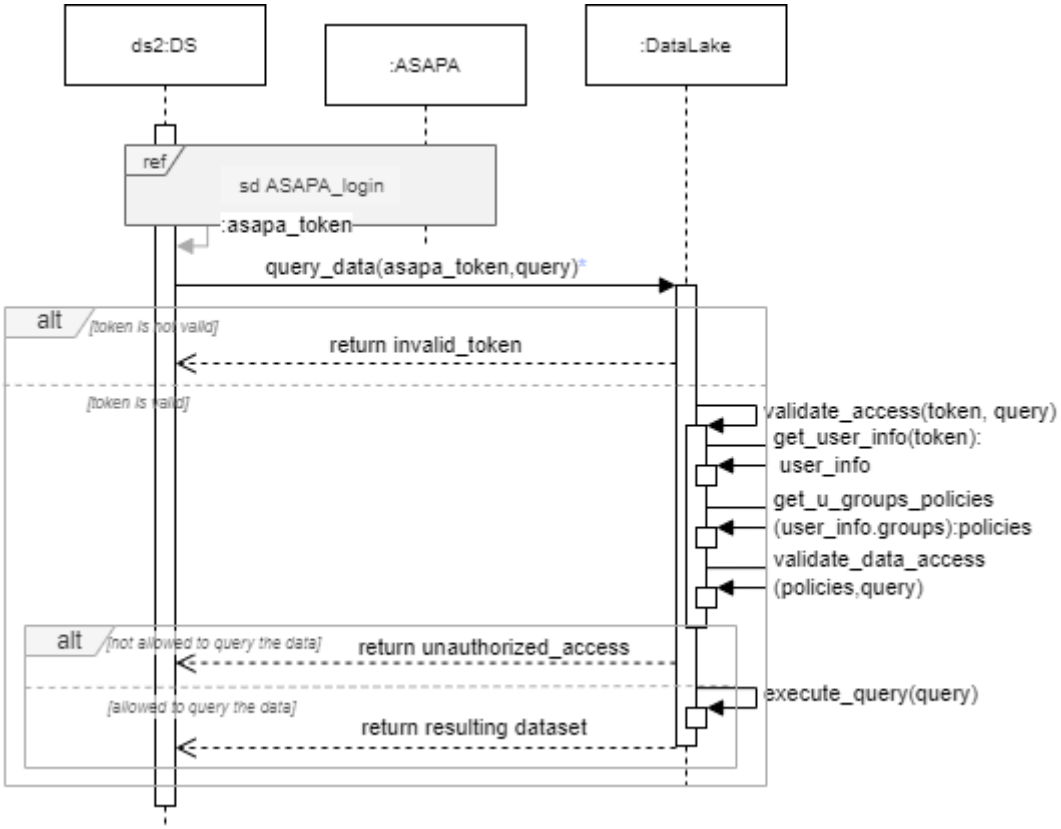

Figure 29 – Data Query.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857159.

4 Results and Conclusions

This deliverable details the source-code and libraries, produced as a result of the following WP4 tasks:

- **Task 4.3** "*Implementation of Mediation Framework & Interoperability Services*"
- **Task 4.4** "*Implementation & Deployment of Secure Cloud & Big Data Platform*"
- **Task 4.5** "*Human Interaction & Visual Mapping*"
- **Task 4.6** "*SHAPES Authentication, Security & Privacy Assurance*"
- **Task 4.7** "*SHAPES Gateway Reference Implementation*"
- **Task 4.8** "*Integration and Testing of SHAPES TP*"

This report provides the structure of the developed component's software, the build and deployment requirements, the functionalities of the individual components and dependencies specific to them. Finally, the flow of information between the SHAPES core components and interaction with them is also described, in the form of sequence diagrams. These sequence diagrams describe the complete workflow cycle for the interaction of the user or Digital Solution within the SHAPES ecosystem.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857159.

5 Ethical Requirements Check

The focus of this compliance check is on the ethical requirements defined in D8.4 and having impact on the SHAPES solution (technology and related digital services, user processes and support, governance-, business- and ecosystem models).

Ethical issue (corresponding subsection of D8.4 in brackets)	How it has been taken into account in this deliverable (if relevant)
Fundamental Rights (3.1)	Not applicable
Biomedical Ethics and Ethics of Care (3.2)	Not applicable
CRPD and supported decision-making (3.3)	Not applicable
Capabilities approach (3.4)	Not applicable
Sustainable Development and CSR (4.1)	Not applicable
Customer logic approach (4.2)	Not applicable
Artificial intelligence (4.3)	Not applicable
Digital transformation (4.4)	Not applicable
Privacy and data protection (5)	Not applicable
Cyber security and resilience (6)	Not applicable
Digital inclusion (7.1)	Not applicable
The moral division of labour (7.2)	Not applicable
Care givers and welfare technology (7.3)	Not applicable
Movement of caregivers across Europe (7.4)	Not applicable

NOTE: based on the fact that this deliverable outlines only software support to using the SHAPES Technological Platform by 3rd-parties and thus containing ONLY public information, excluding any references to proprietary project data and any identifiable private information (excluding names of authors and contributors) none of the above Ethical issues applies.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857159.

6 References

Only links to EC-reviewed deliverables published on SHAPES portal are provided.

[1] SHAPES Consortium (2020).
D4.1 – SHAPES TP Requirements and Architecture – MAY-2021.

[2] SHAPES Consortium (2020).
D5.3 – SHAPES Digital Solutions V2 – OCT-2021.
D5.1 - SHAPES Digital Solutions V1 – AUG-2022.

[3] SHAPES Consortium (2020).
D4.3 – Integration Plan and Test Cases – OCT-2021

[4] SHAPES Consortium (2020).
D4.6 – SHAPES Interoperability Reference Testing Environment – OCT-2021

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857159.