5G European Validation platform for Extensive trials

Deliverable D4.2 First version of the experimental portal and service handbook

Project Details

Call	H2020-ICT-17-2018
Type of Action	RIA
Project start date	01/07/2018
Duration	36 months
GA No	815074

Deliverable Details

Deliverable WP:	WP4
Deliverable Task:	Task T4.2
Deliverable Identifier:	5G_EVE_D4.2
Deliverable Title:	First version of the experimental portal and service handbook
Editor(s):	Jaime Garcia-Reinoso (UC3M)
Author(s):	Jonathan Almodóvar Herreros, Ginés García Avilés, Jaime García Reinoso, Carlos Guimaraes, Winnie Nakimuli, Cristina Quintana Jordán, Pablo Serrano Yáñez-Mingot (UC3M); Giada Landi, Juan Brenes, Francesca Moscatelli, Elian Kraja, Giacomo Bernini (NXW); Ramón Pérez (TELC); Grzegorz Panek (ORA-PL)
Reviewer(s):	Sofiane Imadali, Rodolphe Legouable, Louiza Yala (ORA-PL); Kostas Trichias (WINGS)
Contractual Date of Delivery:	31/12/2019
Submission Date:	20/12/2019
Dissemination Level:	PU
Status:	Final
Version:	1.0
File Name:	5G EVE – D4.2 First version of the experimental portal and service handbook

Deliverable History

Version	Date	Modification	Modified by
V0	*17/10/2019*	*First ToC*	*Jaime Garcia-Reinoso*
V0.1	*21/10/2019*	*Assign responsible to sections*	*Jaime Garcia-Reinoso*
V0.2	*14/11/2019*	*Portal architecture and service handbook*	*Jaime Garcia-Reinoso*
V0.3	*27/11/2019*	*First round of contributions*	*All authors*
V0.4	*04/12/2019*	*Second round of contributions*	*All authors*
V0.5	*05/12/2019*	*Version ready for reviewers*	*All authors*
V0.6	*13/12/2019*	*Version tackling reviewers' comments*	*All authors*
V1.0	*15/12/2019*	*Editorial changes*	*Jaime Garcia-Reinoso*

Table of Contents

List of Acronyms and Abbreviations

Acronym	Description
API	Application Programming Interface
BLUO	Browse-and-Lookup, Uploading and Onboarding
CB	Context Blueprint
CD	Context Descriptor
CPU	Core Processing Unit
CRUD	Create Read Update Delete
CSS	Cascading Style Sheets
DB	Database
DCM	Data Collection Manager
DCS	Data Collection and Storage
DF	Deployment Flavour
E2E	End to End
EBB	Experiment Blueprint Builder
EEM	Experiment Execution Manager
ELM	Experiment Lifecycle Manager
eMBB	Enhanced Mobile BroadBand
EPC	Evolved Packet Core
ETSI	European Telecommunication Standard Institute
ExpB	Experiment Blueprint
ExpD	Experiment Descriptor
GUI	Graphical User Interface
HTML	Hyper Text Markup Language
HTTP	Hyper Text Transfer Protocol
IBN	Intent Based Networking
IBNS	Intent Based Networking System
ID	Identifier
IAM	Identity and Access Manager
IDMS	Intent Driven Management System
IL	Instantiation Level
I/W	Interworking Layer
JPA	Java Persistency API
KPI	Key Performance Indicator
MANO	Management and Orchestration
MEC	Multi-Access Edge Computing
MSO	Multi-Site Orchestrator
MTC	Machine Type Communication
NBI	North Bound Interface
NFV	Network Function Virtualization
NFVO	Network Function Virtualization Orchestrator
NSD	Network Service Descriptor
PNF	Physical Network Function
RBAC	Role Based Access Control
REST	REpresentational State Transfer
SLA	Service Level Agreement
SNMP	Simple Network Management Protocol
TCB	Test Case Blueprint
TSB	Ticketing System Backend
URL	Uniform Resource Locator
URLLC	Ultra-Reliable Low-Latency Communication
VIM	Virtual Infrastructure Manager
VM	Virtual Machine
VNF	Virtual Network Function
VNFD	Virtual Network Function Descriptor
VNFFG	Virtual Network Function Forwarding Graph
VSB	Vertical Service Blueprint
VSD	Vertical Service Descriptor
WP	Work Package

List of Figures

List of Tables

Executive Summary

The first release of the 5G EVE Portal is a major milestone in the 5G EVE end-to-end facility implementation roadmap. This first version of the Portal provides important services to all actors involved in 5G-EVE experimentation. One important decision taken by the 5G EVE project is to open its interfaces to other projects, but this has implications in the design and implementation of the Portal. Thus, the 5G EVE Portal architecture has been designed to provide services through a backend layer, which can be consumed by the graphical user interface implemented and offered by the 5G EVE projects. Furthermore, these services may be used by other authorised projects too.

This document, building on deliverable D4.1, contains the details about the 5G EVE Portal, including its constituent components, all interfaces connecting these modules as well as the APIs offered by the backend towards other projects and experimenters.

Apart from the integration of the components already presented in D4.1, this deliverable also offers some details of new elements, and how they interact with others. The 5G EVE project pays special attention to security aspects, so the central component of the backend layer is the Role-Based Access Control (RBAC) component. All other components in the backend have to authenticate and authorize all requested actions using this component. For example, the Portal Catalogue Service only responses to authorized users about the content included in its databases; the Data Visualization tool restricts the access to metrics to authorized users too, etc. The Experiment Lifecycle Manager centralizes the most important actions related with the lifecycle of an experiment such as its preparation, execution and finalization, and it is precisely defined in this document. Finally, the File Storage component is described in this document as well. The main service provided by this component is to allow the uploading and downloading of VNF packages, which have to be onboarded by site managers.

Last but not least, this deliverable provides a service handbook to all actors involved in a 5G experiment using the E2E facility provided by the 5G EVE project. The handbook presents an overview of all phases related with an experiment, from the design stage to the execution of an experiment, the actors involved in each step, the components of the graphical interface intended to implement each step and how to implement each task.

Introduction

This document accompanies the first release of the experimental 5G EVE Portal software, including the most important information about its design process, the resulting architecture, which includes the northbound interface to provide services to other users, and implementation details. The present deliverable, together with deliverable D4.1 [1], provide the complete information about the services provided by the first release of the 5G EVE Portal, which are summarized in Table 1.

Table 1. Services provided in the first release of the 5G EVE Portal.

Actor	Service
Anonymous User	Signup, login to Portal.
Experiment/ Developer	Browse and look up tool to view a list of available components in the 5G EVE end-to-end facility.
Experiment/ Developer	Upload VSB, ExpB and Test Case Blueprint. These actions will finish with an OK or Error with a given code.
Experimenter	List available VSB based on the provided query.
Experimenter	List required/missing parameters for the VSB provided.
Experimenter	Post parameters to create a service for the selected VSB and returns an experiment identifier (ExpID), which is in an incomplete state.
Experimenter	List available ExpB based on the provided (*incomplete*) experiment identifier (ExpID).
Experimenter	Get required parameters of the provided ExpB identifier.
Experimenter	With the ExpB selected, experimenters have to add all necessary tests. If successful, the provided ExpID state changes to *completed*.
Experimenter	With the ExpID (must be in *completed* state), this actor creates an Experiment Deployment.
Experimenter/ Site Manager	These actors have access to the status of the experiment deployment object: pending, accepted, rejected, pre-provisioned, ready, running, finished, failed or deleted.
Experimenter	Experimenters may instantiate experiments.
Experimenter	Experimenters may monitor metrics while experiments are running.
Experimenter	Experimenters may visualize the results of their executed (finished) experiments.
Experimenter	Experimenters could delete pending, accepted, finished or failed experiments.
Experimenter	Add/view/comment tickets.
VNF provider	Users may upload and request the onboarding of VNFD to a list of trial sites. Results could be OK or Error with code. This step may generate tickets to the site managers.
Site manager	Add/view/modify tickets.
System administrator	Administrators may execute all services available in the system.

This document also includes a service handbook intended to be used as a guideline for all different actors involved in the definition, preparation and execution of a 5G experiment to implement the proper actions. The service handbook identifies all these actors and the proper element at the graphical interface to perform a particular task.

The rest of the document is organized as follows. Section 1 starts presenting the experimental portal architecture, including subsections to introduce all modules available in this first release, its main functionalities and the northbound interfaces, which may be used by other projects to access the services provided by the portal. Section 2 presents some details about the implementation of the portal. Section 3 presents the service handbook, which offers the workflow to design, prepare, execute and experiment, as well as the main components providing the services to tackle each phase of the experiment lifecycle. Finally, Section 4 concludes with the most important remarks already described in the document and the future work for the second release of the portal.

1 Experimental Portal architecture

Deliverable D4.1 [1] reports the main functional requirements of the 5G EVE Portal, presenting the main experimentation tools and the 5G EVE catalogue designed to cope with some of such requirements. This document follows all recommendations presented in D4.1 to extend available functionalities provided by these experimentation tools and catalogue to encompass all services that have to be offered to all actors, described in that deliverable as well. After a deep analysis of D4.1 and other deliverables produced by the 5G EVE project, this document presents the 5G EVE Portal architecture, shown in Figure 1 (Table 2 presents the main acronyms used in the architecture). For the sake of completeness, Figure 1 shows not only the Portal architecture but some other elements also defined by the 5G EVE project, in order to better describe all interactions between the Portal with the Interworking Layer, mainly.

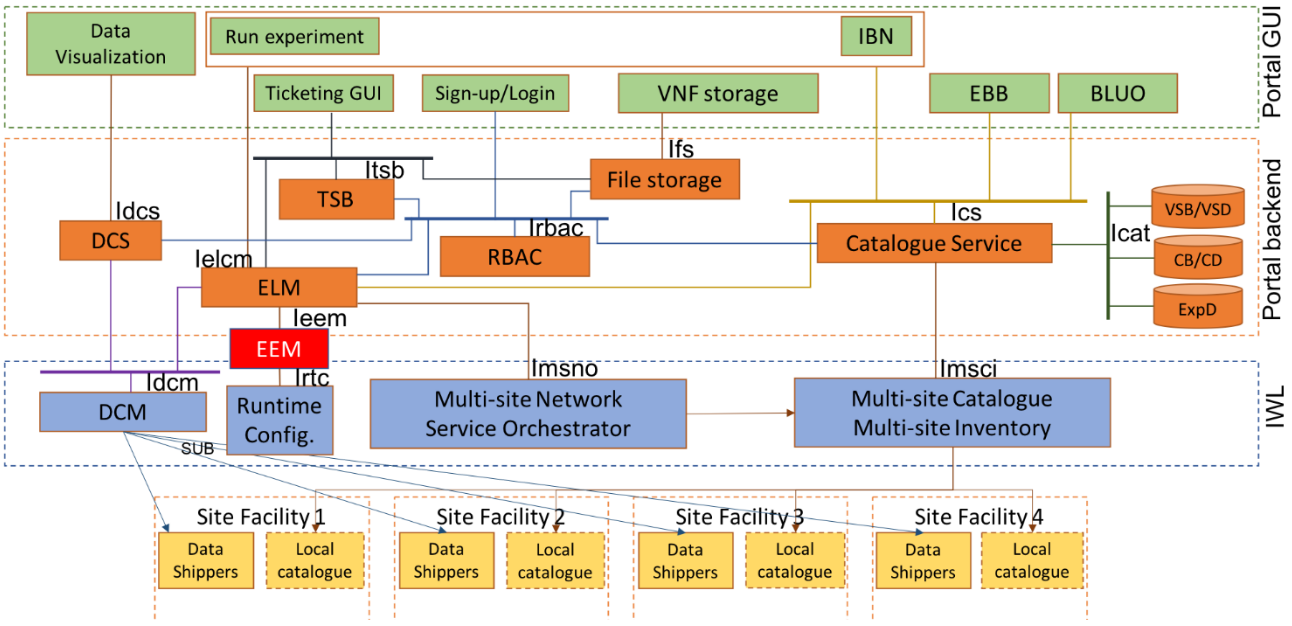


Figure 1: 5G EVE Portal architecture

Table 2. Acronyms used in the 5G EVE Portal architecture.

Acronym	Description	Acronym	Description
BLUO	Browse, Look-Up and Onboarding Tool	ELM	Experiment Lifecycle Manager
CB	Context Blueprint	ExpD	Experiment Descriptor
CD	Context Descriptor	IBN	Intent-Based Networking
DCM	Data Collection Manager	RBAC	Role-Based Access Control
DCS	Data Collection and Storage	TSB	Ticketing System Backend
EBB	Experiment Blueprint Builder	VSB	Vertical Service Blueprint
EEM	Experiment and Execution Manager	VSD	Vertical Service Descriptor

One of the main decisions taken while designing this architecture is to split the 5G EVE Portal in two sublayers: the Portal backend and the Portal Graphical User Interface (GUI). This way, the Portal backend exposes different interfaces to the outside, which will be used by the Portal GUI to offer its own services and which may also be used by other entities to consume the services offered by the Portal backend.

This section provides a detailed description of all components of the 5G EVE Portal architecture, the services offered by each block and the interfaces to consume such services. Following the sub-layer approach, section 1.1 presents all components included in the backend and delivered in the first release of the Portal, section 1.2

describes in detail all interfaces exposed by the backend, section 1.3 includes the main interfaces used by the backend to provide its own services, and finally section 1.4 covers the Portal GUI components.

1.1 Portal backend components

This subsection presents the description of all components included in the portal backend.

1.1.1 Role-Based Access Control

The Role-Based Access Control (RBAC) is in charge of providing users management and authentication/authorization functionality. This component relies on an Identity and Access Management service (IAM) to store users, generate and renew authentication artefacts that belongs to each user. The RBAC component will expose a REST API in order to make all the functionality available to the front-end application and for other back-end components that might need it, avoiding HTTP redirections.

Users' management consists of a set of functionalities to create, read, update and delete users. Figure 2 shows the interactions between the different modules involved in the registration process.

The authentication process is token-based, which means that registered users will be able to obtain an access token, valid for a specific period of time, which will allow them to fetch specific resources from other back-end components. Together with the access token, they will receive a different token called "refresh token", whose main goal is to allow users to obtain a new access token once it is expired. Finally, RBAC will provide logout functionality, making both access and refresh tokens invalid. Figure 3 shows the interactions between front-end and RBAC during the authentication process.

The RBAC component will be interacting with the IAM service, hence it will be registered as client of the latter. By registering the RBAC component at the IAM service, the RBAC will become a trusted component for the IAM service, which means that it will acquire a unique authentication key for a continuous, easy and trusted communication.

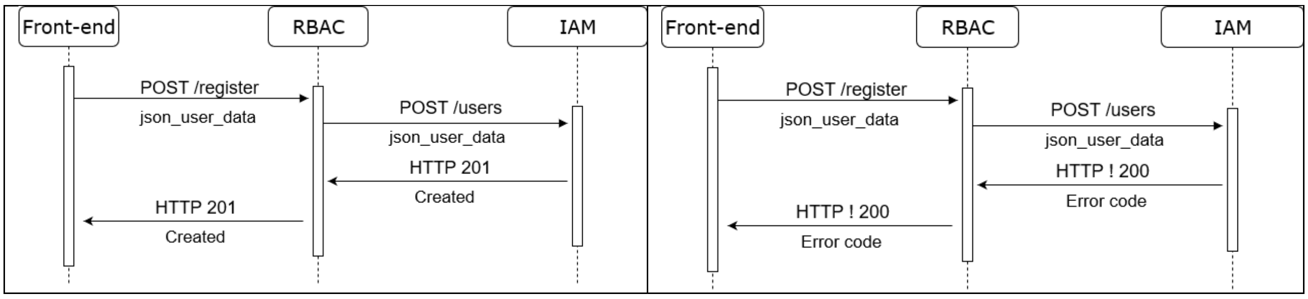


Figure 2: Registration process sequence diagram.

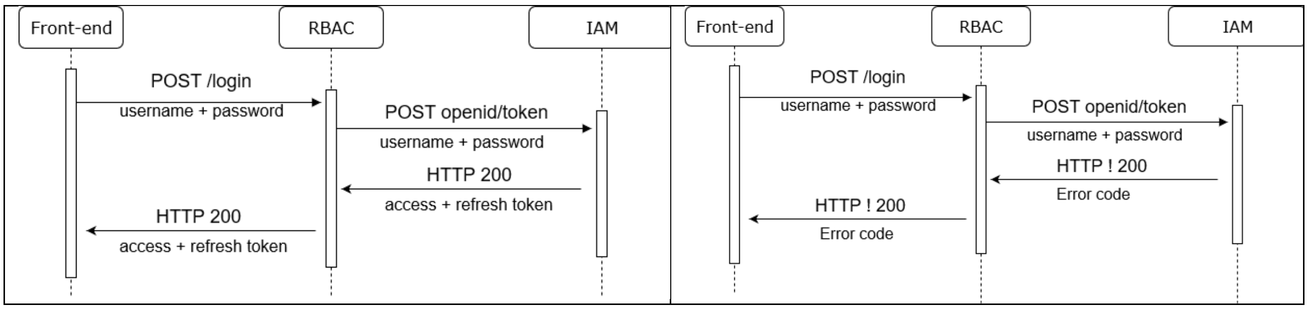


Figure 3: Authentication process sequence diagram.

Authorization

Authorization is a process by which a certain entity can determine whether a client has permissions to perform a specific action over specific resources or not. In order to do so, The RBAC component will provide roles, which usually identify a type or category of user and groups that usually are used to affiliate users with different roles. Both roles and groups can be assigned to specific users in order to be able to determine their permissions over certain resources. By registering each back-end component (DCS, ELCM, etc.) at the IAM service, they will be made trusted components that will easily interact with the IAM service. With this approach, each service will be able to, i) validate access tokens included at the client requests and, ii) retrieve information about the owner of the token (role and groups). Figure 4 shows the different interactions between modules during the authorization process.

The IAM service also allows trusted components to create their own roles and groups. Roles can be defined at two different levels:

- "Site" level: Roles available across all the applications that belongs to a specific realm. In our case, all services (DCS, ELCM, etc.) will be placed at the same realm.
- Client level: Roles available in a specific namespace dedicated to a client. All the services of 5G-EVE will be clients for Keycloak [2], so they will be able to manage their own set of roles.

Unlike roles, groups created by trusted components are, by default, available to all the trusted components.

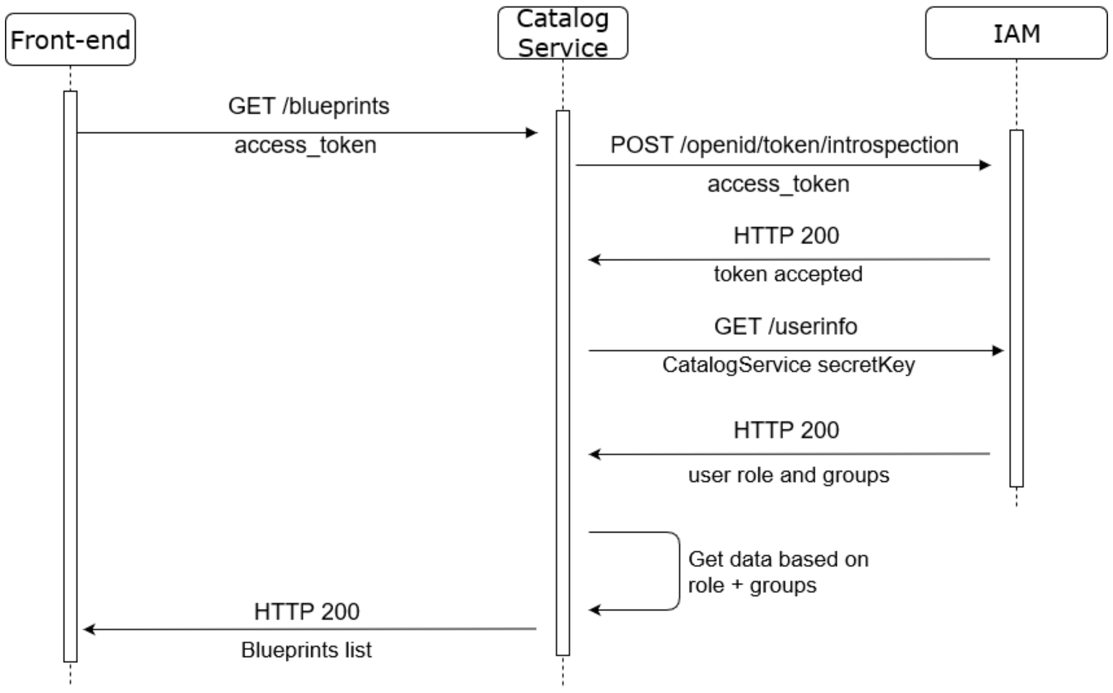


Figure 4: Example of service using authorization.

1.1.2 Experiment Lifecycle Manager

The Experiment Lifecycle Manager (ELM) is the Portal backend component in charge of processing the requests to create new experiment instances or to perform actions related to existing experiment, offering a REST-based north-bound interface (NBI) that can be invoked by external REST clients to manage experiments. This REST API is typically used by the 5G EVE Portal GUI, which mediates between a 5G EVE user and the 5G EVE platform for all the procedures related to the creation, management and monitoring of experiments. However, the system allows also external clients (e.g. platforms developed in the context of ICT-19 projects) to directly access the ELM functionalities via REST API. All the requests received at the ELM NBI are authenticated and authorized through the RBAC component (see section 1.1.1), according to the actions permitted for the different 5G EVE roles, as defined in D4.1 [1].

Table 3 reports all the actions that can be performed through the Experiment Lifecycle Manager, providing details about the 5G EVE roles that are allowed to request the related functionality.

Table 3. Actions supported by the Experiment Lifecycle Manager

Functionality	Enabled 5G EVE role	Notes
Creation of a new experiment.	Experimenter	
Approval or refusal of an experiment.	Site Manager	Only for the experiments on the managed site.
Change of proposed timeslot for the execution of an experiment.	Experimenter Site Manager	Only for experiments owned by the experimenters or located on the managed site.
Notification of readiness for an experiment environment.	Site Manager	Only for the experiments on the managed site.
Instantiation of an experiment.	Experimenter	Only for owned experiments.
Execution of an experiment.	Experimenter	Only for owned experiments.
Abortion of an experiment.	Experimenter	Only for owned experiments. Not supported in first release.
Termination of an experiment.	Experimenter	Only for owned experiments.
Request of information about an experiment.	Experimenter Site Manager	Only for experiments owned by the experimenters or located on the managed site.
Removal of an experiment.	Experimenter	Only for owned experiments.

The ELM manages internally the Finite State Machines (FSM) for all the active experiments, it maintains records with the information related to the experiments and their running executions. It coordinates the actions of the other functional elements in the Portal backend and in the I/W framework that are involved in the instantiation, monitoring, execution and validation of experiments. In particular, the ELM interacts with the following entities:

- RBAC, for request authentication and authorization purposes
- 5G EVE Portal Catalogue, for retrieving the blueprints and descriptors of the requested experiments. This information is used to identify the NFV network services to be instantiated on the 5G EVE platform, the monitoring metrics and KPIs to be collected through the DCS and the test cases to be executed for a given experiment.
- Ticketing tool, for generating and issuing tickets related to the scheduling and preparation of the virtual environment.
- Data Collection and Storage, to register and unregister all the metrics, KPIs and results associated to an experiment.
- Multi-Site Network Orchestration (MSNO) at the I/W framework, to request the instantiation and termination of the NFV network services where the experiment will be executed.
- Experiment Execution Manager (EEM) to request the executions of the experiment and collect information about their results.

The FSM representing the evolution of the lifecycle of an experiment is shown in Figure 5. Whenever the ELM receives a request to create a new experiment, it instantiates a new FSM to manage its lifecycle, it creates a new entry in its internal records, and it interacts with the ticketing tool to notify the target site manager. The FSM is initialized with the "Scheduling" state and all the descriptors and blueprints related to the experiment request are gathered from the Portal catalogue. The timeslot proposed by the experimenter for the execution of the experiment can be negotiated with the site manager; during this negotiation period the FSM remains in the "Scheduling" state. If the site manager accepts the proposed scheduling, the FSM moves to the "Accepted" state; in case of refusal, the FSM move to the "Refused" state and the experiment cannot proceed.

An experiment in "Accepted" state may need a manual configuration of the environment where it will be executed at the target site. When this environment is ready, the site manager will need to send a notification (through the Portal GUI or the REST API of the ELM). As consequence, the ELM will interact with the Ticketing tool to notify the experimenter and the FSM will move to the "Ready" state. At this stage, the experimenter will be able to request the instantiation of the virtual environment to run the experiment. At the reception of this deployment request, the ELM translates the experiment descriptor into a suitable NFV network service and triggers an interaction with the I/W framework MSNO to request its instantiation. The FSM moves to the "Instantiating" state. In case of successful notification from the MSNO, the ELM performs a registration with the DCS related to the metrics, KPIs and results to be monitored and moves the FSM to the "Instantiated" state. In case of failure notification from the MSNO, the experiment moves to a final state "Failed".

Once the experiment environment is properly instantiated, the experimenter may request to run an "experiment execution", specifying the target service configuration and test case(s) to be executed. The execution request triggers an interaction with the EEM, and the FSM moves to the "Running execution" state. In this state there are a number of interactions between ELM and EEM, which allows the ELM to be informed about the progress of the given execution. When the EEM notifies the execution termination, the ELM retrieves the related results, stores them into the internal ELM database updating the record entry of the given experiment, and moves again the FSM to the "Instantiated" state, so that the experimenter can request to run another execution. When the experimenter has completed all his/her tests and invokes the experiment termination (or at the expiration of the scheduled time interval negotiated at the beginning for the experiment execution), the ELM sends a request to the MSNO for terminating the associated NFV network service and the FSM moves to the "Terminating" state. Finally, at the reception of the notification from the MSNO, the ELM removes the monitoring registrations and moves the experiment FSM to the final "Terminated" state. Record entries for experiments in the final states "Refused", "Failed" and "Terminated" can be also removed from the system through a deletion request.

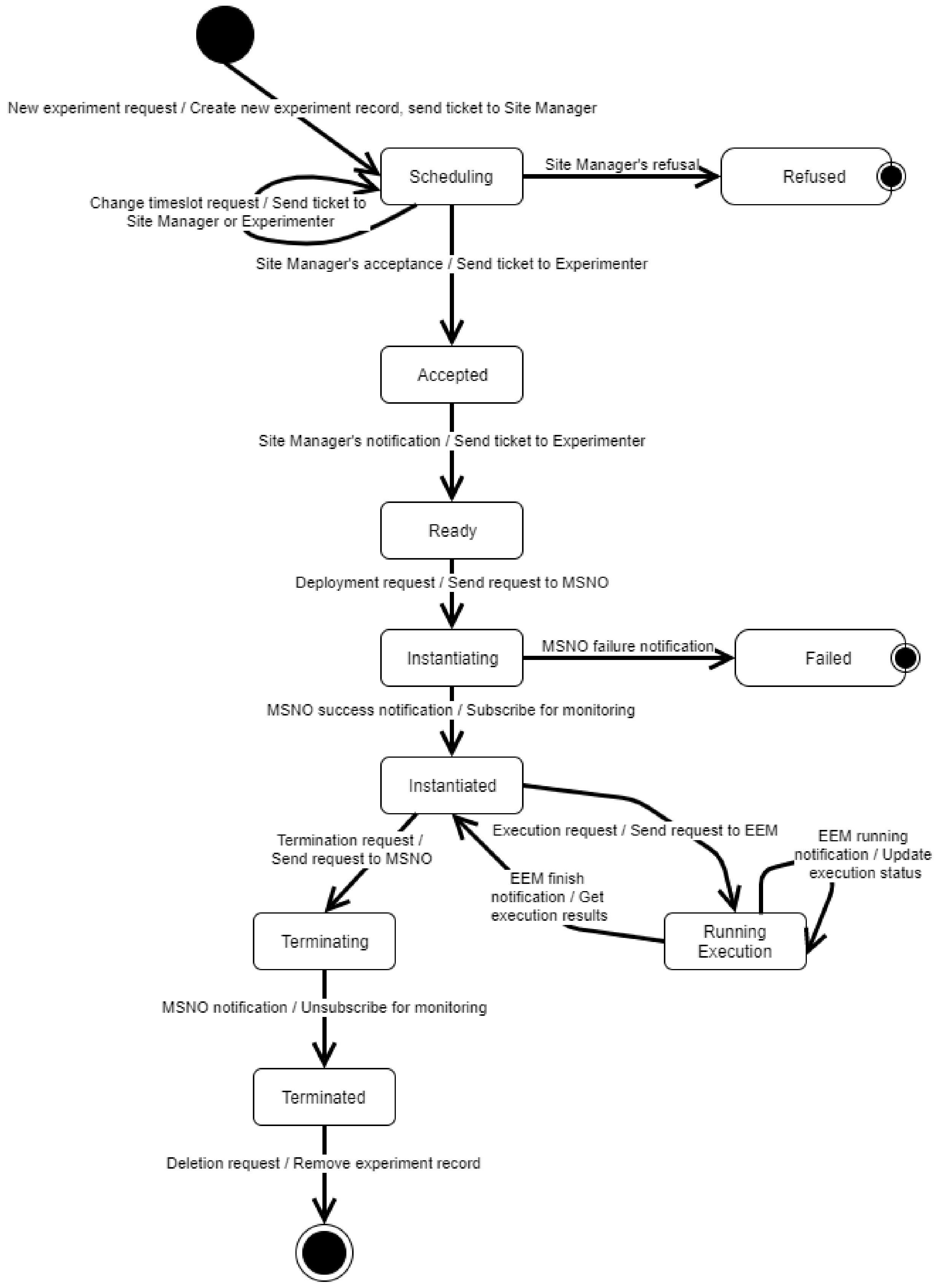

New experiment request / Create new experiment record, send ticket to Site Manager
Scheduling
Site Manager's refusal
Refused
Change timeslot request / Send ticket to Site Manager or Experimenter
Site Manager's acceptance / Send ticket to Experimenter
Accepted
Site Manager's notification / Send ticket to Experimenter
Ready
Deployment request / Send request to MSNO
Instantiating
MSNO failure notification
Failed
MSNO success notification / Subscribe for monitoring
Instantiated
Execution request / Send request to EEM
EEM running notification / Update execution status
Termination request / Send request to MSNO
EEM finish notification / Get execution results
Running Execution
Terminating
MSNO notification / Unsubscribe for monitoring
Terminated
Deletion request / Remove experiment record

Figure 5: Finite state machine for an experiment lifecycle

1.1.3 Data Collection and Storage

The **Data Collection and Storage** component (DCS) placed in the Portal backend brings together two of the three tools that belong to the **Experiment monitoring and Maintenance & Results Collection** toolchain, already presented in D4.1 [1]. These are the following:

- The **data collection, aggregation and pre-processing tool**, which collects the experiment results, metrics (both infrastructure and application metrics) and KPIs generated for a given experiment provided by the Data Collection Manager component from the I/W Framework.
- The **data indexing and storage tool**, which enables the capability of searching and filtering among all the data gathered by the data collection tool for obtaining the useful information that will be displayed afterwards in the Portal GUI, also providing data persistence.

As explained in D4.1 [1], Logstash and Elasticsearch, from the Elastic (ELK) Stack [3], are the two tools which will implement these functions, respectively. Then, the Data Collection and Storage component is highlighted in the following picture, which represents the Elastic Stack toolchain implemented in 5G EVE in combination with Beats and Kafka:

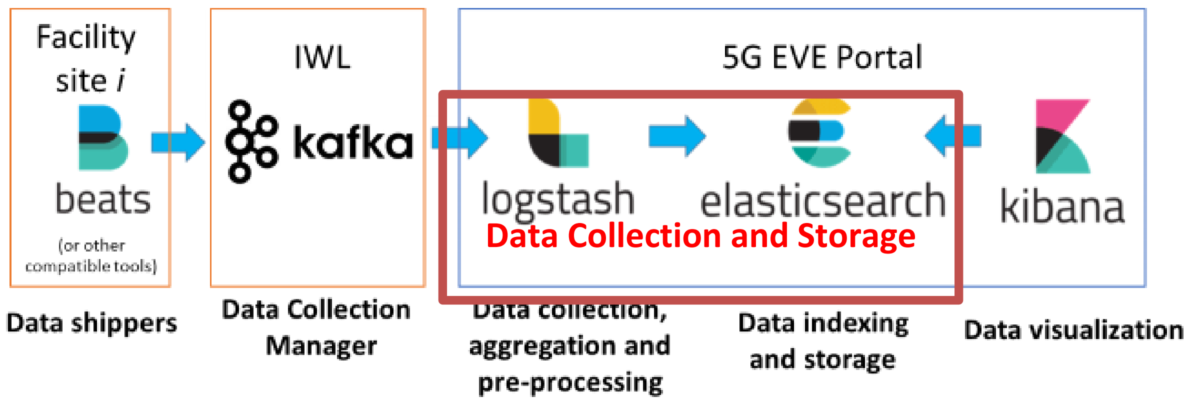

Facility site *i*
beats
(or other compatible tools)
Data shippers
IWL
kafka
Data Collection Manager
5G EVE Portal
logstash
elasticsearch
kibana
Data Collection and Storage
Data collection, aggregation and pre-processing
Data indexing and storage
Data visualization

Figure 6: Data Collection and Storage component representation in the Elastic Stack toolchain.

The authentication of users and the definition of roles and restrictions to only access the data that corresponds to each user, depending on the use case, is intended to be provided through the RBAC component in the Irbac interface (see 1.1.1), including the necessary open-source plugins in order to enable the Single Sign-On authentication and authorization mechanisms in the Elastic Stack. The plugins finally implemented will be described in detail in the next portal implementation deliverable.

The main functionalities that will be carried out by each component are the following:

- **Data collection, aggregation and pre-processing tool (Logstash):**
  - Ingest data securely from multiple input sources simultaneously.
  - Execute different transformations and enhancements to the collected data by using filters, which parse each event, identify named fields to build structure and transform them to converge on a common format for more powerful analysis and business value.
  - Ship the data to various supported output destinations (e.g. Elasticsearch).
  - Extend and improve the previous pipeline (ingest-filter-ship) with new plugins, which can be connected through specific APIs.
  - Guarantee at-least-once delivery for the data received with a persistent queue in case of failure, and also provide scalability to ingestion spikes without having to use an external queueing layer.
  - Monitor the status of the nodes and pipelines through the Data Visualization tool in the Portal GUI.
- **Data indexing and storage tool (Elasticsearch):**
  - Provide a RESTful search and analytics engine with one centralized data storage.

- Allow to perform and combine many types of searches (structured, unstructured, metric…), and also performing data aggregation in order to explore trends and patterns in the data.
- Leverage and access to all managed data at a very high speed thanks to the use of indexes for saving data.
- Scale horizontally if needed, going from prototype to production seamlessly by running this component on a single node the same way that in a cluster.
- Rank the search results based on a variety of factors (from term frequency or recency to popularity and beyond). Mix and match these along with functions to fine tune how the results show up to the experimenters.
- Detect failures to keep the deployed environment and the data safe and available with cross-cluster replication, using a secondary cluster as a hot backup.
- Allow to connect, build and maintain clients in many languages such as Java, Python, .NET, SQL and PHP through the usage of standard RESTful APIs and JSON.

1.1.4 File Storage

The File Storage module is intended to be used by other modules or by some actors to upload large files, similar to a cloud storage system. The main user of this module is the VNF Storage component at the Portal GUI layer, which is used by VNF providers to upload VNF packages, so the site managers selected in this process will receive a ticket requesting the onboarding of the selected packages in their sites. As commented before, the File Storage module, after a successful uploading of the file, has to create as many tickets as the sites selected by the VNF provider. These tickets should include the link to the uploaded package, so site managers can download such file.

Files uploaded to this module will be removed after a given time (to be still decided, although this information will be conveniently provided to the user) because of space constraints. Furthermore, this module must provide a configuration file to fix the maximum file size that the user may upload.

1.1.5 Catalogue Service

The Portal Catalogue is used to store all the blueprints and descriptors associated to the definition of a 5G EVE experiment. In particular, it manages the information elements for Vertical Services, experiment execution contexts, test cases and experiments. Its functionalities have been fully documented in deliverable D4.1 [1], section 5.

1.1.6 Ticketing System Backend

The Ticketing System Backend (TSB) component offers a service to other modules at the backend and to the Ticketing GUI module through the *Itsb* interface to create, comment and delete tickets. The main goal of this module is to notify events like errors, experiment requests, VNF onboarding requests, etc. to the corresponding actors who have to manage such events. The design of this module has been already presented in D4.1 [1], section 4.2.

1.2 Portal backend northbound interface

This subsection includes a detailed information of the REST API offered by the Portal backend layer. This definition is done component by component, starting with an overall view of all endpoints offered by each component. Then, each endpoint is totally specified with the request body, the response body, and the code after a successful transaction as well as possible error codes.

1.2.1 Role-Based Access Control Interface (Irbac)

The RBAC component allows requests to register a new user, create a new user session, refresh an access token, and close a user session. The REST API endpoints are defined in Table 4, having the details of each endpoint in the subsequent tables (Table 5 to Table 8).

Table 4. Role-Based Access Control (RBAC) REST API.

HTTP method	URI	Description
POST	/portal/rbac/register	Endpoint to create a new user at the portal
POST	/portal/rbac/login	Allows the creation of a new session for a specific user who is already registered at the portal.
POST	/portal/rbac/refreshtoken	Allows users to obtain a new access token once the one they have is expired
GET	/portal/rbac/logout	Closes a user session

Table 5. REST API - POST /portal/rbac/register

POST /portal/rbac/register			
	Parameter name	Parameter type	Description
Request Body (JSON)	email	String	Email address of the new user
	username	String	Username of the new user
	firstName	String	First Name of the new user
	lastName	String	Last Name of the new user
	password	String	Password
Response body			
Successful response HTTP code: 201 CREATED			
Error response HTTP code: 400 BAD REQUEST, 500 INTERNAL ERROR			

Table 6. REST API – POST /portal/rbac/login

POST /portal/rbac/login			
	Parameter name	Parameter type	Description
Request Body (JSON)	email	String	Email address of the new user
	password	String	Password
Response body	access_token	String	Access token to include in all the requests for resources
	Refresh_token	String	Refresh token that allows access token renewal
Successful response HTTP code: 200 OK			
Error response HTTP code: 400 BAD REQUEST, 500 INTERNAL ERROR			

Table 7. REST API – POST /portal/rbac/refreshtoken

POST /portal/rbac/refreshtoken			
	Parameter name	Parameter type	Description
Request Body (JSON)	refresh_token	String	Refresh token to generate a new access token
Response body	access_token	String	Access token to include in all the requests for resources
	refresh_token	String	Refresh token that allows access token renewal
Successful response HTTP code: 200 OK			
Error response HTTP code: 400 BAD REQUEST, 500 INTERNAL ERROR			

Table 8. REST API – GET /portal/rbac/logout

GET /portal/rbac/logout			
	Parameter name	Parameter type	Description

Authentication Headers	Authorization: bearer + access_token	String	Header that stores the access token of the user
Request Body (JSON)			
Response body	access_token	String	Access token to include in all the requests for resources
	refresh_token	String	Refresh token that allows access token renewal
Successful response HTTP code: 204 NO_CONTENT			
Error response HTTP code: 401 UNAUTHORIZED, 500 INTERNAL ERROR			

1.2.2 Experiment Lifecycle Manager Interface (Ielm)

The ELM REST API allows to request the creation, management and monitoring of experiments in the 5G EVE platform. The list of the ELM REST APIs is defined in Table 9. The following tables Table 10 – Table 17 specify the single REST APIs.

Table 9. Experiment Lifecycle Manager REST API

HTTP method	URI	Description
POST	/portal/elm/experiment	Creates a new experiment.
GET	/portal/elm/experiment<?[filter_parameters]>	Returns a list of experiments matching a given filter, i.e. based on the experiment identifier or the experiment descriptor identifier. The acceptable filter_parameters are the following: <Exp_ID; ExpD_ID>.
PUT	/portal/elm/experiment/{expId}/status	Changes the status of an experiment. The permitted changes are the following: - from scheduling to accepted - from scheduling to refused - from accepted to ready
PUT	/portal/elm/experiment/{expId}/timeslot	Changes the proposed timeslot for an experiment. The experiment must be in scheduling state.
POST	/portal/elm/experiment/{expId}/action/deploy	Deploys the virtual environment to run the experiment (i.e. instantiates the associated NFV network service). The experiment must be in ready state.
POST	/portal/elm/experiment/{expId}/action/execute	Executes of one or more tests for a given experiment. The experiment must be in instantiated state.
POST	/portal/elm/experiment/{expId}/action/terminate	Terminates the virtual environment where the experiment has been executed (i.e. terminates the associated NFV network service). The experiment must be in instantiated state.
DELETE	/portal/elm/experiment/{expId}	Removes an experiment and its record from the system. The experiment must be in refused, terminated or failed state.

Table 10. REST API – POST /portal/elm/experiment

POST /portal/elm/experiment			
	Parameter name	Parameter type	Description

Request body	experimentDescriptorId	String	ID of the experiment descriptor defining the characteristics of the experiment to be created.
	timeslot	ExecutionTimeSlot	Definition of the timeslot proposed for the execution of the experiment. Includes startTime and endTime.
	targetSites	List<EveSite>	List of sites where the experiment must be instantiated and executed.
Response body	experimentId	String	ID of the created experiment.
Successful response HTTP code: 201 CREATED			
Error response HTTP code: 400 BAD REQUEST, 403 FORBIDDEN, 404 NOT FOUND, 500 INTERNAL ERROR			

Table 11. REST API – GET /portal/elm/experiment<?[filter_parameters]>

GET /portal/elm/experiment<?[filter_parameters]>			
	Parameter name	Parameter type	Description
URI Query Parameters	expId	String	ID of the requested experiment.
	expDId	String	ID of the experiment descriptor associated to the requested experiments.
Response body	experiments	List<Experiment>	Information about the requested experiments, including their results if available.
Successful response HTTP code: 200 OK			
Error response HTTP code: 400 BAD REQUEST, 403 FORBIDDEN, 404 NOT FOUND, 500 INTERNAL ERROR			

Table 12. REST API – PUT /portal/elm/experiment/{expId}/status

PUT /portal/elm/experiment/{expId}/status			
	Parameter name	Parameter type	Description
Request body	status	Enum	New status of the experiment.
URI Variables	expId	String	ID of the experiment to be updated.
Response body	--	--	--
Successful response HTTP code: 202 ACCEPTED			
Error response HTTP code: 400 BAD REQUEST, 403 FORBIDDEN, 404 NOT FOUND, 409 CONFLICT, 500 INTERNAL ERROR			

Table 13. REST API – PUT /portal/elm/experiment/{expId}/timeslot

PUT /portal/elm/experiment/{expId}/timeslot			
	Parameter name	Parameter type	Description
Request body	timeslot	ExecutionTimeSlot	Definition of the new timeslot proposed for the execution of the experiment. Includes startTime and endTime.
URI Variables	expId	String	ID of the experiment to be updated.
Response body	--	--	--
Successful response HTTP code: 202 ACCEPTED			
Error response HTTP code: 400 BAD REQUEST, 403 FORBIDDEN, 404 NOT FOUND, 409 CONFLICT, 500 INTERNAL ERROR			

Table 14. REST API – POST /portal/elm/experiment/{expId}/action/deploy

POST /portal/elm/experiment/{expId}/action/deploy			
	Parameter name	Parameter type	Description
Request body	--	--	--
URI Variables	expId	String	ID of the experiment to be de-ployed.
Response body	--	--	--
Successful response HTTP code: 202 ACCEPTED			
Error response HTTP code: 400 BAD REQUEST, 403 FORBIDDEN, 404 NOT FOUND, 409 CON-FLICT, 500 INTERNAL ERROR			

Table 15. REST API – POST /portal/elm/experiment/{expId}/action/execute

POST /portal/elm/experiment/{expId}/action/execute			
	Parameter name	Parameter type	Description
Request body	testCaseDescriptor-Configuration	Map<String, Map<String, String>>	Specification of the test cases to be executed and their user config-uration. The key of the outer map is the testCaseDescriptorId. The value is an inner map with the user parameters for that test cases. The test cases and the user param-eters specified in the request over-rides the ones defined in the ex-periment descriptor. If this field is empty, all the test cases defined in the experiment descriptor are exe-cuted by default.
URI Variables	expId	String	ID of the experiment to be exe-cuted.
Response body	--	--	--
Successful response HTTP code: 202 ACCEPTED			
Error response HTTP code: 400 BAD REQUEST, 403 FORBIDDEN, 404 NOT FOUND, 409 CON-FLICT, 500 INTERNAL ERROR			

Table 16. REST API – POST /portal/elm/experiment/{expId}/action/terminate

POST /portal/elm/experiment/{expId}/action/terminate			
	Parameter name	Parameter type	Description
Request body	--	--	--
URI Variables	expId	String	ID of the experiment to be termi-nated.
Response body	--	--	--
Successful response HTTP code: 202 ACCEPTED			
Error response HTTP code: 400 BAD REQUEST, 403 FORBIDDEN, 404 NOT FOUND, 409 CON-FLICT, 500 INTERNAL ERROR			

Table 17. REST API – DELETE /portal/elm/experiment/{expId}

DELETE /portal/elm/experiment/{expId}			
	Parameter name	Parameter type	Description
URI Variables	expId	String	ID of the experiment to be de-leted.
Response body	--	--	--

Successful response HTTP code: 204 NO CONTENT
Error response HTTP code: 400 BAD REQUEST, 403 FORBIDDEN, 404 NOT FOUND, 409 CONFLICT, 500 INTERNAL ERROR

1.2.3 Data Collection and Storage Interface (Idcs)

As presented in 5GEVE deliverable D4.1 [1], the Elastic Stack will be used in the 5G EVE project to implement the DCS and the Data Visualization components. As a result, the REST APIs of all the components of the Elastic Stack (Logstash, Elasticsearch and Kibana) are available and can be used for a given experiment.

In the case of the DCS REST API, it will be defined and described based on the Elasticsearch REST API, as it is the component which directly interacts with the Data Visualization (Kibana) component. This REST API is available in the Elastic official website[1]. Note that Elasticsearch (and also Logstash) can be managed directly through Kibana, but this REST API enables the possibility to configure and access Elasticsearch features directly, as long as the user has enough privileges to access to the selected resources.

Some of the most interesting functionalities for the 5G EVE project provided by Elasticsearch through its REST API are presented below. However, the final REST API definition will be specified in future implementation deliverables. Table 18 to Table 27 introduce the relevant REST APIs.

Table 18. Data Collection and Storage REST API

HTTP method	URI	Description
GET	/_ingest/pipeline/{pipeline}	Returns information about one or more ingest pipelines. This API returns a local reference of the pipeline.
PUT	/_ingest/pipeline/{pipeline}	Creates or updates an ingest pipeline. Changes made using this API take effect immediately.
DELETE	/_ingest/pipeline/{pipeline}	Deletes one or more existing ingest pipeline.
GET	/{index}	Returns information about one or more indexes.
PUT	/{index}	Creates a new index.
PUT	/{index}/_mapping	Adds new fields to an existing index or changes the search settings of existing fields.
POST	/{index}/_graph/explore	Extracts and summarizes information about the documents and terms in an Elasticsearch index. The easiest way to understand the behaviour of this API is to use the Graph UI to explore connections.
GET/POST	/{index}/_search	Returns search hits that match the query defined in the request.
DELETE	/{index}	Deletes an existing index.

Table 19. REST API – GET /_ingest/pipeline/{pipeline}

GET /_ingest/pipeline/{pipeline}			
	Parameter name	Parameter type	Description
Request body	--	--	--
URI Variables	pipeline	String	(Optional) Comma-separated list or wildcard expression of pipeline IDs used to limit the request.
Query Parameters	master_timeout	Time units	(Optional) Specifies the period of time to wait for a connection to

[1] https://www.elastic.co/guide/en/elasticsearch/reference/current/rest-apis.html

			the master node. If no response is received before the timeout expires, the request fails and returns an error. Defaults to 30s.
Response body	description	String	Description of the ingest pipeline.
	processors	List<Processor>	Array of processors used to pre-process documents before indexing. Processors are executed in the order provided.
	version	Integer	(Optional) Version number used by external systems to manage ingest pipelines. Versions are not used or validated by Elasticsearch; they are intended for external management only.
Successful response HTTP code: 200 OK.			
Error response HTTP code: 400 BAD REQUEST, 403 FORBIDDEN, 404 NOT FOUND, 500 INTERNAL ERROR.			

Table 20. REST API – PUT /_ingest/pipeline/{pipeline}

PUT /_ingest/pipeline/{pipeline}			
	Parameter name	Parameter type	Description
Request body	description	String	Description of the ingest pipeline.
	processors	List<Processor>	Array of processors used to pre-process documents before indexing. Processors are executed in the order provided.
	version	Integer	(Optional) Version number used by external systems to manage ingest pipelines. Versions are not used or validated by Elasticsearch; they are intended for external management only.
URI Variables	pipeline	String	ID of the ingest pipeline to create or update.
Query Parameters	master_timeout	Time units	(Optional) Specifies the period of time to wait for a connection to the master node. If no response is received before the timeout expires, the request fails and returns an error. Defaults to 30s.
Response body	description	String	Same as in request body.
	processors	List<Processor>	Same as in request body.
	version	Integer	Same as in request body.
Successful response HTTP code: 200 OK.			
Error response HTTP code: 400 BAD REQUEST, 403 FORBIDDEN, 404 NOT FOUND, 500 INTERNAL ERROR.			

Table 21. REST API – DELETE /_ingest/pipeline/{pipeline}

DELETE /_ingest/pipeline/{pipeline}			
	Parameter name	Parameter type	Description
Request body	--	--	--

URI Variables	pipeline	String	Pipeline ID or wildcard expression of pipeline IDs used to limit the request. To delete all ingest pipelines in a cluster, use a value of *.
Query Parameters	timeout	Time units	(Optional) Specifies the period of time to wait for a response. If no response is received before the timeout expires, the request fails and returns an error. Defaults to 30s.
	master_timeout	Time units	(Optional) Specifies the period of time to wait for a connection to the master node. If no response is received before the timeout expires, the request fails and returns an error. Defaults to 30s.
Response body	--	--	--
Successful response HTTP code: 200 OK.			
Error response HTTP code: 400 BAD REQUEST, 403 FORBIDDEN, 404 NOT FOUND, 500 INTERNAL ERROR.			

Table 22. REST API – GET /{index}

GET /{index}			
	Parameter name	Parameter type	Description
Request body	--	--	--
URI Variables	index	String	Comma-separated list or wildcard expression of index names used to limit the request. Use a value of _all to retrieve information for all indices in the cluster.
Query Parameters	allow_no_indices	Boolean	(Optional) If true, the request does not return an error if a wildcard expression or _all value retrieves only missing or closed indices. This parameter also applies to index aliases that point to a missing or closed index.
	expand_wildcards	String	(Optional) Controls what kind of indices that wildcard expressions can expand to. Valid values are: all: Expand to open and closed indices. open (default): Expand only to open indices. closed: Expand only to closed indices. none: Wildcard expressions are not accepted.
	flat_settings	Boolean	(Optional) If true, returns settings in flat format. Defaults to false.

	include_defaults	String	(Optional) If true, return all default settings in the response. Defaults to false.
	ignore_unavailable	Boolean	(Optional) If true, missing or closed indices are not included in the response. Defaults to false.
	local	Boolean	(Optional) If true, the request retrieves information from the local node only. Defaults to false, which means information is retrieved from the master node.
	master_timeout	Time units	(Optional) Specifies the period of time to wait for a connection to the master node. If no response is received before the timeout expires, the request fails and returns an error. Defaults to 30s.
Response body	index	Index	Information about the index.
Successful response HTTP code: 200 OK.			
Error response HTTP code: 400 BAD REQUEST, 403 FORBIDDEN, 404 NOT FOUND, 500 INTERNAL ERROR.			

Table 23. REST API – PUT /{index}

PUT /{index}			
	Parameter name	Parameter type	Description
Request body	aliases	Alias	(Optional) Index aliases which include the index.
	mappings	Mapping	(Optional) Mapping for fields in the index. If specified, this mapping can include: Field names, Field datatypes and Mapping parameters.
	settings	IndexSetting	(Optional) Configuration options for the index.
URI Variables	index	String	Name of the index you wish to create.
Query Parameters	wait_for_active_shards	String	(Optional) The number of shard copies that must be active before proceeding with the operation. Set to all or any positive integer up to the total number of shares in the index (number_of_replicas+1). Default: 1, the primary shard.
	timeout	Time units	(Optional) Specifies the period of time to wait for a response. If no response is received before the timeout expires, the request fails and returns an error. Defaults to 30s.
	master_timeout	Time units	(Optional) Specifies the period of time to wait for a connection to the master node. If no response is received before the timeout

			expires, the request fails and returns an error. Defaults to 30s.
Response body	--	--	--
Successful response HTTP code: 200 OK.			
Error response HTTP code: 400 BAD REQUEST, 403 FORBIDDEN, 404 NOT FOUND, 500 INTERNAL ERROR.			

Table 24. REST API – PUT /{index}/_mapping

PUT /{index}/_mapping			
	Parameter name	Parameter type	Description
Request body	properties	Mapping	Mapping for a field. For new fields, this mapping can include: Field name, Field datatype and Mapping parameters.
URI Variables	index	String	Comma-separated list or wildcard expression of index names used to limit the request. To update the mapping of all indices, omit this parameter or use a value of _all.
Query Parameters	allow_no_indices	Boolean	(Optional) If true, the request does not return an error if a wildcard expression or _all value retrieves only missing or closed indices. This parameter also applies to index aliases that point to a missing or closed index.
	expand_wildcards	String	(Optional) Controls what kind of indices that wildcard expressions can expand to. Valid values are: all: Expand to open and closed indices. open (default): Expand only to open indices. closed: Expand only to closed indices. none: Wildcard expressions are not accepted.
	ignore_unavailable	Boolean	(Optional) If true, missing or closed indices are not included in the response. Defaults to false.
	timeout	Time units	(Optional) Specifies the period of time to wait for a response. If no response is received before the timeout expires, the request fails and returns an error. Defaults to 30s.
	master_timeout	Time units	(Optional) Specifies the period of time to wait for a connection to the master node. If no response is received before the timeout expires, the request fails and returns an error. Defaults to 30s.

Response body	--	--	--
Successful response HTTP code: 200 OK.			
Error response HTTP code: 400 BAD REQUEST, 403 FORBIDDEN, 404 NOT FOUND, 500 INTERNAL ERROR.			

Table 25. REST API – PUT /{index}/_graph/explore

PUT /{index}/_graph/explore			
	Parameter name	Parameter type	Description
Request body	query	Query	A seed query that identifies the documents of interest. Can be any valid Elasticsearch query.
	vertices	List<Vertex>	Specifies or more fields that contain the terms you want to include in the graph as vertices.
	connections	List<Connection>	Specifies or more fields from which you want to extract terms that are associated with the specified vertices.
	controls	List<Control>	Direct the Graph API how to build the graph.
URI Variables	index	String	Index to which you want to apply the Graph explore operation.
Query Parameters	--	--	--
Response body	vertices	List<Vertex>	An array of all of the vertices that were discovered. A vertex is an indexed term, so the field and term value are provided. The weight attribute specifies a significance score. The depth attribute specifies the hop-level at which the term was first encountered.
	connections	List<Connection>	The connections between the vertices in the array. The source and target properties are indexed into the vertices array and indicate which vertex term led to the other as part of exploration. The doc_count value indicates how many documents in the sample set contain this pairing of terms (this is not a global count for all documents in the index).
Successful response HTTP code: 200 OK.			
Error response HTTP code: 400 BAD REQUEST, 403 FORBIDDEN, 404 NOT FOUND, 500 INTERNAL ERROR.			

Table 26. REST API – GET/POST /{index}/_search

GET/POST /{index}/_search			
	Parameter name	Parameter type	Description
Request body	query	Query	(Optional) Defines the search definition using the Query DSL.

URI Variables	index	String	Comma-separated list or wildcard expression of index names used to limit the request.
Query Parameters[2]	*(see footnote 2)*	*(see footnote 2)*	*(see footnote)*
Response body	took	Integer	Milliseconds it took Elasticsearch to execute the request.
	timed_out	Boolean	If true, the request timed out before completion; returned results may be partial or empty.
	_shards	Object	Object containing a count of shards used for the request.
	hits	Object	Contains returned documents and metadata.
Successful response HTTP code: 200 OK.			
Error response HTTP code: 400 BAD REQUEST, 403 FORBIDDEN, 404 NOT FOUND, 500 INTERNAL ERROR.			

Table 27. REST API – DELETE /{index}

DELETE /{index}/			
	Parameter name	Parameter type	Description
Request body	--	--	--
URI Variables	index	String	Comma-separated list or wildcard expression of indices to delete. In this parameter, wildcard expressions match only open, concrete indices. You cannot delete an index using an alias.
Query Parameters	allow_no_indices	Boolean	(Optional) If true, the request does not return an error if a wildcard expression or _all value retrieves only missing or closed indices. This parameter also applies to index aliases that point to a missing or closed index.
	expand_wildcards	String	(Optional) Controls what kind of indices that wildcard expressions can expand to. Valid values are: all: Expand to open and closed indices. open (default): Expand only to open indices. closed: Expand only to closed indices. none: Wildcard expressions are not accepted.

[2] This request manages a lot of Query parameters and have been omitted for better readability of the document. They can be found in the following link: https://www.elastic.co/guide/en/elasticsearch/reference/current/search-search.html#search-search-api-request-body

	ignore_unavailable	Boolean	(Optional) If true, missing or closed indices are not included in the response. Defaults to false.
	timeout	Time units	(Optional) Specifies the period of time to wait for a response. If no response is received before the timeout expires, the request fails and returns an error. Defaults to 30s.
	master_timeout	Time units	(Optional) Specifies the period of time to wait for a connection to the master node. If no response is received before the timeout expires, the request fails and returns an error. Defaults to 30s.
Response body	--	--	--
Successful response HTTP code: 200 OK.			
Error response HTTP code: 400 BAD REQUEST, 403 FORBIDDEN, 404 NOT FOUND, 500 INTERNAL ERROR.			

1.2.4 File Storage Interface (Ifs)

This component is mainly used to exchange VNF packages between VNF providers and site managers to request the onboarding of a VNF package. Table 28 to Table 30 introduce the relevant REST API end-points.

Table 28. File Storage REST API

HTTP method	URI	Description
POST	/portal/fs/upload	Used to upload a single *zip* file and a list of one or more sites that have to be notified.
GET	/portal/fs/download/{fileUuid}	A user requests to download the file identified with *UUID*.

Table 29. REST API - POST /portal/fs/upload

POST /portal/fs/upload			
	Parameter name	Parameter type	Description
Request body	dzuuid	String	UUID of the uploaded file
	dzchunkindex	Integer	Index of the chunk being transmitted.
	dztotalfilesize	Integer	Total size of the file.
	dzchunksize	Integer	Size of the chunk being transmitted.
	dztotalchunkcount	Integer	Total number of chunks of the file.
	dzchunkbyteoffset	Integer	Offset of the chunk in the file.
	List<site>	String	Sites to be notified about the current uploading.
URI Variables	--	--	--
Response body	--	--	--
Successful response HTTP code: 202 ACCEPTED			
Error response HTTP code: 400 BAD REQUEST, 403 FORBIDDEN, 404 NOT FOUND, 409 CONFLICT, 500 INTERNAL ERROR			

Table 30. REST API - GET /portal/fs/download/{fileUuid}

GET /portal/fs/download/{fileUuid}			
	Parameter name	Parameter type	Description
Request body	-	-	-
URI Variables	Uuid	String	UUID of the file to be down-loaded
Response body	File	Base64	The requested file
Successful response HTTP code: 200 OK			
Error response HTTP code: 400 BAD REQUEST, 403 FORBIDDEN, 404 NOT FOUND, 409 CON-FLICT, 500 INTERNAL ERROR			

1.2.5 Catalogue Service Interface (Ics)

The Portal Catalogue REST APIs allow to create, retrieve, and remove blueprints and descriptors for vertical services, experiment execution contexts, test cases, and experiments. Their full specification is available in the 5G EVE deliverable D4.1 [1], section 5.3.1.

1.2.6 Ticketing System Backend Interface (Itsb)

Similar to other modules, the TSB will be offered by using an open source project, in this case Bugzilla. The REST API provided by Bugzilla can be found in their website[3]. Some examples of the Bugzilla REST API are shown in Table 31 to Table 34.

Table 31. Ticketing System backend REST API

HTTP method	URI	Description
POST	/portal/bugzilla/rest/bug	Create a new bug
PUT	/portal/bugzilla/rest/bug/{Id}	Update an existing bug
GET	/portal/bugzilla/rest/bug/{Id}	Gets information about a particular bug

Table 32. REST API - POST /portal/bugzilla/rest/bug

POST /portal/bugzilla/rest/bug			
	Parameter name	Parameter type	Description
Request body	Product	String	The name of the product the bug is being filed against.
	Component	String	The name of a component in the product above.
	Summary	String	A brief description of the bug be-ing filed
	Version	String	A version of the product above; the version the bug was found in
	Description	String	(defaulted) The initial description for this bug. Some Bugzilla instal-lations require this to not be blank
	Op_sys	String	(defaulted) The operating system the bug was discovered on
	Priority	String	(defaulted) What order the bug will be fixed in by the developer,

[3] https://wiki.mozilla.org/Bugzilla:REST_API

			compared to the developer's other bugs
URI Variables	--	--	--
Response body	Id	Integer	This is the ID of the newly-filled bug.
Successful response HTTP code: 202 ACCEPTED			
Error response HTTP code: 51 INVALID OBJECT, 103 INVALID ALIAS, 104 INVALID FIELD, 105 INVALID COMPONENT, 106 INVALID PRODUCT, 107 INVALID SUMMARY, 504 INVALID USER			

Table 33. REST API - PUT /portal/bugzilla/rest/bug/{Id}

PUT /portal/bugzilla/rest/bug/{Id}			
	Parameter name	Parameter type	Description
Request body	Product	String	The name of the product the bug is being filed against.
	Component	String	The name of a component in the product above.
	Summary	String	A brief description of the bug being filed
	Version	String	A version of the product above; the version the bug was found in
	Description	String	(defaulted) The initial descrip-tion for this bug. Some Bugzilla installations require this to not be blank
	Op_sys	String	(defaulted) The operating sys-tem the bug was discovered on
	Priority	String	(defaulted) What order the bug will be fixed in by the devel-oper, compared to the develop-er's other bugs
URI Variables	--	--	--
Response body	Id	Integer	This is the ID of the newly-filled bug.
Successful response HTTP code: 202 ACCEPTED			
Error response HTTP code: 51 INVALID OBJECT, 103 INVALID ALIAS, 104 INVALID FIELD, 105 INVALID COMPONENT, 106 INVALID PRODUCT, 107 INVALID SUMMARY, 504 INVALID USER			

Table 34. REST API - GET /portal/bugzilla/rest/bug/{Id}

GET /portal/bugzilla/rest/bug/{Id}			
	Parameter name	Parameter type	Description
Request body	--	--	--
URI Variables	Id	String	The ID of the requested bug
Response body	Actual_time	Double	The total number of hours that this bug has taken so far. If you are not in the time-tracking group,

			this field will not be included in the return value
	Alias	Array	The unique aliases of this bug. An empty array will be returned if this bug has no aliases
	Assigned_to	String	The login name of the user to whom the bug is assigned
	Creator	String	The login name of the person who filed this bug (the reporter)
	Status	String	The current status of the bug
	…	…	…
URI Variables	--	--	--
Response body	Id	Integer	This is the ID of the newly-filled bug.
Successful response HTTP code: 202 ACCEPTED			
Error response HTTP code: 101 INVALID BUG ALIAS, 101 INVALID BUG ID, 102 ACCESS DENIED			

1.3 Portal backend southbound interface

The Southbound Interface (SBI) of the portal backend is responsible of the communication between the different components of the Interworking framework. The SBI implements a REST client to consume the services defined by the following components of the Interworking framework, and highlighted in Figure 7:

- Data Collection Manager (DCM)
- Runtime Configurator
- Multi-site Network Service Orchestrator
- Multi-site Catalogue/Inventory

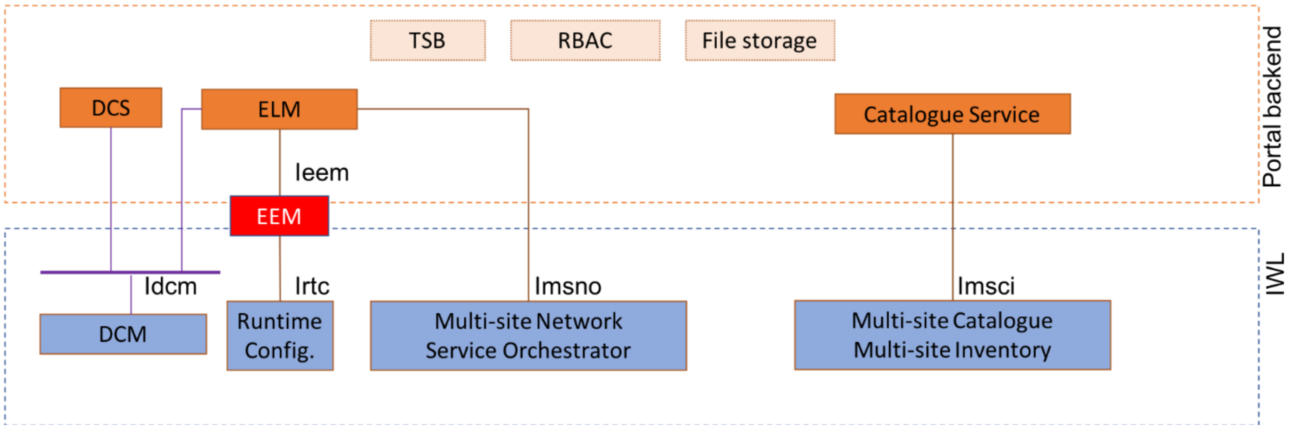


Figure 7: Simplified architecture of Southbound Interface of Portal backend and I/W Layer

1.3.1 DCM Interface (Idcm)

The Data Collection Manager of the Interworking Layer exposes the Idcm interface[4] to the DCS and ELCM in order to enable the interaction between these components through the publish-subscribe queue. This interface can be implemented with two different technologies:

- **Kafka native protocol:** this is the case for the interconnection between the DCM and the DCS, as the DCS will use both Logstash and Python clients for deploying the subscribers to the different topics that may be present for each experiment.
- **REST API:** built for components that are not compatible with the Kafka native protocol, e.g. the ELCM. In this case, the three main operations provided by the DCM (subscribe, publish and unsubscribe note that the delivery operation is performed automatically) are exposed through a REST API.

The different interactions between components are described in D4.1 [1] with the different phases that belong to the experiment monitoring and performance analysis: subscription phase, monitoring and data collection phase, and withdrawal phase.

In order to achieve the process automation in all these phases, the DCM OpenAPI specification has been slightly modified, as it was directly based on the Kafka Confluent REST API, but it has been detected that it is not enough for the workflows that are intended to be executed. As a result, the server which holds the DCM will also include a REST Proxy, which receives the requests from external components and redirects them to Kafka in order to execute properly the three operations aforementioned. This new OpenAPI specification is presented below and will be extended in the future D3.4 related to the update of the I/W Layer implementation, including all the feedback obtained during the implementation phase that will be performed in the following months.

Table 35. Updated Data Collection Manager OpenAPI specification.

HTTP method	URI	Description
POST	/dcm/subscribe	Subscribe to the given list of topics. This operation also creates the topics in Kafka if they do not already exist (subscribe operation).
POST	/dcm/publish/{topicName}	Post messages to a given topic (publish operation).
DELETE	/dcm/unsubscribe	Unsubscribe to the given list of topics. This operation also deletes the topics in Kafka if it has not been done before (unsubscribe operation).

Table 36. Data Collection Manager REST API – POST /dcm/subscribe

POST /dcm/subscribe			
	Parameter name	Parameter type	Description
Request body	topics	List<String>	Array of topics, separated by commas. They can be topics related to signalling operations, infrastructure and application metrics, KPIs and results (if needed).
	expId	String	Experiment ID.
URI Variables	--	--	--
Query Parameters	--	--	--

[4] This interface is also exposed to the Results Analysis and Validation module, which is currently defined in WP5 and is out of the scope of this deliverable.

Response body	--	--	--
Successful response HTTP code: 201 Accepted.			
Error response HTTP code: 400 BAD REQUEST, 403 FORBIDDEN, 404 NOT FOUND, 500 INTERNAL ERROR.			

Table 37. Data Collection Manager REST API – POST /dcm/publish/{topicName}

POST /dcm/publish/{topicName}			
	Parameter name	Parameter type	Description
Request body	record	List<value>	"value" data is provided to the components subscribed to topicName. The content of this "value" parameter depends on the publisher. In case of VNFs/PNFs, this must contain, at least, the metric name and its value. For the ELCM, which interacts with the DCM for delivering the topics to be (un)subscribed, the parameters included within "value" must be the following: "topic" (String): topic to be (un)subscribed. It can be a topic related infrastructure and application metrics, KPIs and results (if needed). "expId" (String): experiment ID. "action" (String): can be "subscribe" or "unsubscribe", depending on the phase in which this operation is executed.
URI Variables	topicName	String	Topic name.
Query Parameters	--	--	--
Response body	--	--	--
Successful response HTTP code: 201 Accepted.			
Error response HTTP code: 400 BAD REQUEST, 403 FORBIDDEN, 404 NOT FOUND, 500 INTERNAL ERROR.			

Table 38. Data Collection Manager REST API – DELETE /dcm/unsubscribe

DELETE /dcm/unsubscribe			
	Parameter name	Parameter type	Description
Request body	topics	List<String>	Array of topics, separated by commas. They can be topics related to signalling operations, infrastructure and application metrics, KPIs and results (if needed).
	expId	String	Experiment ID.
URI Variables	--	--	--
Query Parameters	--	--	--
Response body	--	--	--
Successful response HTTP code: 201 Accepted.			

Error response HTTP code: 400 BAD REQUEST, 403 FORBIDDEN, 404 NOT FOUND, 500 INTER-NAL ERROR.

1.3.2 Multi-site Network Service Orchestrator Interface (Imnso)

The Experiment Lifecycle Manager (ELCM) implements REST client that is able to request resources defined in MNSO. The ELCM provides ability to request Lifecycle Management operations defined by MNSO and executed through multiple site orchestrators.

The API published by the 5G EVE project for the Multi-Site Network Orchestrator[5] contains full ETSI NFV SOL 005 Network Service LCM API (defined in standard [5]). Table 39 highlights the available method at the MNSO level, which the ELCM are able to request.

Table 39. Experiment Lifecycle Manager to Multi-site Network Service Orchestrator API specification.

HTTP METHOD	URI	Description
POST	/nslcm/v1/ns_in-stances	Create a new Network Service in the MSNO.
GET	/nslcm/v1/ns_in-stances	Returns the information of all NS available at MSNO/MSI
POST	/nslcm/v1/ns_in-stances/{nsIn-stanceId}/instantiate	Instantiate a NS in the local NFV-O
GET	/nslcm/v1/ns_in-stances/{nsIn-stanceId}	Returns the information of a NS
DELETE	/nslcm/v1/ns_in-stances/{nsIn-stanceId}	Deletes a NS from MSNO
POST	/ns_instances/{nsIn-stanceId}/terminate	Terminate NS instance from MNSO

1.3.3 Multi-Site Service Inventory/Service Catalogue Interface (Imsci)

The interface between the Portal Catalogue and the I/W Framework Catalogue is used to on-board the Network Service Descriptors (NSD) associated to the blueprints and to retrieve information about the NSDs and the VNF packages available in the 5G EVE sites. This interface is fully documented in D4.1 [1], section 5.3.2.

1.3.4 Experiment Execution Manager Interface (Ieem)

The interface between the ELM and the EEM is used to create and to run experiment executions, as well as to receive asynchronous notifications about their progress and to query information about their results. This inter-face is fully documented in D5.2 [6], section 3.3.2.

1.4 Portal GUI

This subsection introduces the main components defined at the graphical user interface (GUI) layer, where the user can exploit the services provided by the Portal backend.

[5] https://github.com/5GEVE/OpenAPI/v0.2/MSN

1.4.1 VNF Storage

A VNF provider may access the 5G EVE graphical interface where he/she can select one local file to be uploaded to the portal and one or more sites to be contacted, using a multiple-choice list. After a successful uploading, the system creates as many tickets as sites selected by the user. The rest of the process can be followed using the ticketing service.

1.4.2 Sign-up/Login

The Sign-up and login components allow the creation of new users together with a plain and simple way of authentication across multiple micro-services. Both components are simple to use and the users are guided to create a new account or perform a correct login to obtain the required access token for accessing all the functionalities provided by the Portal.

1.4.3 Experiment Blueprint Builder

The Experiment Blueprint Builder is a component of the Portal web GUI, it provides a simple wizard to guide the experiment developer in the definition of an Experiment Blueprint. This wizard is organized in steps where the user selects the components of the Experiment Blueprint, defines its parameters (e.g. metrics and KPIs) and on-boards the associated NSD and translation rules. It should be noted that initially, the NSD must be compiled manually by the user. Upcoming deliverable D4.3 will introduce a new tool to automate the generation of the NSD starting from the high-level attributes of the blueprint.

The steps of the wizard for building a new experiment blueprint are depicted in **Figure 8** to **Figure 14**. In particular, the experiment developer initially provides general information about the experiment blueprint, e.g. its name, version and a textual description (**Figure 8**). The second step (**Figure 9**) requires the selection of the site where to execute the experiment and the corresponding vertical service, this is followed by the selection of the context blueprint(s) to be used in the experiment (**Figure 10**). Both vertical service and context can be selected from the list of available blueprints in a specific site catalogue, proposed in the web GUI.

The next step (**Figure 11**) allows the developer to on-board the NSD of the service to be deployed on the 5G EVE site which runs the experiment. The NSD is provided in a json file and structured according to the ETSI NFV IFA 014 [4] format. After that, the experiment developer specifies how the network service needs to be instantiated for different configurations of the experiments. This is done through the specification of translation rules (**Figure 12**). This allows to match a set of service parameter ranges with the target triple <NSD; deployment flavour; instantiation level>.

Finally, the experiment developer species all the details related to the experiment execution and validation. This is done defining the metrics and KPIs to be collected (**Figure 13**) and the list of Test Cases to be executed (**Figure 14**).

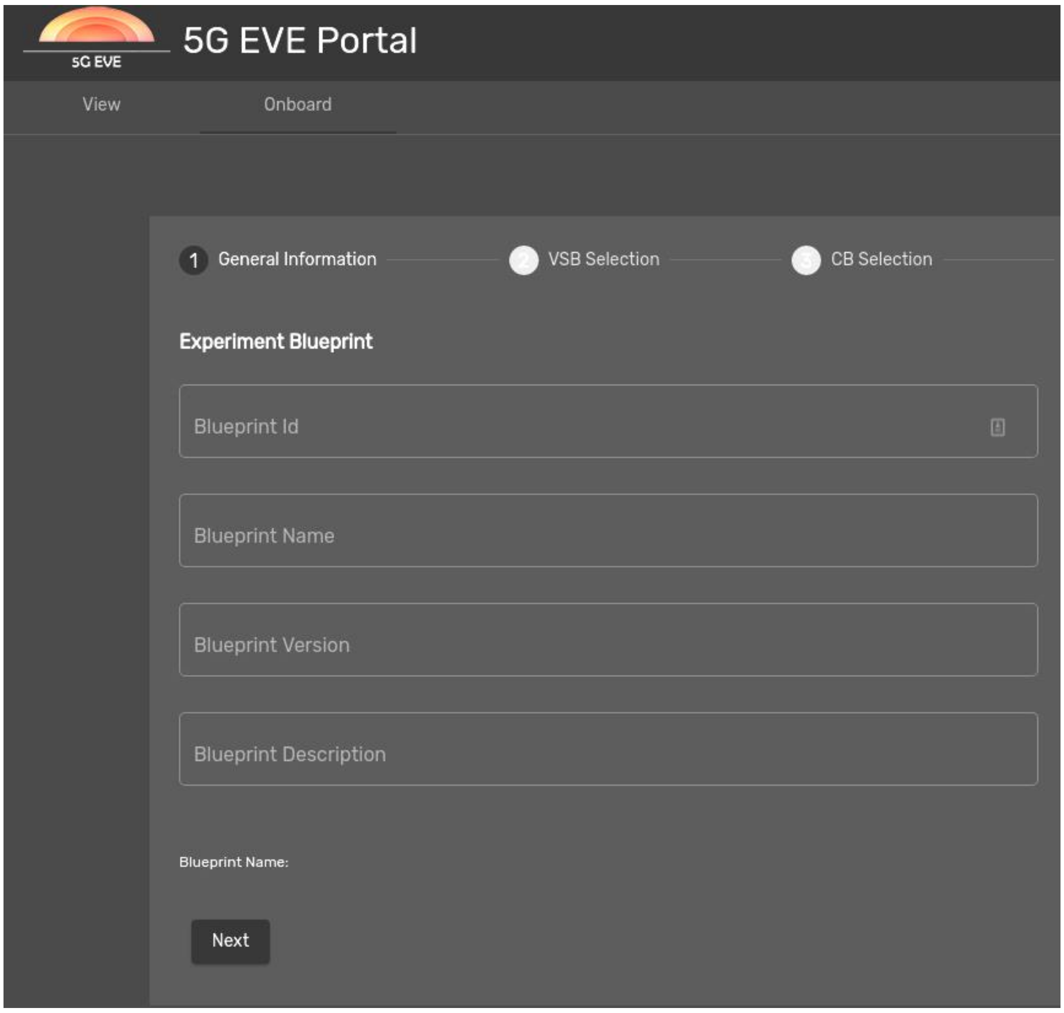


Figure 8: Experiment Blueprint Builder – Specification of general information

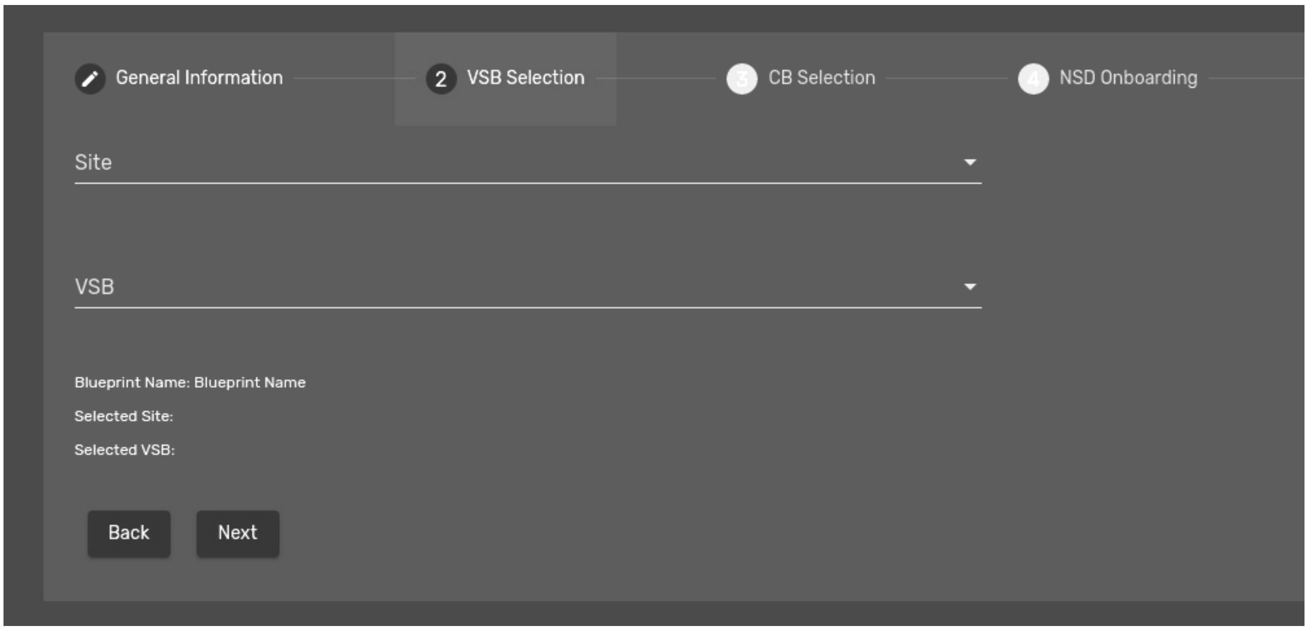
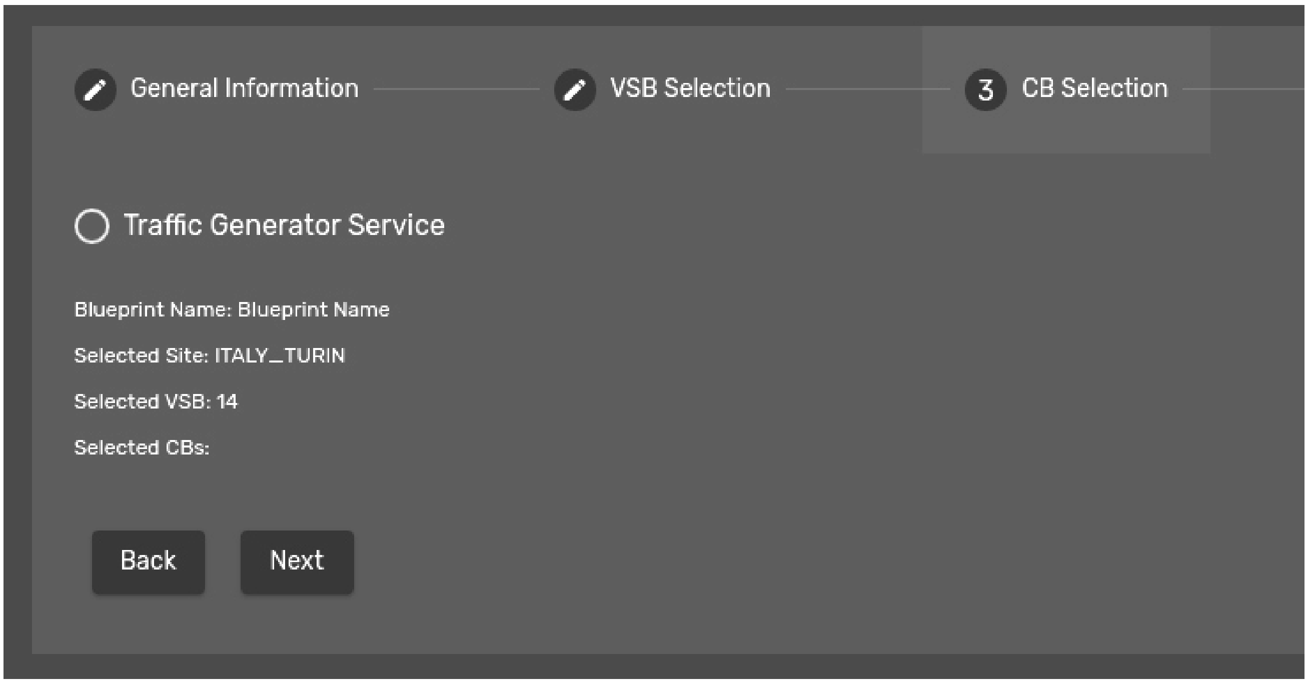


Figure 9: Experiment Blueprint Builder – Selection of Vertical Service Blueprint



Traffic Generator Service

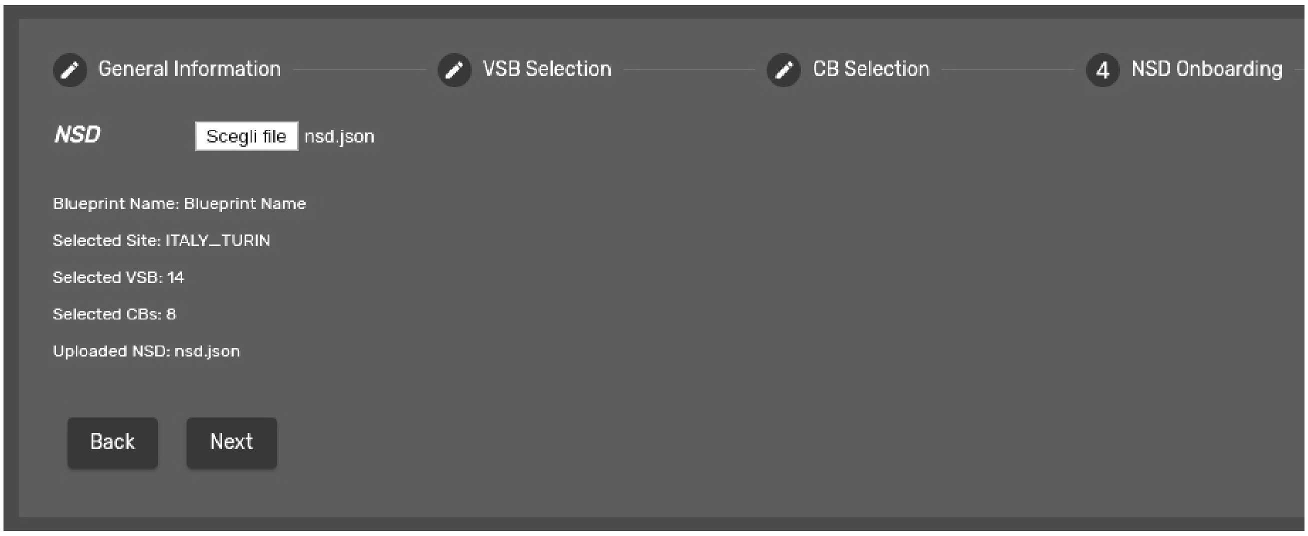
Blueprint Name: Blueprint Name

Selected Site: ITALY_TURIN

Selected VSB: 14

Selected CBs:

Figure 10: Experiment Blueprint Builder – Selection of Context Blueprint



NSD nsd.json

Blueprint Name: Blueprint Name

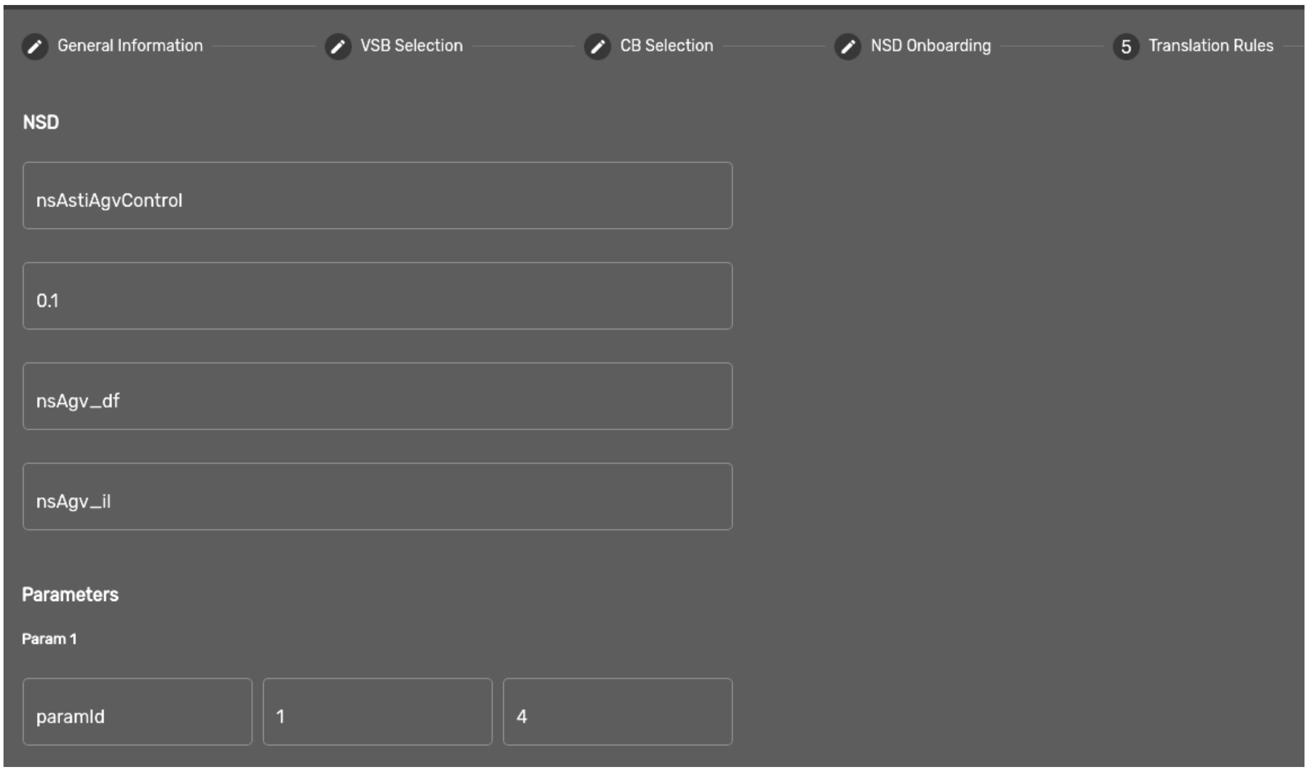
Selected Site: ITALY_TURIN

Selected VSB: 14

Selected CBs: 8

Uploaded NSD: nsd.json

Figure 11: Experiment Blueprint Builder – NSD onboarding



NSD

nsAstiAgvControl

0.1

nsAgv_df

nsAgv_il

Parameters

Param 1

paramId	1	4

Figure 12: Experiment Blueprint Builder – Specification of translation rules

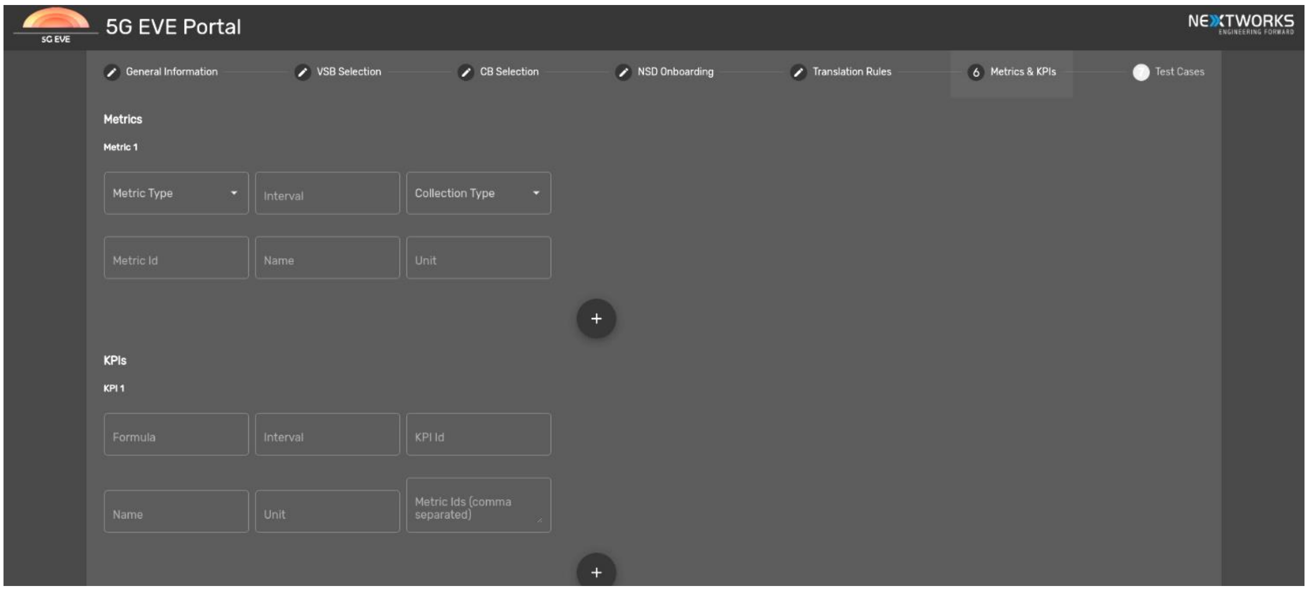
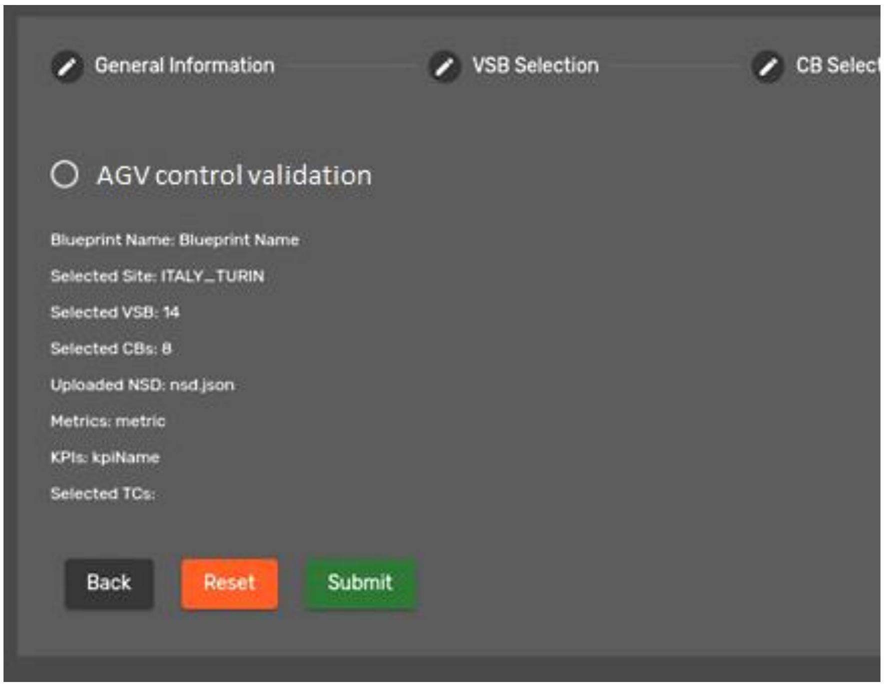


Figure 13: Experiment Blueprint Builder – Specification of metrics and KPIs



AGV control validation

Blueprint Name: Blueprint Name
Selected Site: ITALY_TURIN
Selected VSB: 14
Selected CBs: 8
Uploaded NSD: nsd.json
Metrics: metric
KPIs: kpiName
Selected TCs:

Figure 14: Experiment Blueprint Builder – Selection of test case blueprints

1.4.4 Data Visualization

In the Portal GUI, the **Data Visualization** component is the top layer from the **Experiment monitoring and Maintenance & Results Collection** toolchain, as described in D4.1 [1]. It presents the results, metrics and KPIs collected and saved by the DCS through an intuitive GUI, enabling the monitoring of the progress of the experiment in terms of that information displayed and allowing verticals to interact partially with the visualization tool in an online fashion.

The component implementing this functionality is **Kibana**[6], from the Elastic Stack, as presented in Figure 6. Its main functionalities are the following:

- Visualize the data provided by the DCS and navigate all the Experiment monitoring and Maintenance & Results Collection stack from the same point.
- Provide freedom to select the way to give shape to the data, using a huge variety of interactive visualizations.
- Share visualizations to other actors easily by using the sharing option that works for each stakeholder (e.g. embed a dashboard, share a link, or export to PDF, PNG or CSV files and send as an attachment).
- Organize the dashboards and visualizations through specific spaces.
- Use role-based access control to invite users to certain spaces (and not others), giving them access to specific content and features.
- Monitor the whole Experiment monitoring and Maintenance & Results Collection stack, enabling the configuration of additional features (e.g. add data, secure access, manage pipelines, read the content of ingested files, etc.) by using a visual UI.
- Customise the way of representing data with unique logos, colours and design elements, uploading these designs to the platform in order to use them.

[6] https://www.elastic.co/products/kibana

As user management in the Elastic Stack is provided by the Elasticsearch component, Kibana can also leverage on the functionalities provided by the Role-Based Access Control (RBAC) component for ensuring the Single Sign-On authentication and authorization of users in the monitoring platform.

1.4.5 Browse and look-up

The Browse and look-up tool provides a graphical interface to facilitate to 5G EVE's users the browsing and visualization of the elements stored in 5G EVE catalogues, at the portal and at the I/W framework. The tool provides a view of blueprints and descriptors for vertical services, experiment execution contexts, test cases, experiments, NFV network services, VNFs and PNFs, showing their main information and graphical representations. A full description of the browse and look-up tool is available in D4.1 [1], section 4.3.

2 Experimental Portal implementation

This section presents implementation details about all the components included in the 5G EVE Portal, both at the backend and the GUI layers. The software can be found following the links shown in Table 40.

Table 40. Links to the 5G EVE Portal software.

Component	URL
Sign-up/Login and RBAC	https://github.com/5GEVE/5G-EVE-PORTAL-BACKEND-rbac
VNF storage and File storage	https://github.com/5GEVE/5G-EVE-PORTAL-BACKEND-fs
Experiment Blueprint Builder	Included in Portal Catalogue GUI, available at: https://github.com/nextworks-it/slicer-catalogue/tree/5geve-release/EVE_CATALOGUE_GUI
Experiment Lifecycle Manager	https://github.com/nextworks-it/experiment-portal/tree/master/ExperimentLifecycleManager
Ticketing System Backend	https://github.com/5GEVE/5G-EVE-PORTAL-BACKEND-tsb
Other components declared in D4.1	https://www.5g-eve.eu/wp-content/uploads/2019/11/5g-eve-d4.1-experimentation-tools-and-vnf-repository.pdf

2.1 Portal backend

The Portal backend is composed by a number of stand-alone software components that interact with each other and with the underlying I/W framework, mainly using REST APIs. The entities that provide services for the 5G EVE users also offer north-bound APIs towards the Portal GUI, which mediates the access to the different functionalities implemented in the backend. The following sections provide the details of the implementation for the components of the Portal backend.

2.1.1 Role-Based Access Control

The RBAC component is implemented in Python using the Flask microframework. Flask is a super lightweight WSGI web application framework that provides simplicity, flexibility and fine-grained control. Based on that, we implemented a set of endpoints that conforms a REST API exposed both to front-end application and back-end components. RBAC can be seen as a middleware interface between the front-end application and the IAM service for making this communication decoupled and avoid HTTP redirections.

For the IAM service, we deployed one of the most valuated open source options called Keykloak. The Keycloak service is based on Java and provides Identity and Access Management together with Single Sign-On capabilities. In order to communicate with the IAM service, we have implemented an OpenID client in python in order to enable the communication between the Flask application and Keycloak. The OpenID client uses both OpenID and administration endpoints exposed by Keycloak. We have included the client as a module of the Flask application to combine the endpoints exposed by Flask and the functionality provided by the OpenID client.

2.1.2 Experiment Lifecycle Manager

The Experiment Lifecycle Manager (ELM) is implemented in Java, as a Spring-boot application. It adopts Apache Maven [7] as building tool, PostgreSQL [8] as backend database and RabbitMQ [9] as internal bus for the exchange of asynchronous messages among the ELM internal components.

The ELM high-level software architecture is represented in Figure 15. At the ELM northbound, the REST APIs offered to the ELM clients is implemented through the "ELM REST controller" module. The HTTP messages are based on JSON and they are parsed and formatted using the Jackson library [10]. The authentication and authorization of the HTTP messages is handled interacting with the RBAC system, using Keycloak as backend server (this interaction is omitted from the picture for simplicity).

The ELM REST controller manages the received requests providing the parsing and a first validation of the message format, delegating the actual implementation of the requested functionalities to the "ELM Engine". The ELM Engine represents the core of the ELM prototype and it is developed as a singleton that implements the java "ELM Service Interface", which in turn offers all the primitives related to the creation, retrieval and lifecycle management of 5G EVE experiments.

Whenever the ELM receives a request for the creation of a new experiment, the ELM Engine creates a new entry in the internal records and instantiates a new "Experiment Instance Manager". This module is in charge of managing the Finite State Machine (FSM) regulating the lifecycle of the experiment (see section 1.1.2 for details about the FSM of 5G EVE experiments) and implements the logic that coordinates the interaction with the 5G EVE platform external entities. For persistency reasons, the information related to the experiments are kept not only in the local memory, but also in the PostgreSQL-based database. The access to the database is handled using the Java Persistency API (JPA), that mediates the interaction with the ELM JPA repositories, and it is wrapped through the "ELM Records Manager". This component provides a set of synchronized methods to create, update, query and delete experiments from the internal records.

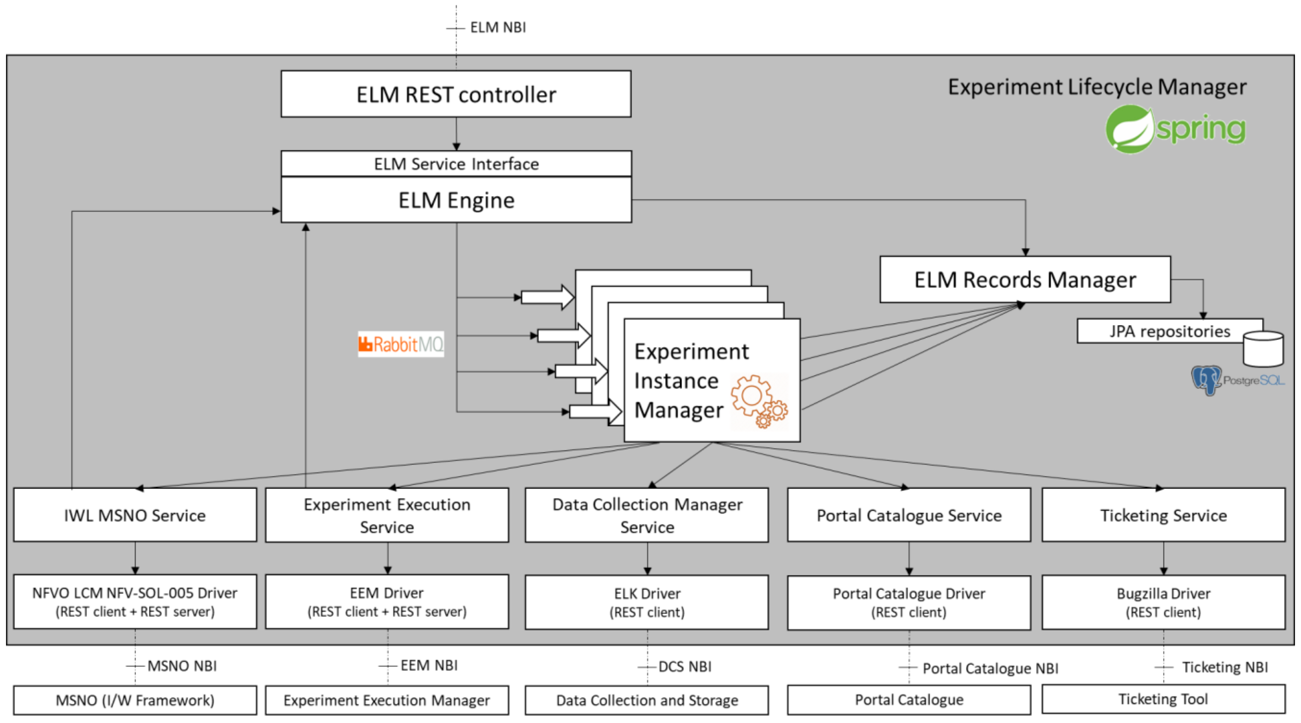

ELM NBI
Experiment Lifecycle Manager
spring
ELM REST controller
ELM Service Interface
ELM Engine
ELM Records Manager
JPA repositories
PostgreSQL
RabbitMQ
Experiment Instance Manager
IWL MSNO Service
Experiment Execution Service
Data Collection Manager Service
Portal Catalogue Service
Ticketing Service
NFVO LCM NFV-SOL-005 Driver (REST client + REST server)
EEM Driver (REST client + REST server)
ELK Driver (REST client)
Portal Catalogue Driver (REST client)
Bugzilla Driver (REST client)
MSNO NBI
EEM NBI
DCS NBI
Portal Catalogue NBI
Ticketing NBI
MSNO (I/W Framework)
Experiment Execution Manager
Data Collection and Storage
Portal Catalogue
Ticketing Tool

Figure 15: Experiment Lifecycle Manager: high-level architecture

The ELM engine acts as a message dispatcher towards the different instances of Experiment Instance Managers, enabling an efficient management of concurrent experiments and the asynchronous communication with external software components, like the I/W framework MSNO and the EEM. In particular, the ELM Engine receives requests or notifications coming from external entities, it identifies the particular experiment instance, the message is referred to and, where needed, it dispatches the message to the related Experiment Instance Manager, feeding its FSM process. The interaction between ELM Engine and Experiment Instance Manager is handled through RabbitMQ queues, using topics with the structure "lifecycle.<action>.<experimentId>". Each Experiment Instance Manager is registered to receive all the messages with topic "lifecycle.*.<experimentId>", where the "experimentId" field indicates the unique identifier of the managed experiment.

The queries received by the ELM Engine, e.g. from the ELM NBI through the ELM REST controller, are processed synchronously directly interacting with the ELM Records Manager.

The interaction with the other elements of the 5G EVE platform is handled through a set of "south-bound service" components. Each of them is implemented as a singleton that wraps the complexity of the tool-specific communication and offers a single point of contact, modelled as a well-defined java interface, towards the external tool for all the ELM internal entities. The details of the communication are handled through dedicated drivers, allowing to easily manage future changes in the external components.

The drivers are implemented as REST clients for all the external components requiring only a synchronous interaction with the ELM. These components are the DCS, used to register and unregister experiments metrics and KPIs; the Portal Catalogue, used to query blueprints or descriptors and to retrieve the translation towards the NFV network service descriptors; and the Bugzilla-based Ticketing tool used to send tickets to the 5G EVE users.

The drivers towards entities with an asynchronous and bidirectional interaction with the ELM include not only a REST client, but also a REST controller. This module is in charge of receiving and parsing the external notifications, forwarding them to the ELM Engine where they are processed and dispatched to the associated Experiment Instance Manager. This kind of interaction is implemented for the MSNO and the EEM.

The ELM software is released as open source software under the Apache 2.0 license and it is available in the following public repository:

https://github.com/nextworks-it/experiment-portal/tree/master/ExperimentLifecycleManager

2.1.3 Data Collection and Storage

As described previously, the Data collection and storage component will be implemented with Logstash and Elasticsearch, including different plugins for value-added functionalities, such as the management of the infrastructure and application metrics, KPIs and results that are provided by the ELCM, or the integration of authentication and authorization mechanisms provided by the RBAC with Keycloak.

Currently, a “dockerized” environment for testing the 5G EVE Monitoring & Data Collection tools is available in the following repository:

https://github.com/5GEVE/5geve-wp4-monitoring-dockerized-env

This project implements the Elastic Stack connected to a Kafka cluster, which plays the role of the Data Collection Manager component of the Interworking Layer.

Currently, it misses the following functionalities, which will be detailed, included and referenced in the next deliverable D4.5:

- Decoupling of Data Collection Manager and ELK Stack in the repository aforementioned. It is intended to deploy both components separately in the integration phase and check that the workflows are executed correctly.
- Full integration of Keycloak for enabling Single Sign-On procedures for authentication and authorization of users. This will be done with the installation of a Keycloak Proxy in the server which holds the DCS, handling the communication with the RBAC.
- Definition of roles for establishing the permissions for accessing to the different REST APIs of each component that belongs to ELK.
- Deployment of a Python client for the interaction with the Data Collection Manager during all phases of the experiment monitoring and performance analysis workflow, using the Kafka native protocol for the communication between both components.

2.1.4 File Storage

This component is implemented using Python and several Python libraries like flask and *werkzeug*. Because a file could be sent in several chunks, the file storage backend is prepared to receive several POST methods for the same file. For the first chunk, the server checks if a file with the same name already exist in the data directory. If so, the server replies with a 400 error.

After receiving a chunk, the server appends the chunk to the file by using the *dzchunkbyteoffset* parameter included in the request, which identifies the starting position of the chunk within the whole file. When the last chunk of the file is received, and after checking that the local file size matches the *dztotalfilesize* parameters included in the request, the server replies back with a success message.

2.1.5 Catalogue Service

The 5G EVE Portal Catalogue is developed as an extension of the Vertical Service Blueprint catalogue embedded in the 5G-TRANSFORMER Vertical Slicer[7]. The software design of the Portal Catalogue is fully documented in deliverable D4.1 [1], section 5.4.

The 5G EVE Portal Catalogue software is released as open source software under the Apache 2.0 license and it is available in the following public repository:

https://github.com/nextworks-it/slicer-catalogue/tree/5geve-release

2.1.6 Ticketing System Backend

As already described in deliverable D4.1 [1], this component will be deployed using the Bugzilla open source software [11].

2.2 Portal GUI

2.2.1 VNF Storage

The VNF storage component will be implemented as a component of the 5G EVE Portal Catalogue. This component will be implemented inside Angular 8 framework, following the approach of implementing a stateless component that relies on a service to perform all the functionality.

2.2.2 Sign-up/Login

The sign-up and login components are implemented inside the web-based GUI for the 5G EVE Portal Catalogue. Both sign-up and login are implemented as components inside Angular 8 framework. We have implemented two angular services supporting the authentication functionality that consumes the RBAC component placed at the back-end. Following this structure, the functionality and visualization of the component is decoupled from the service that consumes the RBAC component.

The implementation will be included as part of the 5G EVE Portal Catalogue repository at "EVE_CATALOGUE_GUI".

2.2.3 Data Visualization

As described previously, this component will be implemented with Kibana from the Elastic Search, providing an environment which integrates both Data Collection and Storage and Data Visualization components.

This Data Visualization component is also integrated in the "dockerized" environment which has been mentioned in the Data Collection and Storage implementation subsection (2.1.3). As a first approach, the integration of this component in the Portal GUI will be done providing a single URL to access to all the capabilities exposed by Kibana, but it is expected to integrate both GUI in a single view eventually, aspect that will be covered in next implementation deliverables.

2.2.4 Browse and lookup

[7] https://github.com/5g-transformer/5gt-vs

The Browse and lookup tool is implemented as a web-based GUI for the 5G EVE Portal Catalogue. The web application runs on Node JS, it is written in TypeScript and it is based on the Angular 8 framework, using the Angular Material toolkit for the graphical layout. Its implementation is reported in deliverable D4.1 [1], sections 4.3.2-4.3.4.

The software is an open source software under the Apache 2.0 license, and it is available in the following public repository (in the "EVE_CATALOGUE_GUI" folder):

https://github.com/nextworks-it/slicer-catalogue/tree/5geve-release

3 Service handbook

This section presents the main steps to design, define, prepare and execute experiments using the 5G EVE Portal GUI. This explanation will be structured based on the three different phases already presented in D4.1 [1]: (i) experiment design and definition, (ii) experiment preparation and (ii) experiment execution. For the sake of completeness, in this deliverable we include Figure 16, which is also included in D4.1. This figure describes the workflow and iterations between the different actors and the 5G EVE Portal GUI to implement each phase.

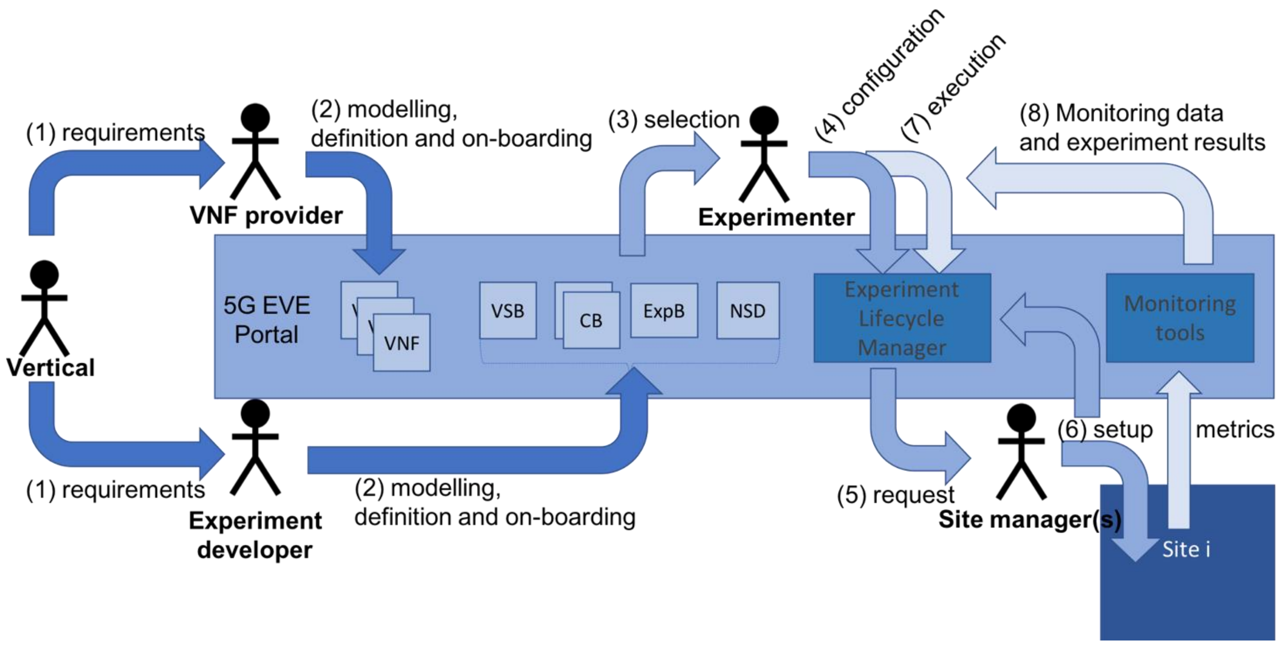


Figure 16: Experiment phases, main actions and actors involved.

3.1 Experiment design and definition

This first stage starts with an offline iteration between verticals and both VNF providers and experiment developers (step 1 in Figure 16). The result of this iteration is a set of VNFs generated by the VNF providers to be on-boarded in one or different sites and the proper information to define the related Blueprints and NSDs (see D4.1 [1]). Then, the following actions have to be executed in sequence (steps 2 in Figure 16):

- VNF uploading performed by VNF provider using the VNF storage functionality provided by the Portal GUI. VNF providers have to upload a *zip* file including the proper VNFD and constituent images, together with a list of sites where the uploaded VNF have to be onboarded. This action triggers the creation of one ticket per selected site, including the link to the uploaded *zip* package. This way, site managers can download such package to check it and, if valid, onboard it on their local NFVO.
- Blueprints and NSD definition and onboarding by experiment developers. There are different steps to perform these actions:
    - Experiment developers have to define VSB, CB, TCB and NSD supported by the Browse, lookup and onboarding tool, which can be used to collect the identifiers of the constituent components (i.e., VNFs and PNFs). Experiment developers can browse available components, using the provided information to define all these blueprints and descriptors offline, to finally onboard them to the 5G EVE catalogue.
    - Define Experiment Blueprints (ExpB) by experiment developers using the Experiment Blueprint Builder available at the Portal GUI. This tool provides the functionality to select a base

VSB and then attach zero or more Context Blueprints. Finally, one or more Test Case Blueprints could be attached to the ExpB.

The above procedures are summarized in Table 41 below.

Table 41. 5G EVE Portal Experiment design and definition actions summary.

Action	Actor	GUI Component	Description	Output
VNF Uploading	VNF Provider	VNF Storage	Upload a zip file containing the proper VNFD and the corresponding images. Can include a list of sites indicating where the uploaded VNFs should be onboarded.	As many tickets as sites selected.
Blueprints and NSD definition and onboarding	Experiment Developers	BLUO	Define VSB, CB, TCB & NSD, followed by onboarding of these descriptors to the 5G EVE catalogue	Blueprints and NSDs onboarded on the 5G EVE Portal catalogue.
		EBB	Define ExpB by attaching to a VSB one or more CBs and TCBs.	An ExpB.

3.2 Experiment preparation

Experimenters and site managers are the two actors involved in this phase. The former use Blueprints and NSDs created in the previous phase to complete the missing information to fully define an experiment. This is done in steps (3) and (4) shown in Figure 16, and there are two different ways to perform this task:

- Experimenters use one of the functionalities provided by the Intent-Based Networking (IBN) tool to express their intent to execute an experiment. The IBN tool selects the most appropriate ExpB, filling in the input parameters using the information provided in the intent. If, after processing the intent, the tool detects any missing parameter, additional information will be requested from the experimenter. The result is the creation of an experiment object, which includes a reference to the resulting ExpD.
- Experimenters can also use the guided selection service provided by the IBN tool to select an ExpB and then the tool requests all input parameters. Similar to the previous case, the result is the creation of an experiment object, which includes a reference to the resulting ExpD.

The previous steps end with the creation of a new ticket, which will be received by all site managers involved in the experiment (this information is included in the experiment object). The ticket includes all information available in the experiment object, such as the experiment descriptor identifier, the proposed days to execute the experiment, etc. With this information, the site managers have to prepare their infrastructure to accommodate the experiment. Site managers will reply to these tickets to schedule the experiment. This is further illustrated in Table 42.

Table 42. 5G EVE Portal Experiment preparation.

Action	Actor	GUI Component	Description	Output
Experiment preparation	Experimenters	IBN	Use the intent to execute an experiment and with this intent, the IBN tool selects the most appropriate ExpB and fills in the input parameters.	ExpD and experiment object. Creation of a new ticket
			Select an ExpB and the IBN tool requests all input parameters	
	Site Managers	Ticketing GUI. Run Experiment.	Receive of a new ticket generated by the previous step and preparation of the 5G EVE infrastructure to accommodate the experiment	Schedule experiment

3.3 Experiment execution

When the experiment is in the ready state, experimenters can request the execution of the experiment (step (7) in Figure 16). Experimenters, using the Run Experiment service provided by the 5G EVE GUI, requests the execution of a given experiment object. Finally, experimenters can monitor the metrics defined in the Experiment Blueprint either while the experiment is running or after it is completed, as shown in step (8) in Figure 16:

- Experimenters, using the Data Visualization tool provided by the 5G EVE GUI, can monitor the metrics already defined in the Experiment Blueprint.

A summary of the experiment execution procedure is shown in the following Table 43.

Table 43. 5G EVE Portal Experiment execution.

Action	Actor	GUI Component	Description	Output
Experiment execution	Experimenters	Run Experiment	Request the execution of a given experiment	Running Experiment
		Data Visualization	Monitor the metrics of a running/ completed experiment as defined in the ExpB	Experiment Metrics

It is important to note that there will be other services provided by the 5G EVE GUI such as the performance diagnostics and validation tool. Note that all these services will be described in deliverables produced in work package 5.

4 Conclusions

This deliverable includes the architecture, REST APIs and implementation details of the first release of the 5G EVE Portal. The services available in this first release are detailed in the introduction of this document, and this is a subset of all services that have to be offered by the Portal, which is included in D4.1. Actors accessing the 5G EVE end-to-end facility have two different ways to consume the services provided by the Portal: (1) using the graphical user interface provided by the 5G EVE Portal GUI or (2) by using the REST API offered by the Portal backend.

The service handbook included in this document shows all steps that have to be done in order to implement the different phases of an experiment, using the 5G EVE Portal GUI.

The second release of the 5G EVE Portal will provide in month 24 (June 2020) all services defined in D4.1, including a complete integration among all components inside the Portal backend and the integration between these components and those provided by other work packages in the 5G EVE project.

Acknowledgment

This project has received funding from the EU H2020 research and innovation programme under Grant Agreement No. 815074.

References

[1] 5G EVE Project, "Deliverable D4.1: Experimentation tools and VNF repository", October 2019.
[2] Keycloak, https://www.keycloak.org/ Date accessed: 13 December 2019.
[3] ELK stack, https://www.elastic.co/what-is/elk-stack Date accessed: 13 December 2019.
[4] ETSI GS NFV-IFA 014, "Network Functions Virtualisation (NFV) Release 3; Management and Orchestration; Network Service Templates Specification", v3.1.1, August 2018
[5] ETSI GS NFV-SOL 005, "Network Functions Virtualisation (NFV) Release 2; Protocols and Data Models; RESTful protocols specification for the Os-Ma-nfvo Reference Point", v2.6.1, April 2019
[6] 5G EVE Project, "Deliverable D5.2: Model-based testing framework", December 2019
[7] Apache Maven, https://maven.apache.org/ Date accessed: 13 December 2019.
[8] PostgreSQL, https://www.postgresql.org/ Date accessed: 13 December 2019.
[9] RabbitMQ, https://www.rabbitmq.com/ Date accessed: 13 December 2019.
[10] Jackson JSON library, https://github.com/FasterXML/jackson Date accessed: 13 December 2019.
[11] Bugzilla, https://www.bugzilla.org/ Date accessed: 13 December 2019.