5Genesis

5TH GENERATION END-TO-END NETWORK, EXPERIMENTATION, SYSTEM INTEGRATION, AND SHOWCASING

[H2020 - Grant Agreement No. 815178]

Deliverable D3.8

Open APIs, service level functions and interfaces for verticals (Release B)

Editor Elisa Jimeno (ATOS)

Contributors ATOS(Atos Spain SA), UMA(Universidad de Málaga), LMI(L.M. Ericsson Limited), UNIS(University of Surrey), INTEL (INTEL Deutschland GmbH), NEMERGENT (Nemergent Solutions SL), NCSRD (NATIONAL CENTER FOR SCIENTIFIC RESEARCH "DEMOKRITOS"), FhG (FRAUNHOFER GESELLSCHAFT ZUR FOERDERUNGDER ANGEWANDTEN FORSCHUNG E.V)

Version 1.0

Date March, 28th 2021

Distribution PUBLIC (PU)

List of Authors

ATOS	ATOS SPAIN
Elisa Jimeno, Javier Melian, Luis Gomez	
UMA	University of Malaga
Bruno Garcia	
FRAUNHOFER	Fraunhofer-Gesellschaft zur Förderung der angewandten Forschung e.V.
A. Prakash	
NCSRD	National Center For Scientific Research “DEMOKRITOS”
H. Koumaras	

Disclaimer

The information, documentation and figures available in this deliverable are written by the 5GENESIS Consortium partners under EC co-financing (project H2020-ICT-815178) and do not necessarily reflect the view of the European Commission.

The information in this document is provided “as is”, and no guarantee or warranty is given that the information is fit for any particular purpose. The reader uses the information at his/her sole risk and liability.

Copyright

The 5GENESIS Consortium consists of:

NATIONAL CENTER FOR SCIENTIFIC RESEARCH "DEMOKRITOS"	Greece
AIRBUS DS SLC	France
ATHONET SRL	Italy
ATOS SPAIN SA	Spain
AVANTI HYLAS 2 CYPRUS LIMITED	Cyprus
AYUNTAMIENTO DE MALAGA	Spain
COSMOTE KINITES TILEPIKOINONIES AE	Greece
EURECOM	France
FOGUS INNOVATIONS & SERVICES P.C.	Greece
FON TECHNOLOGY SL	Spain
FRAUNHOFER GESELLSCHAFT ZUR FOERDERUNG DER ANGEWANDTEN FORSCHUNG E.V.	Germany
IHP GMBH – INNOVATIONS FOR HIGH PERFORMANCE MICROELECTRONICS/LEIBNIZ-INSTITUT FUER INNOVATIVE MIKROELEKTRONIK	Germany
INFOLYSIS P.C.	Greece
INSTITUTO DE TELECOMUNICACOES	Portugal
INTEL DEUTSCHLAND GMBH	Germany
KARLSTADS UNIVERSITET	Sweden
L.M. ERICSSON LIMITED	Ireland
MARAN (UK) LIMITED	UK
MUNICIPALITY OF EGALEO	Greece
NEMERGENT SOLUTIONS S.L.	Spain
ONEACCESS	France
PRIMETEL PLC	Cyprus
RUNEL NGMT LTD	Israel
SIMULA RESEARCH LABORATORY AS	Norway
SPACE HELLAS (CYPRUS) LTD	Cyprus
TELEFONICA INVESTIGACION Y DESARROLLO SA	Spain
UNIVERSIDAD DE MALAGA	Spain
UNIVERSITAT POLITECNICA DE VALENCIA	Spain
UNIVERSITY OF SURREY	UK

Version History

Rev. N	Description	Author	Date
1.0	Release of D3.8	ATOS	31/03/2021

LIST OF ACRONYMS

Acronym	Meaning
3GPP	3rd Generation Partnership Project
5G	5th Generation (of mobile communications)
API	Application Programming Interface
B2B	Business to Business
BSS	Business Support System
GA	Grant Agreement
CA	Consortium Agreement
CAPIF	Common API Framework
Dx.y	Deliverable Number
DB	Data Base
ED	Experiment Descriptor
ELCM	Experiment Life cycle Manager
EPS	Evolved Packet system i.e. 4G, LTE
E-UTRA	Evolved Universal Terrestrial Radio Access
ID	IDentificator
IT	Information Technology
KPI	Key Performance Indicators
MANO	Management and Orchestration
MBMS	Multimedia Broadcast Multicast Services
NFV	Network Function Virtualization
NFVO	NFV Orchestrator
NRF	Network Repository Function
NS	Network Service
NSD	Network Service Descriptor
Mx	Month # of the project work plan (e.g. M2)
NEF	Network Exposure Function
NR	New radio
OSM	Open Source MANO
OSS	Operational Support System
REST	Representational State Transfer
SBA	Service Based Architecture
SBI	South-Bound Interface
SCEF	Service Capability Exposure Function
SMF	Session Management Function

Tx.y	Task Number
TR	Technical Report
TS	Technical Specification
UI	User Interface
UE	User Equipment
VNF	Virtual Network Function
VNFD	Virtual Network Function Descriptor
WPx	Work Package Number

Executive Summary

The 5GENESIS facility is composed of five experimental Platforms with complementary features, distributed across Europe. Each one of these Platforms follows a common reference implementation architecture, that requires an open Application Programming Interface (API) in order to offer to Experimenters, the required interfaces and common method to interact with the Facility. Therefore, offered to the broadest possible audience, the APIs provide interaction with the platforms for the validation of the 5G KPIs uses cases.

The focus of this deliverable is to present the design and implementation of these 5GENESIS Open APIs (architecture, flow diagram, exposed features, and interfaces) from the final development of the Release B related to the activities under T3.4 Open API, service level function and interfaces for verticals.

The 5Genesis Open API is the main interface for Experimenters to define and execute their experiments. The Dispatcher is the component engine that exposes the Open API and redirects the request to the required service in the infrastructure. All the requests are secured, thanks to the authenticator module developed to authenticate the user before the action can be taken. Key components of the Open APIs architecture include the Validator for NS and ED, and the Distributor component that validates correct access of Experimenters to offer the interaction of the Experiment and resources executed.

These interfaces can be interacted by command line for more experience users, but it also offers a Portal with a friendly Web User Interface (UI) to facilitate the interaction with the Facility. The Portal itself plays the role of a client of the 5GENESIS Open API and is able to display the execution logging output from all execution stages of the experiments (Pre-Run, Run and Post-Run). Besides, for each experiment execution, it provides a link to a customized experiment specific Grafana dashboard for easy visualization of the data generated by the experiment. Release B of the Portal, as final implementation is presented in this deliverable.

In summary, this document presents the endpoints available for the Experimenters to interact with the 5GENESIS facility, either via the Portal (an abstraction layer and client example of the Open API), or directly using the Open API.

Table of Contents

1. INTRODUCTION

This deliverable is the second release of the two reports on *Open API, service-level functions and interfaces for verticals* with the updated information regarding what has been implemented during the second period (M16-M33) of the project lifetime. The previous report, *D3.8, was* delivered in month 15.

The purpose of these two deliverables is to present the exposed Open API by the 5GENESIS Facility with the goal of offering, especially to 5G vertical industries, an open and common method for experimentation. This deliverable includes the description of the Open APIs final implementation (Release B) and the improved functionalities identified in the roadmap for its evolution. It includes the design, implementation and testing of the features as well as the tenant web Portal, which is an alternative method for experimenters to access the 5GENESIS facility; its goal is easing the experimenters´ work when using the facility.

The work described in both mentioned deliverables corresponds to the task *T3.4 Open API, service-level functions and interfaces for verticals* included in the *WP3 Openness Framework and Integral Components of the Facility*. These two deliverables are complemented with 14 other deliverables, each corresponding to the other tasks included in WP3 and delivered in M33 and M36. All these documents together will provide a complete overview of the work and results delivered by WP3 during the duration of the project.

The deliverable is organized in the following manner:

- Section 1 (this section) is an introduction to the deliverable.
- Section 2 provides a summary of the features presented during the Release A and the improvements during the second release of the component.
- Section 3 introduces the main Reference Points identified for the 5GENESIS Portal. The new architecture of the Dispatcher is also explained.
- Section 4 is the main section of the deliverable as it presents the Dispatcher component. The design, workflow, interfaces and implementation of the application gateway and interaction with the sub-components are described here.
- Section 5 describes the required packages and manual to deploy, instance and run the Dispatcher component trough the swagger interface.
- Section 6 defines the testing framework and evaluated tests of the internal functionalities of the Open APIs interfaces that assure the correct functioning of the component.
- Section 7 presents the design and implementation of the Graphical User interfaces provided by the Facility of the Portal
- Section 8 provides the conclusions of the work done.

2. RELEASE A SUMMARY

During the evolution of the project, the 5GENESIS architecture has been amendment to comply with the new features and requirements of the components and use cases. The most relevant changes have been focused on the coordinator layer delivered for the most part as the Open 5GENESIS Experimentation Framework[1]. In this document we will focus on the Open APIs as the entry point of the facility and its interfaces with the underlying components. The figure bellow presents the updated 5GENESIS architecture and highlights the components that belong to the Dispatcher.

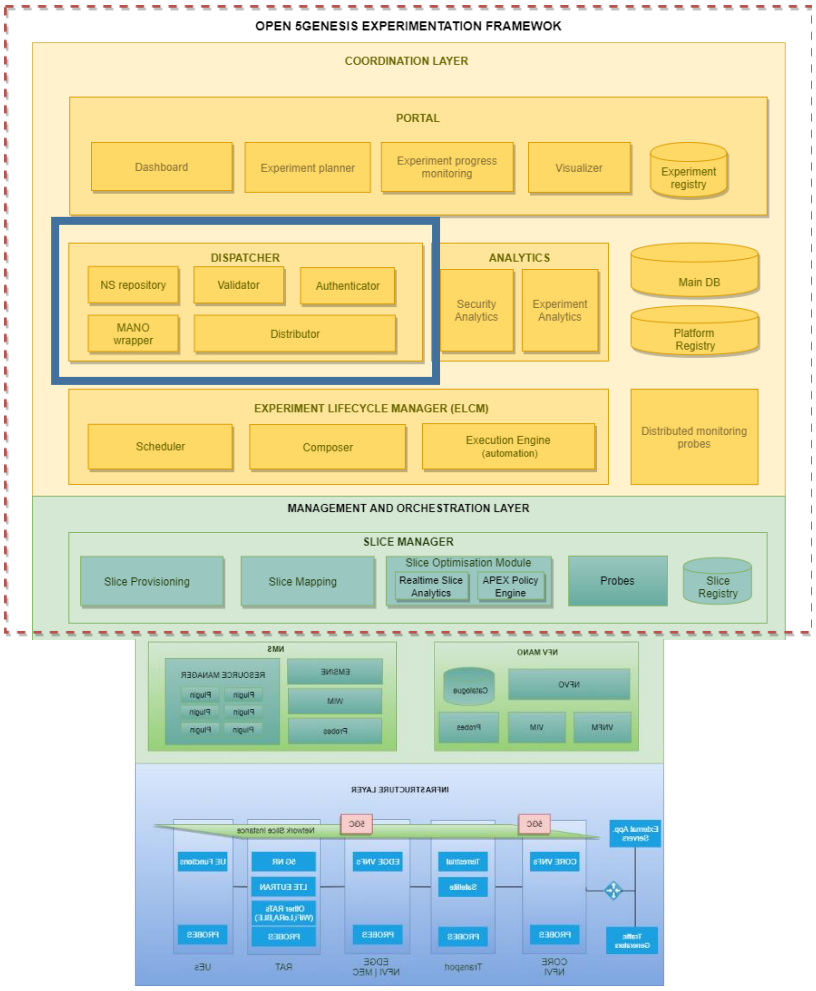


Figure 1: 5GENESIS Architecture

During the first period of the project, the Dispatcher component was designed in order to comply with the requirements for the underlying components. The following picture depicted in Figure 2 illustrates the actual architecture at the time of delivering the Release A component. In the figure we can see the initial components identified in the architecture and the interfaces to the other components in the 5GENESIS Coordination Layer, and the communication with the MANO Layer, by connecting with the NFVO in the infrastructure.

[1] https://github.com/5genesis

During the early stages of the project, very few features had been implemented to validate the first cycle of evaluation during the delivery of Release A. In the meantime, due to the evolution of the components involved in the 5Genesis architecture, the functionalities of the Open APIs have also evolved.

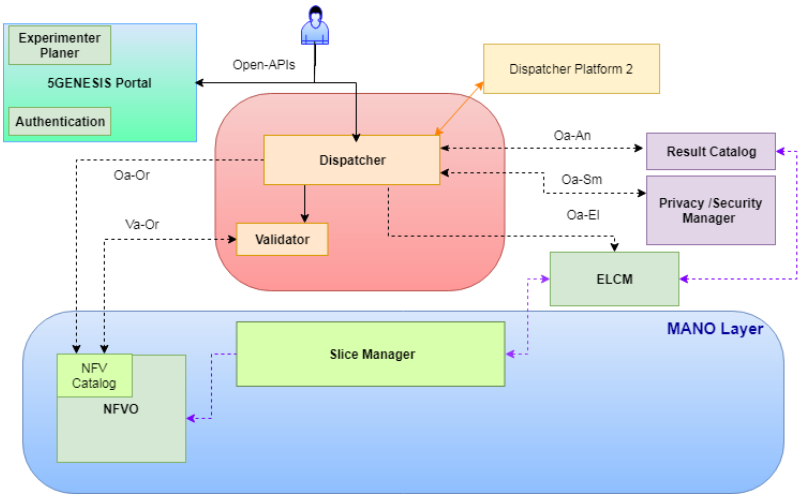


Figure 2: Open API Release A

The main advances in the new Release B of the Open APIs component will be described in this deliverable. In this section we list the features that has been implemented so far.

- Authenticate user to access the platform.
- Store NFV descriptors in the platform repository.
- Onboard Network Services (composed of VNFD, NSD and images) (MANO layer).
- Launch Experiment (Onboard Experiment Descriptor).
- Validate VNFD, NSD and ED.
- Retrieve List of experiment resources (Network Services, Test Cases, Scenarios, UEs…).
- Retrieve KPIs (Analytics module expose the query-result of the test cases from a specific experiment).
- Distribute Experiments among facilities.
- Dispatcher robot self-validation testing.
- Authorize Experimenter to own resources.

3. OPENAPIS IN 5GENESIS

5GENESIS architecture has been designed in a set of three layers to differentiate the different modules developed in the framework of the project. These three layers explained in detail in Deliverable D2.4 [1], called Coordinator, Management and Infrastructure layer, allows to separate the communication for the different phases in the Experimentation facility.

The Coordinator layer define and prepares all the required information and automation of the Experiment; the MANO layer coordinates and reserve the resources in the infrastructure to allocate the slice and instantiate the service and application to perform the Experiment; and the Infrastructure layer performs the execution of the test cases and KPIs validation of the Experiment in the facility based on the definition in the coordinator layer. The Open APIs that resides in the Coordinator Layer, is the entry point (interface) to the facility as it communicates with the underlying modules in a centralized way. The features identified in Deliverable D2.4 [1] are described as follow:

- Authenticate user to access the platform.
- Onboard Network Services (composed of VNFD, NSD and images) (MANO layer).
- Retrieve List of experiment resources (Test Cases, Slice, Scenarios, UEs, Applications, Experiments, Network Services).
- Launch Experiment (Onboard Experiment Descriptor).
- Validate VNFD, NSD and ED.
- Retrieve results (Experiment provide as output the test case results as well as console logs from a specific experiment).
- Distribute Experiments among facilities .

3.1. Reference points

An interface is a point of interconnection between two systems or parts of a system. An interface might qualify as standard when the information flowing through that interface is common to all processes of that type.

5GENESIS Open APIs try to be as close to the standards as possible to ease a potential interoperability between a 5GENESIS Platform and an external one, but also for not reinventing the wheel.

TM Forum's suite of 50+ REST-based Open APIs has been collaboratively developed to be used in a range of scenarios, internally enabling service providers to transform their IT and operational agility and customer centricity, while externally delivering a practical approach to seamless end-to-end management of complex digital services [2].

We have classified the 5GENESIS Open APIs interfaces depending on the type of process in order to compare them with the TM Forum Open APIs.

3.1.1. User management operations

User registration

This interface is used in 5GENESIS to register a user within a platform so the user can interact with it. Studying TM Forum's suite, the closest one is *Customer Management API* [3], which provides a standardized mechanism for customer and customer account management operations such as creation, update, retrieval, deletion and notification of events. Customer can be a person, an organization or another service provider who buys products from an enterprise. Customer management API allows management of identification and financial information about them.

In 5GENESIS we do not need financial information and use only very limited information from the user to avoid GDPR issues and because it is not necessary for our PoC. We can say our interface is a very reduced version of the TM Forum one by removing the extra load.

Platform registration

The platform registration is a specific case of the user registration, so it fits also under the TM Forum *Customer Management API* [2]. The process is the same as with the previous one: removing all unnecessary fields and keeping only the basic ones to validate our PoC: email, user ID and password.

3.1.2. Service catalogue operations

VNFD/NSD CRUD

These are very specific operations related to NFV so it is difficult to find a standard that matches this interface. It can be reduced to catalogue operations by changing the type of item in the catalogue: in our case, to NFV descriptor packages. Some ideas were extracted from similar processes:

- *Product Catalog Management API* [2]: The catalog management API allows the management of the entire lifecycle of the catalog elements, the consultation of catalog elements during several processes such as ordering process, campaign management, sales management.
- *Product Ordering API* [2]: The Product Ordering API provides a standardized mechanism for placing a product order with all of the necessary order parameters. The API consists of a simple set of operations that interact with CRM/Order Negotiation systems in a consistent manner. A product order is created based on a product offer that is defined in a catalog. The product offer identifies the product or set of products that are available to a customer, and includes characteristics such as pricing, product options and market.
- *Resource Catalog Management API* [2]: The Resource Catalog Management API REST specification allows the management of the entire lifecycle of the Resource Catalog elements and the consultation of resource catalog elements during several processes such as ordering process.

- *Service Catalog API* [2]: The Service Catalog Management API REST specification allows the management of the entire lifecycle of the Service Catalog elements and the consultation of service catalog elements during several processes such as ordering process.

Again, the above APIs are too elaborated and do not fit exactly our expectations for this interface. Instead, we have decided to inherit the ETSI standards NFV-SOL 005 API [4] from the NFVO, which is fully with that specification, specifically created for this purpose.

3.1.3. Experiments operations

Launch experiment

This interface was designed to launch an experiment, identified by an experiment descriptor, which is included in the request, therefore, we do not have to deal with a catalogue. It is again a very specific interface designed ad-hoc for 5GENESIS and only has in common with the standard ones that it is intended to activate a service. That is the reason we have selected and studied the following APIs:

- Product Ordering API
- Resource Ordering Management API
- Resource Function Activation and Configuration API
- Service Activation and Configuration API [3]: The REST API Conformance for the Service Activation and Configuration API adds to the other three APIs that were mentioned before in this section.

Abstracting the type of service, this interface is a simplified version of the Service Activation and Configuration API, using a similar service model but with only the basic fields: id, name, description, dates, status, etc. and removing external relationships, which are not needed in 5GENESIS.

3.1.4. Results gathering

The system provides an interface to retrieve the results of all the processing carried out within the 5GENESIS platform after launching an experiment. The following API specification is the closest to that purpose:

Result catalogue

- Performance Management API [3]: The Performance Management API REST specification allows the management and control of the performance of the services.

Our interface works in a similar way, allowing the possibility of retrieving the measurements taken for each job (experiment execution) and also, filter them by a specific measurement. The rest of the proposed API standard has been discarded for 5GENESIS purposes, as we are not using this interface in a flexible way (as originally intended) and as our KPIs are fixed or included in the experiment template and not managed by the API.

3.2. Architecture update

As mentioned before, the Open APIs are considered a virtual interface to the underlying module that can be access by the experimenter and the platform administrator. The Open APIs have been designed based on the previous research and developed as the Dispatcher component in the coordinator layer of the 5GENESIS architecture.

The 5GENESIS Dispatcher is the entry point to the system, offering a virtual interface to the underlying modules with the functionalities to an Experimenter through a single interface. These functionalities are known as the Open APIs, being able to interact with the key features of the underlying modules (as shown in the architecture diagram below) considering security and permission rights to the user without exposing sensitive information and access to the experimenter.

This implementation is based on a NGINX reverse proxy containerized in a Docker environment. By default, The *Dispatcher* includes as added on modules, the Auth, the MANO Wrapper and a Swagger environment to test the available features. On top of all that and to secure all the requests, the Dispatcher provides user registration and authentication using JWT. Consult the Auth documentation in section 5.2.2. Auth module standalone installation, for the available actions and how to use them.

NOTE: As shown below (Figure 3) in the architecture diagram, the *Dispatcher* does not deal with the MANO directly but through a wrapper that simplifies the communication. Also, for simplification, in this document we will refer to the *MANO Wrapper* as ***mano***, which is conceptually correct from the *Dispatcher* point of view.

The following Figure 3 depicts the Open API architecture with the different modules part of the component.

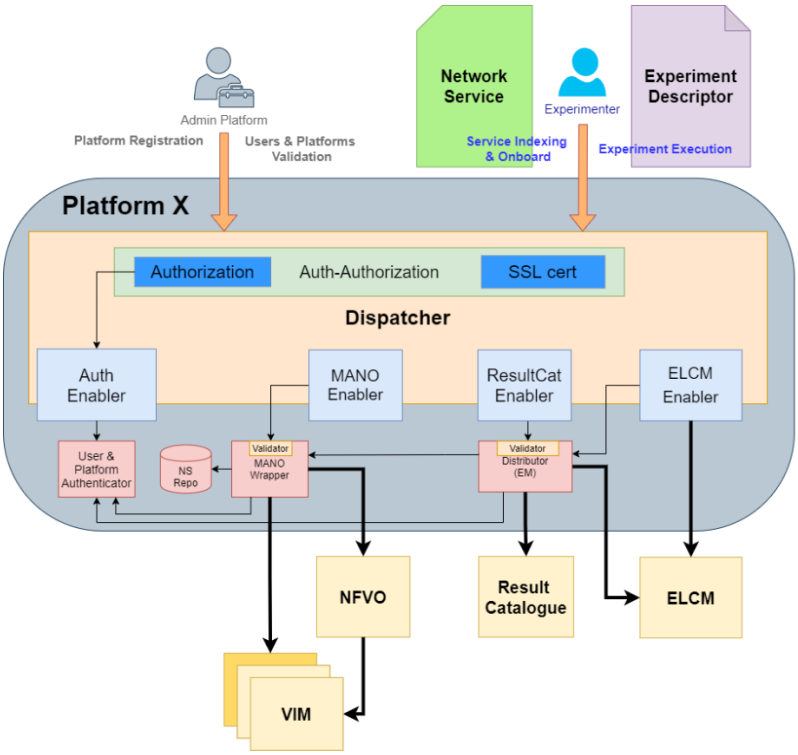


Figure 3: Open API Architecture

As shown in the picture, the Experimenter will connect to the infrastructure through the Dispatcher in order to access the services provided by the Open APIs in the facility.

The Open APIs are managed as docker containers with docker-compose as orchestrator. The composition of those containers is explained in the following table:

Docker image	Docker name	Port	Environment variables	Volumes used
nginx	dispatcher	8082	-	/etc/nginx/nginx.conf /var/log/nginx /etc/ssl /repository
mongo	database	27017-27019	Init Data Bases	/data/db
mano	mano	5101	-	/mano /repository
auth	auth	2000	-	-
swaggerapi/ swagger-ui	swagger	5002	Open APIs file	-
distributor	distributor	5100	ELCM & Result Catalog URLs	-

4. DISPATCHER MODULE

The Open APIs is the central point from which all request received through the Portal, in a more user friendly; or through the Terminal by command-line tools can interact with the 5GENESIS platform. This interface is centralized in the Dispatcher component. It is the application gateway to redirect traffic request from the Experimenter to the underlying management components of the Experiment execution and Services onboarding. It Is composed of Enablers (facilitator) that offers the collection of integrated applications that process the requests to a relative path in the platform and implements authentication based on JWT, securing all the Experiment requests in the platform, providing a reliable infrastructure.

Moreover, in order to communicate with other platforms, it will interconnect with the Dispatcher deployed in the remote location to distribute the Experiment and providing results.

4.1. Authenticator

Authenticator is a REST API module in charge of managing the Authorization required to access the platform. It is designed from the user and admin point of view. The Authenticator manages the authentication, confidentiality, integrity and non-repudiation security features of the platform.

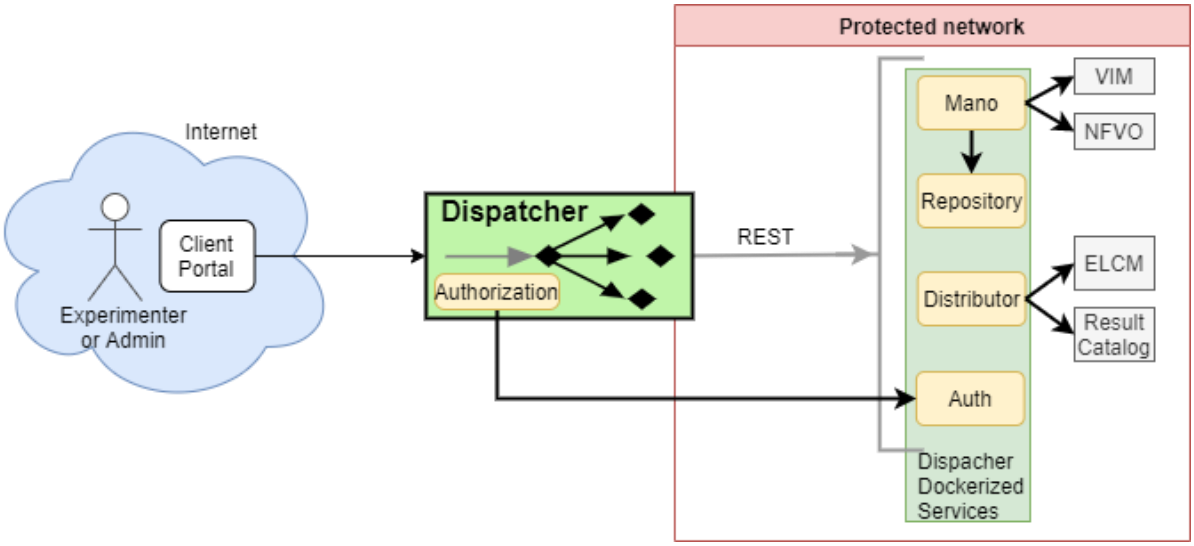


Figure 4: Authentication design

Authentication is delegated to the *Auth* module. Users are registered into it and provides the ability to the user to claim an access token. This token is a JSON Web Token [9]. It contains user's identity (subject id, name, email) and some meta data relatives to the authorization process (issuer, time to live, etc.). The access token can be claimed using Basic Authentication (username + password). The access token is online, that is, a token used by client apps having a direct user interaction (GUI such as: web site, desktop apps, mobile apps, etc). It's a short-lived token, so it is renewed before its expiration date using a refresh token. Once claimed, the access token is renewed as well as the refresh token. And the process is repeated during the whole user session lifetime. Instead of requesting an access token, it is also possible to

authenticate every request using Basic Auth with a user already registered and validated by the Platform Administrator.

Based on the two roles identifies in the platform, the Authenticator has the following features for each role:

- User: also known as Experimenter of the facility, will access the platform to execute the experiments.
  - Registration
  - Get Token
  - Change Password
  - Recover password
  - Delete acount
- Admin: is the administrator of the platform and will validate the registration of users in each facility.
  - Provide user service authorization in a remote platform
  - Show Users
  - Delete an User
  - Validate an User
  - Show list of Platforms
  - Validate a Platform
  - Delete a Platform
  - Drop DataBase

The Database Structure is defined in DB_Model.py and has the following tables to register and have the control of the platform users:

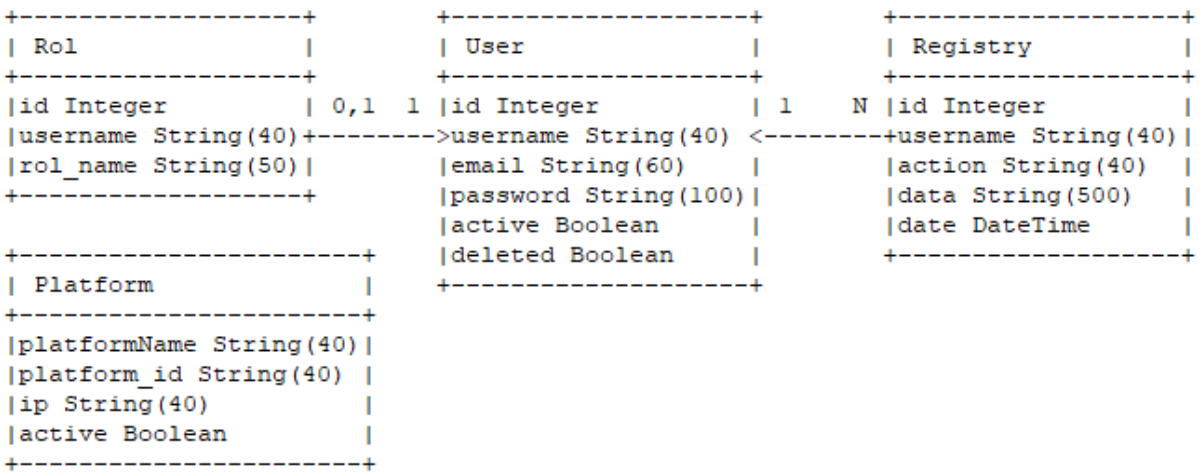


Figure 5: Authorization Database

Once the DB is created, the Admin user is created, with username "Admin", password "Admin" and email "5genesismanagement@gmail.com", settled in the mail config by default.

There are 4 tables.

- For Role table we can found 3 attributes:
  - id >> Primary key
  - username >> Foreign key to User table
  - rol_name >> Role in the system
- User table with 6 attributes:
  - id >> Primary key
  - username >> Unique key to User
  - email >> Unique in the system
  - password >> Password for the user
  - active >> Account validated, true for validated, false for not.
  - deleted >> Account deleted by the users. But it persists for view the traces
- Registry table with 5 attributes:
  - id >> Primary key
  - username >> Foreign key to User table
  - action >> function requested by the User
  - data >> parameters for the request given
  - date >> timestamp with the exact time where the action was requested
- Platform table with 4 attributes:
  - platformName >> Unique key, needed for identify the platform in a simple way
  - platform_id >> Primary key
  - ip >> IP of the platform
  - active >> Platform validated, true for validated, false for not.

The Open APIs are described in a json file indicating all the possible interfaces, its required inputs, and the expected outputs. The json file is exposed on a swagger service in the 5002 port. Admin and users can use this interface. The users can use the token or the basic auth as authorization method for the Open APIs operations. It is recommended to use the token instead the basic auth as it adds more security in the system, due to the time constrained lifetime feature of the token.

The endpoints in the auth component are the following:

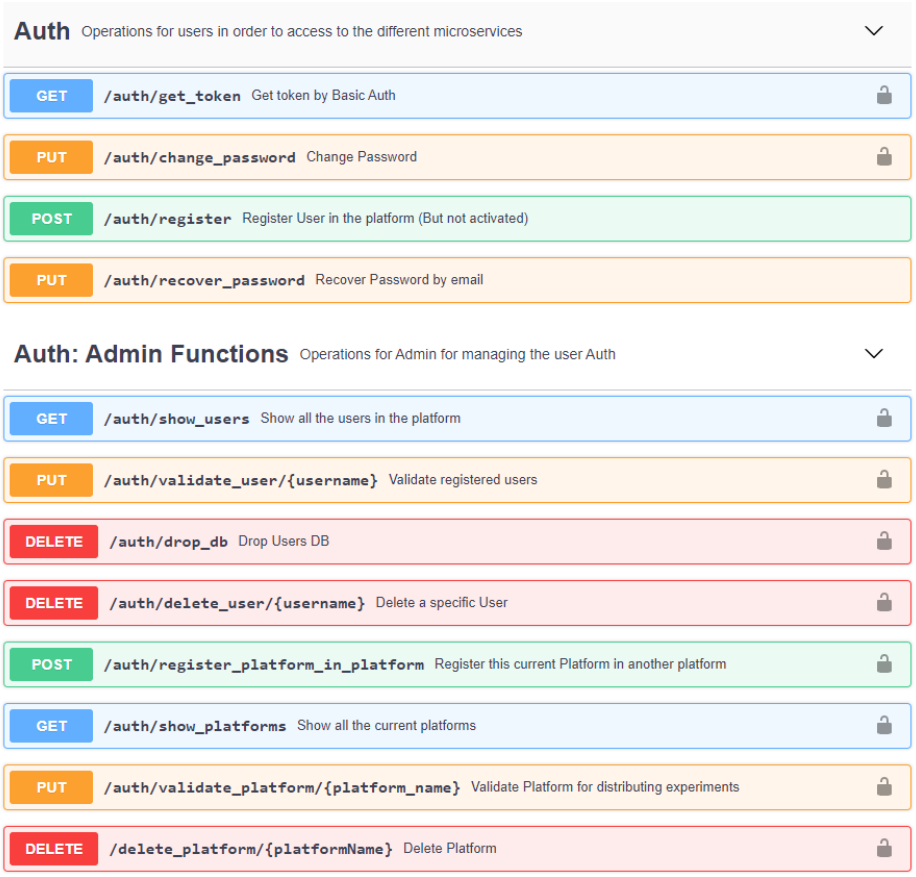


Figure 6: Authorization Open APIs interfaces

Sequence diagrams & requests schema

The following sequence diagrams explain in detail how the authorization module works, the first part of these diagrams is oriented from the user perspective and the second part is oriented from the perspective of an administrator of the system.

User Registration

The first step the user needs to fulfil in order to connect to the platform is to create a valid account to access the facility. In the main page of the Portal, or through the command line, users can request the registration with the username, password to access the facility and a valid email that will be used to validate the account and inform the user about any information related to the facility.

The following diagram show the different steps that the user needs to accomplish to get the account created and validated from the platform administrator.

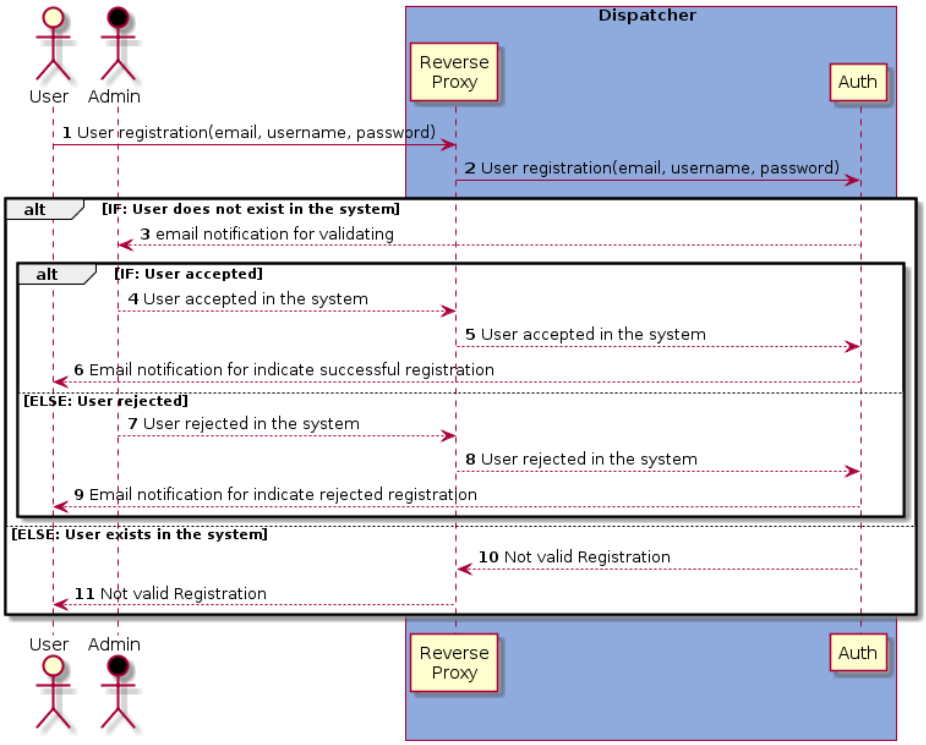


Figure 7: User registration through OA sequence diagram

The request the user will do through the Dispatcher is the following:

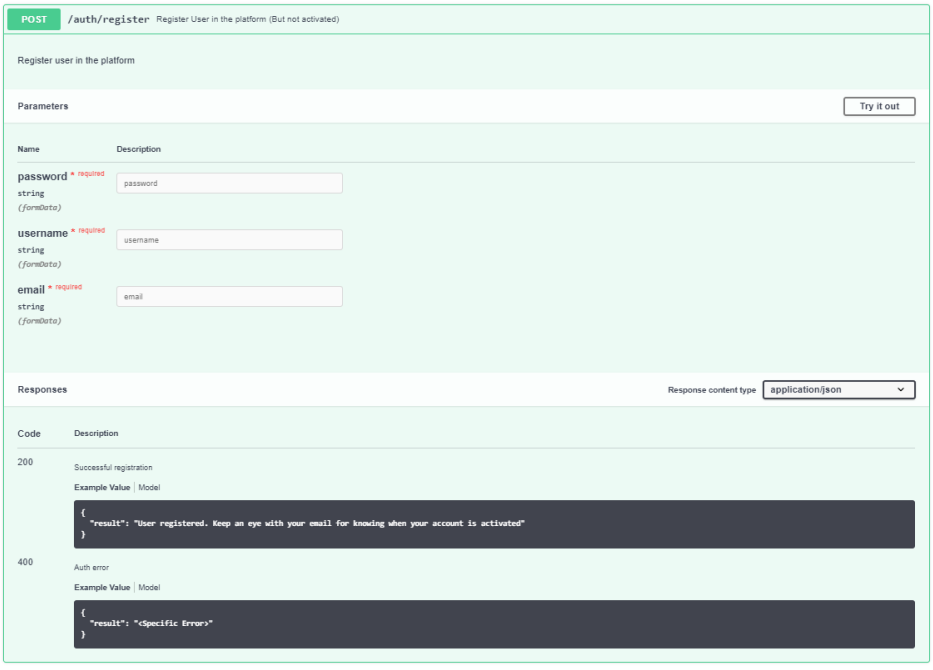


Figure 8: Open APIs register user specification

Password change

The user will be able to change their password if required. This request requires a Basic Auth and the new password that the user wants to change. This sequence diagram explains the flow of this process.

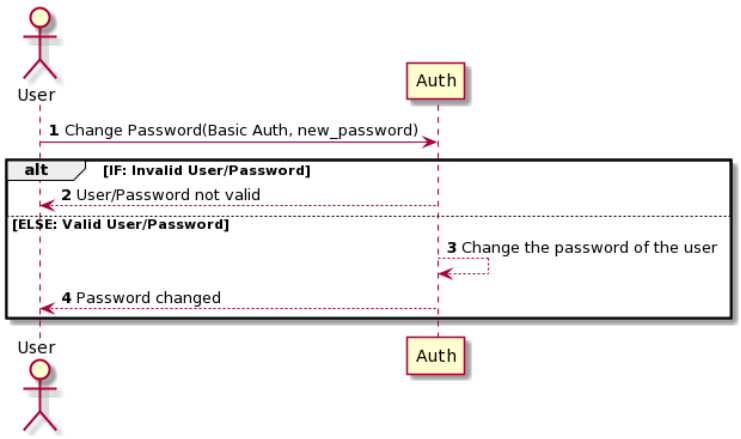


Figure 9: User password change sequence diagram

The request the user will do through the Dispatcher is the following:

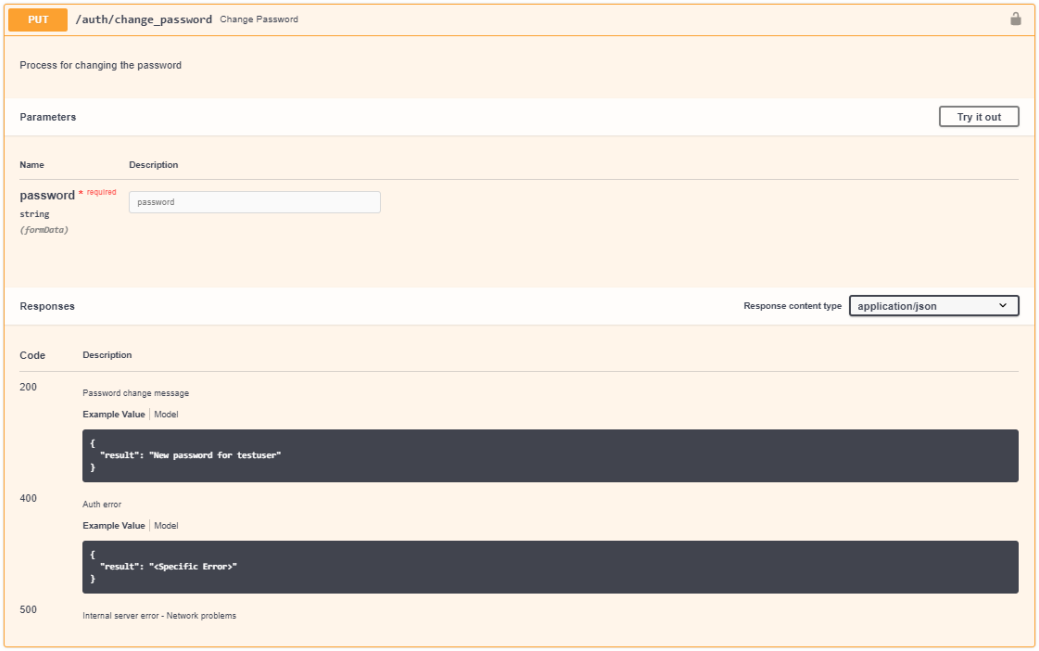


Figure 10: Open APIs change password specification

As we can see in Figure 10, there is a lock up at the top right side indicating the Auth requirement by Basic Auth. And this basic auth will be very frequent in the following requests. If the lock is clicked a popup appears with the Basic Auth form.

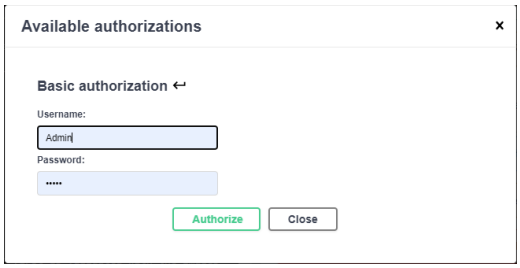


Figure 11: Basic Auth form

Get Token

One of the main utilities in this module is the creation of tokens to obtain an authorized key to request in the system. The utilization of the tokens means that it is not necessary to send every time the basic credentials (User/Password). However, to obtain a token it is necessary a Basic Auth to authorize the user to use the platform with their account permissions. This token needs to be stored by the user and used in the following requests.

The token has a time-life, so it is an authorization way to access in the resources that does not compromise the security of the user and its password and the access is not guarantee along the time due to the token revocation.

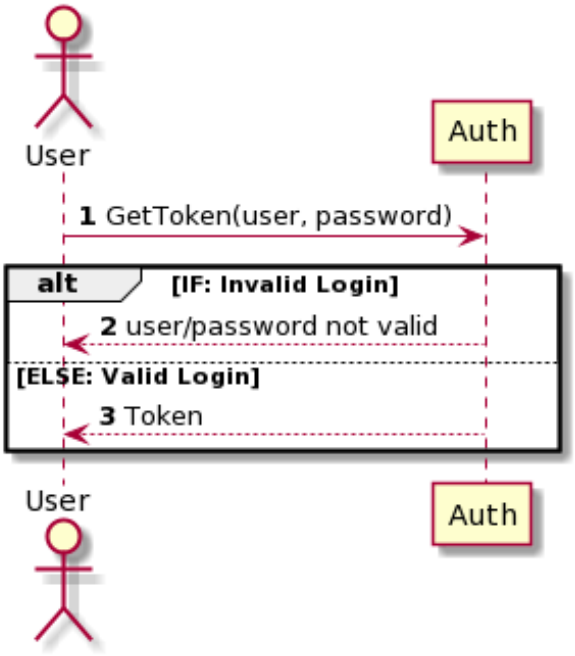


Figure 12: Get token sequence diagram

The request the user will do through the Dispatcher is the following:

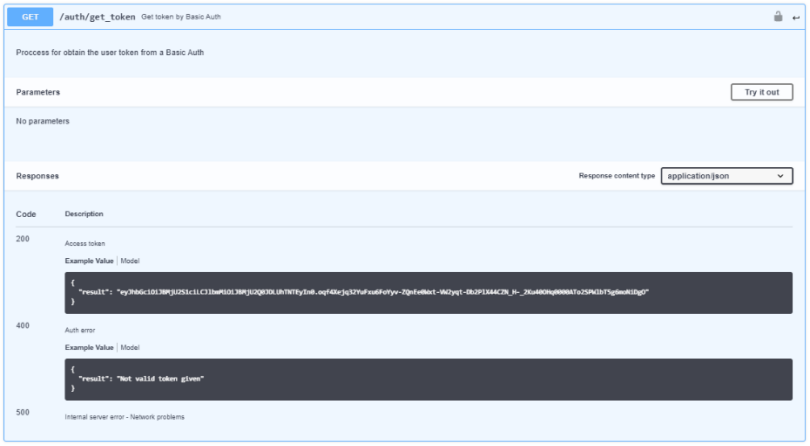


Figure 13: Open APIs get token specification

Recover Password

If a user does not remember their password, the platform will be able to change the user's password and send the new password to the user by email. To this request, requires the email

registered in the system. After the user receives the new password they will be able to change with the change password request.

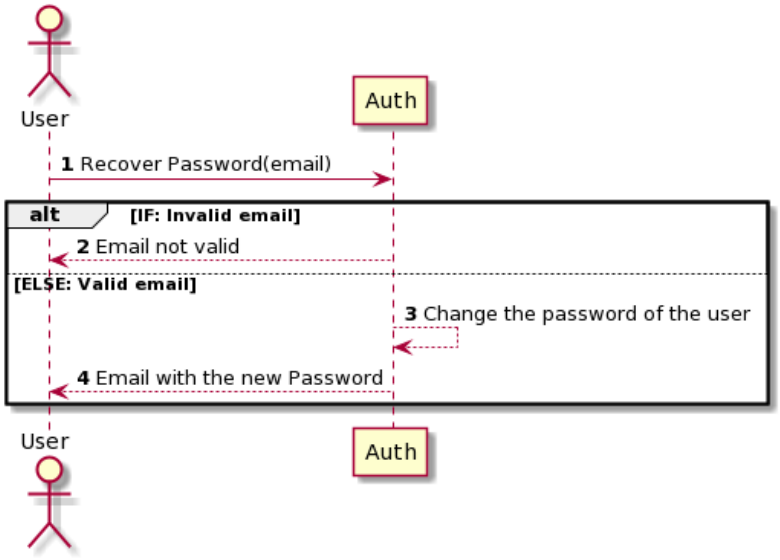


Figure 14: Recover password sequence Diagram

The request the user will do through the Dispatcher is the following:

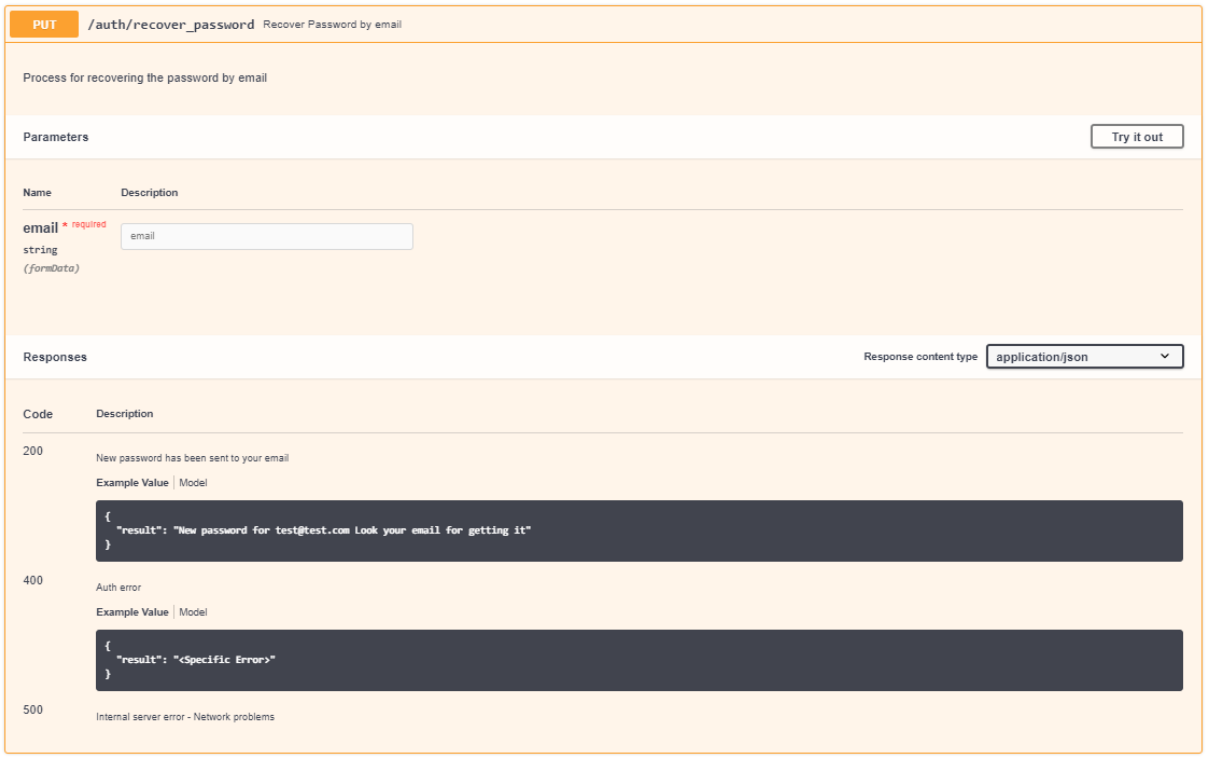


Figure 15: Open APIs recover password specification

The following sequence diagrams are design to the Administrator of the facility in order to manage the Authorization in the platform. The Admin need to request with Basic Auth to force an admin reverification:

Show registered users in the system

The Admin can view all the current users in the platform using the show function in the system. If the Admin wants to use this request, a Basic Auth is required. This request can show the logs of the users in the system if the verbose field is enabled in the request.

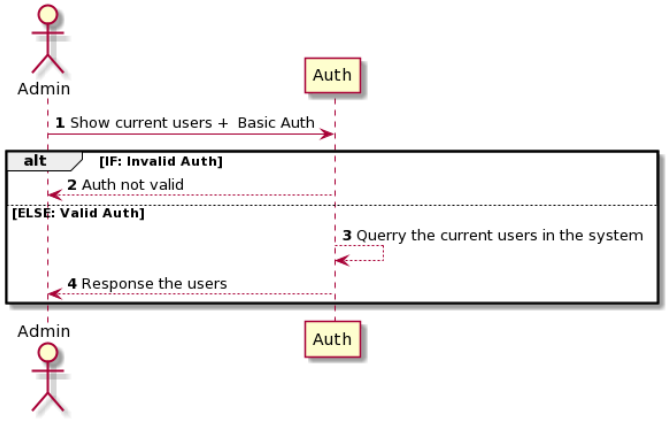


Figure 16: Show users sequence Diagram

The request the admin will do through the Dispatcher is the following:

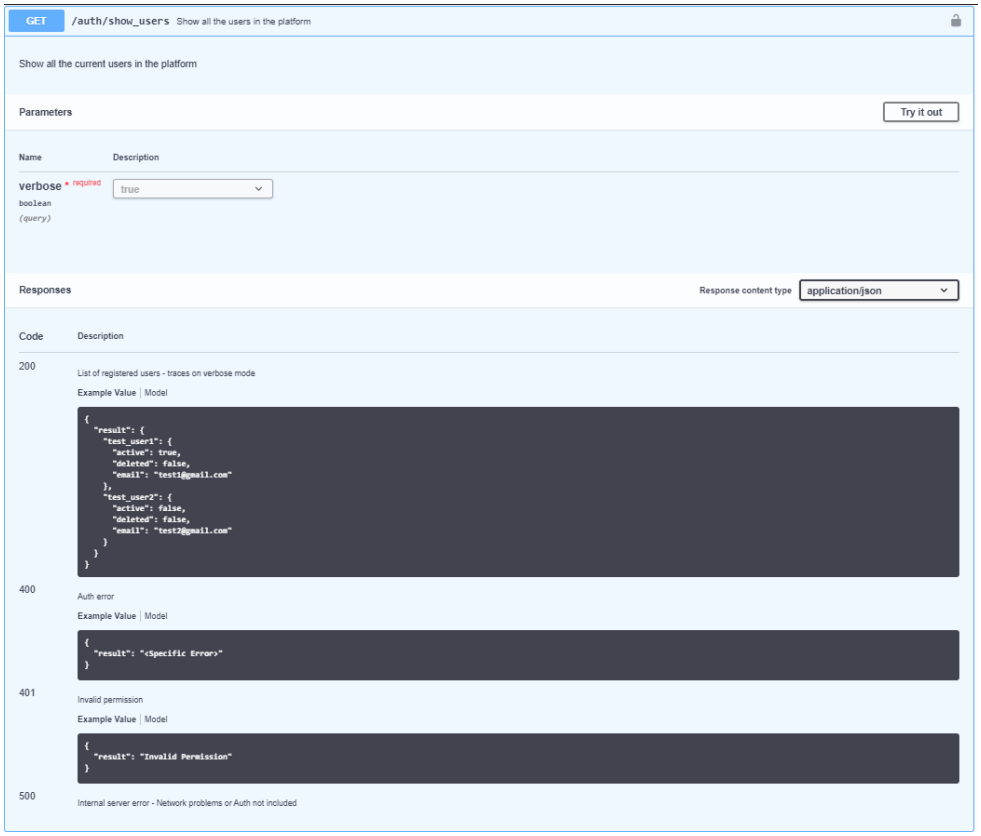


Figure 17: Open APIs show users specification

Drop user database

The Admin can drop the user database to deny access to everyone in the platform. For this request the admin basic auth is required.

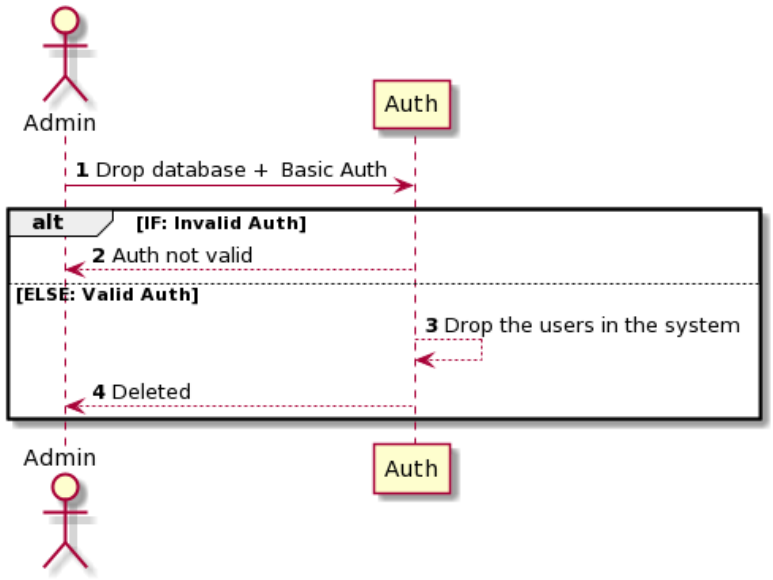


Figure 18: Drop DB sequence diagram

The request the admin will do through the Dispatcher is the following:

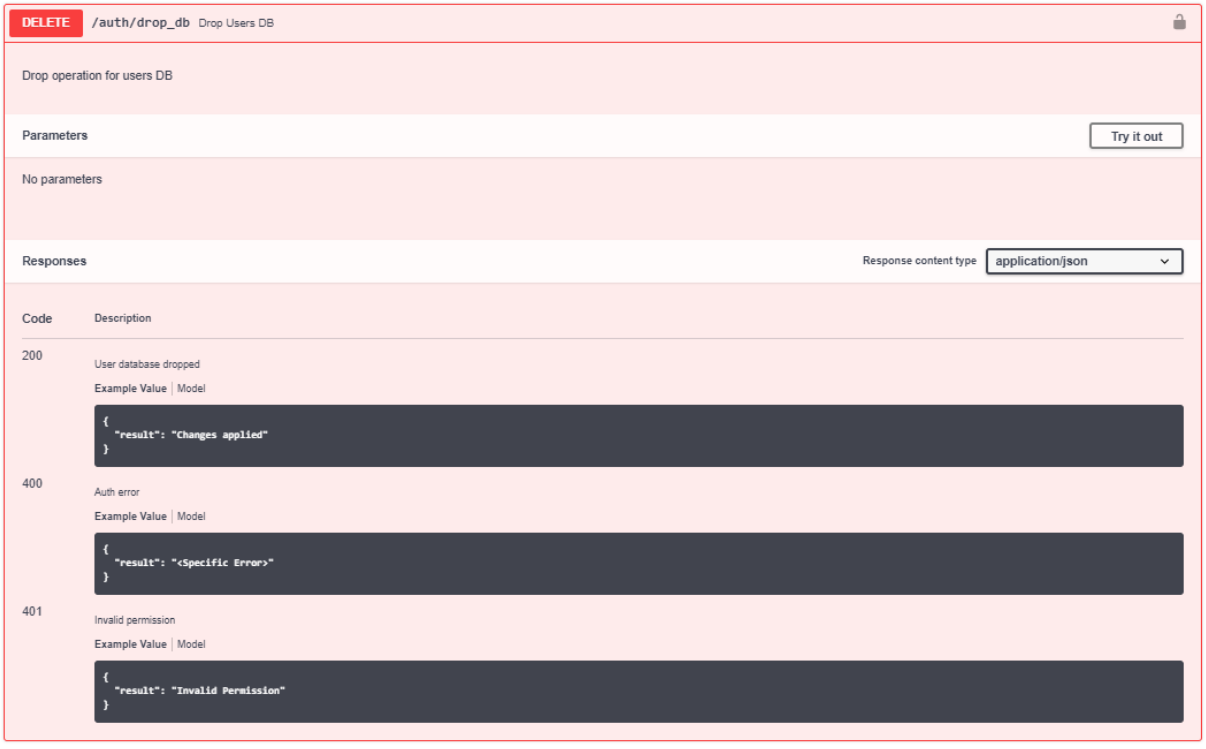


Figure 19: Open APIs drop DB specification

Delete a single user from the database

The Admin can delete a single user in the platform, denying the user to use the platform. For this request the admin basic auth is required.

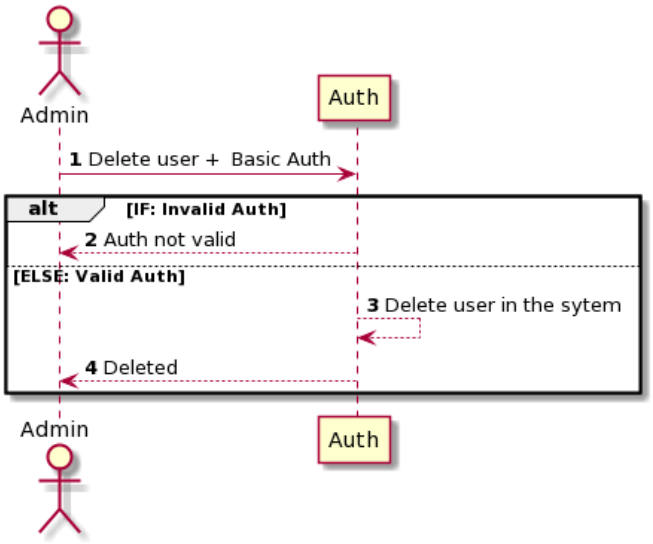


Figure 20: Delete user sequence diagram

The request the admin will do through the Dispatcher is the following:

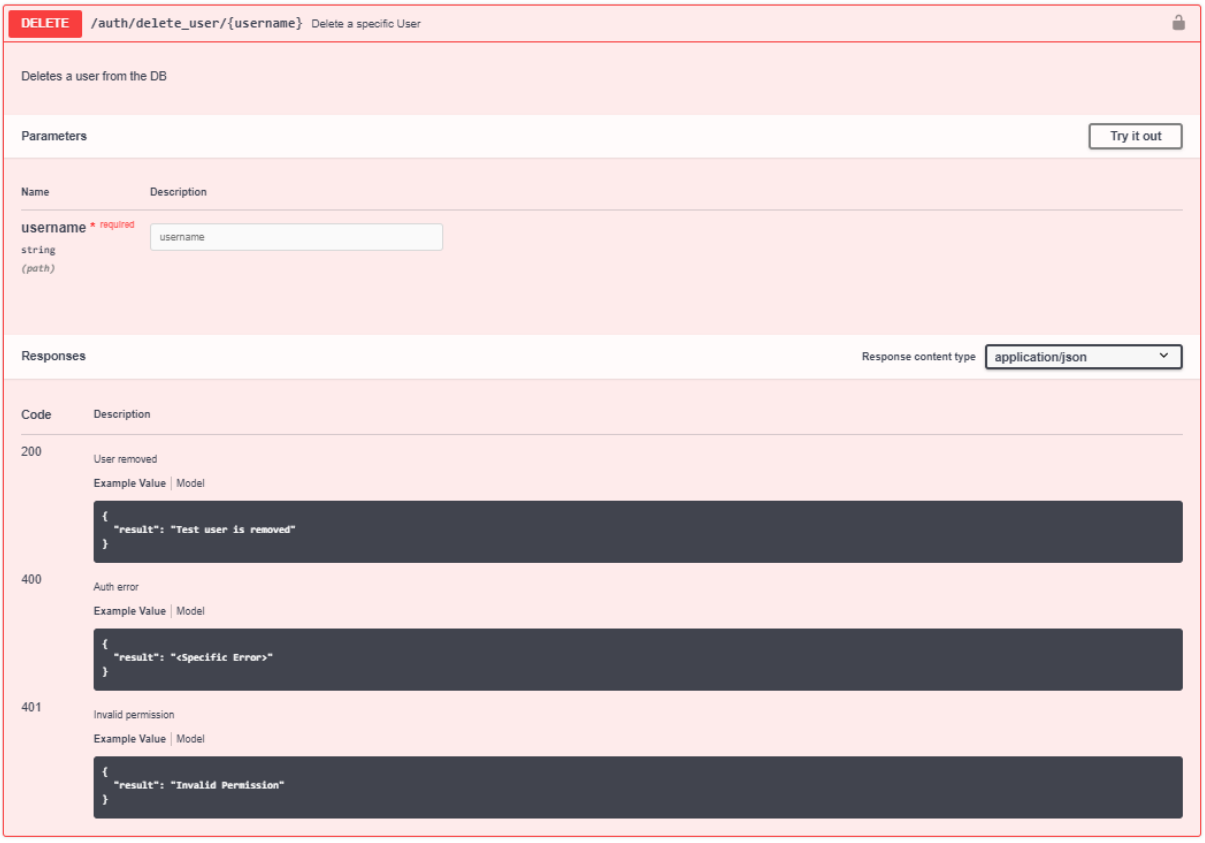


Figure 21: Open APIs delete user specification

Activate a single user from the database

The Admin should activate the users accounts registered in the system. Once a user is activated in the system, the user can use the platform.

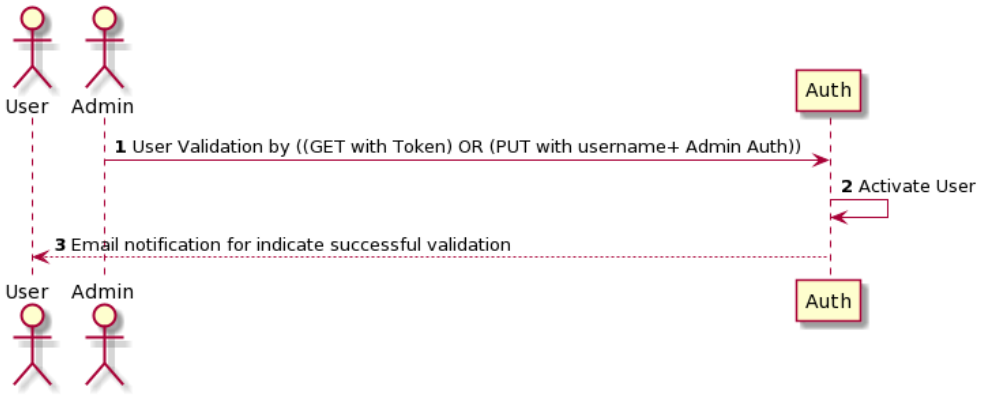


Figure 22: Activate a user sequence diagram

The request the admin will do through the Dispatcher is the following:

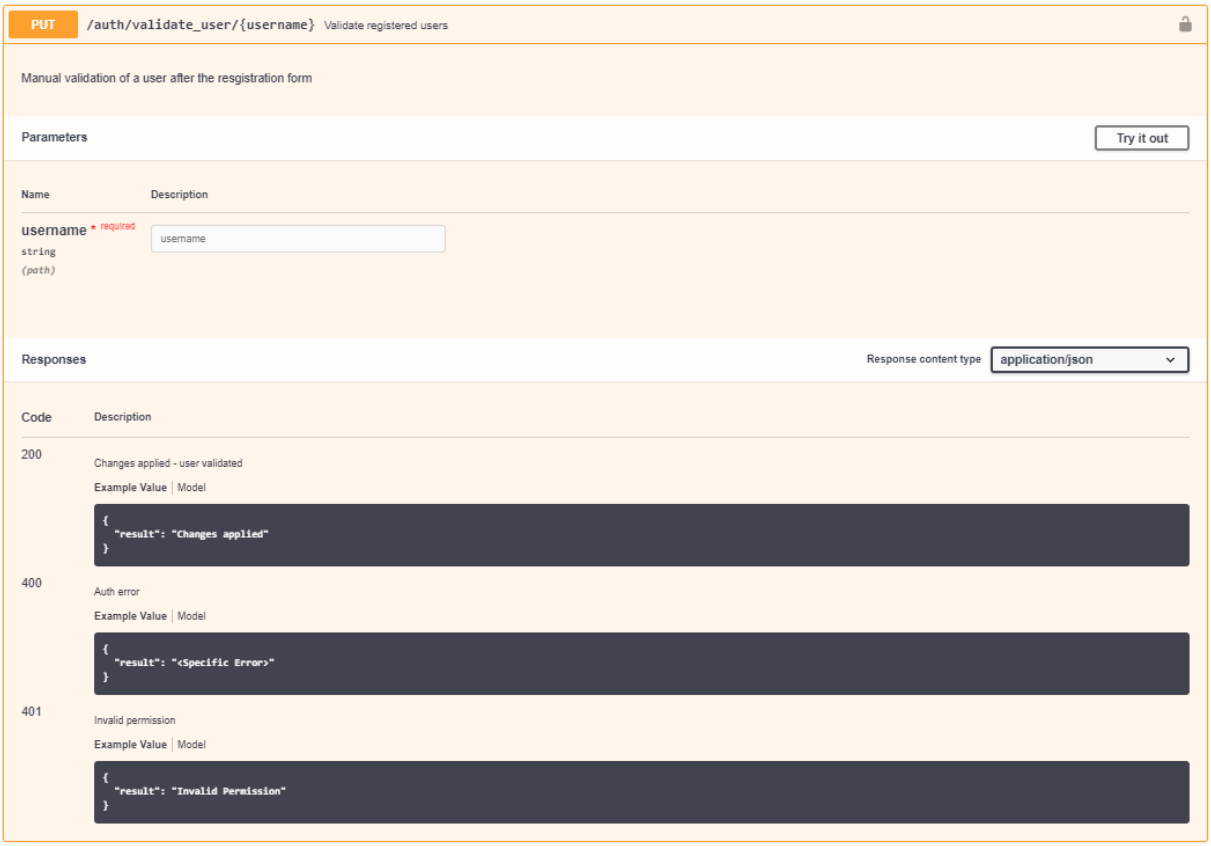


Figure 23: Open APIs validate user specification

Register platform in platform (In a Dispatcher environment)

The Admin of one platform can allow a remote platform to use their infrastructure to execute a distributed experiment. In this scenario, multiple concepts are required in this flow, like the service token. To explain this request, it is necessary to consider two platforms “Platform X” and “Platform Y”, and a need “Platform X have to been able to communicate with the platform Y”. The Admin platform of Y must allow the usage received from the platform X. And the Admin of platform X must to allow the registration of the platform Y in the platform X. The communication between platforms must be secured, for this reason a service token is required to guarantee the use of authorized platforms.

The service token is composed by the platform name and the platformID, that is universally unique identifier (UUID) that is created in the installation phase. Those parameters are encrypted in the Auth module to create a service token that can be an authorization entry point for external access.

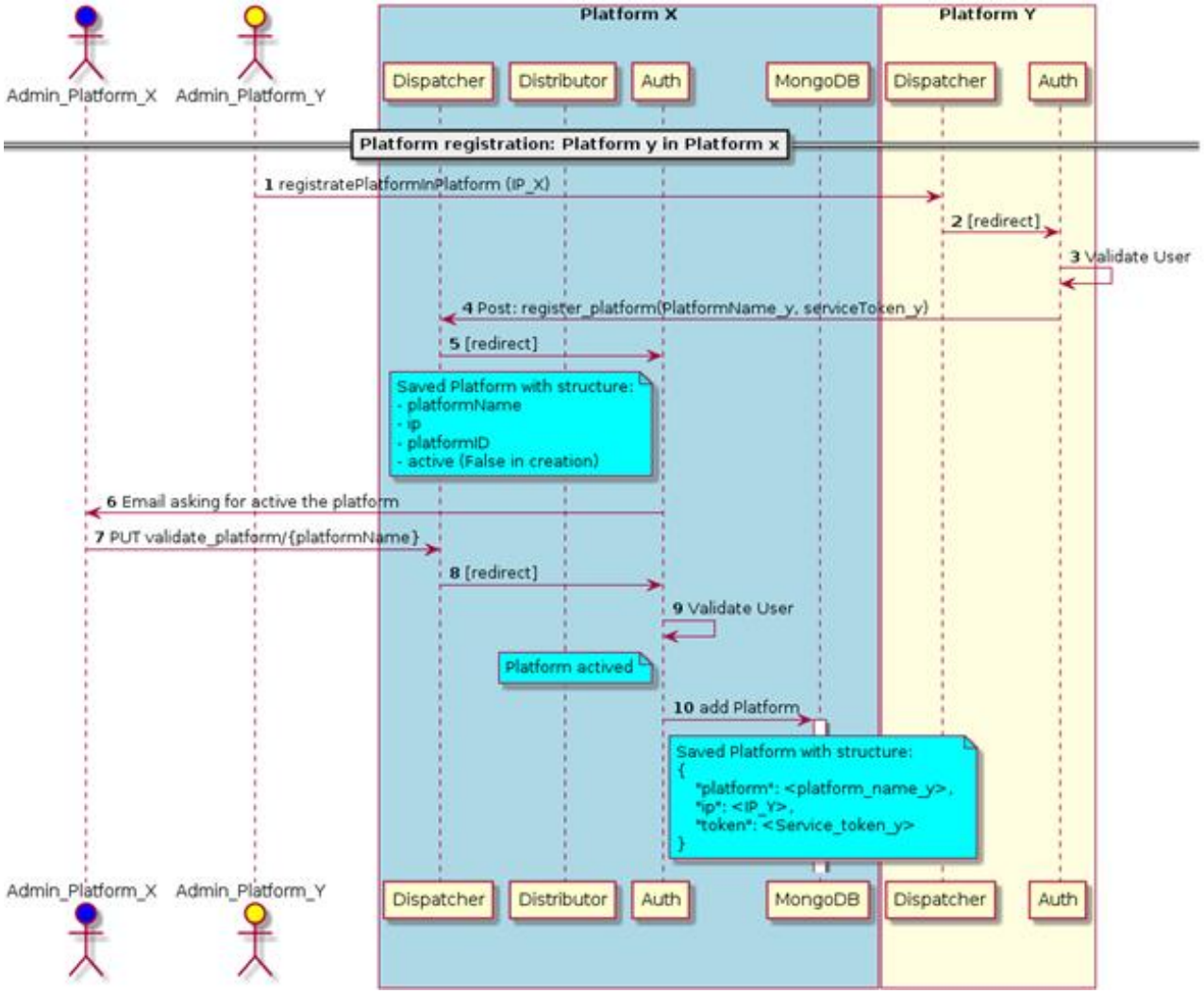


Figure 24: Platform registration sequence diagram

The request the Admin will do through the Dispatcher is the following:

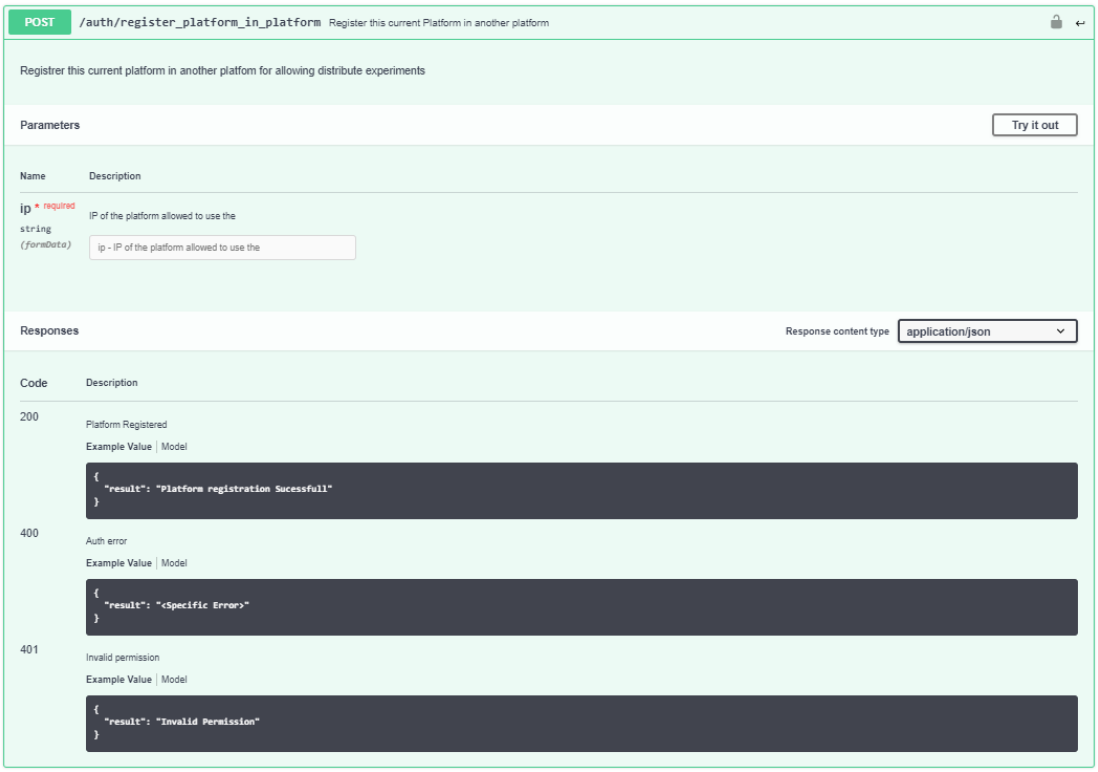


Figure 25: Open APIs register platform specification

Validate platform

After the register_platform_in_platform request is sent to the remote platform, the remote Admin will need to validate the platform in the system, the following action is required:

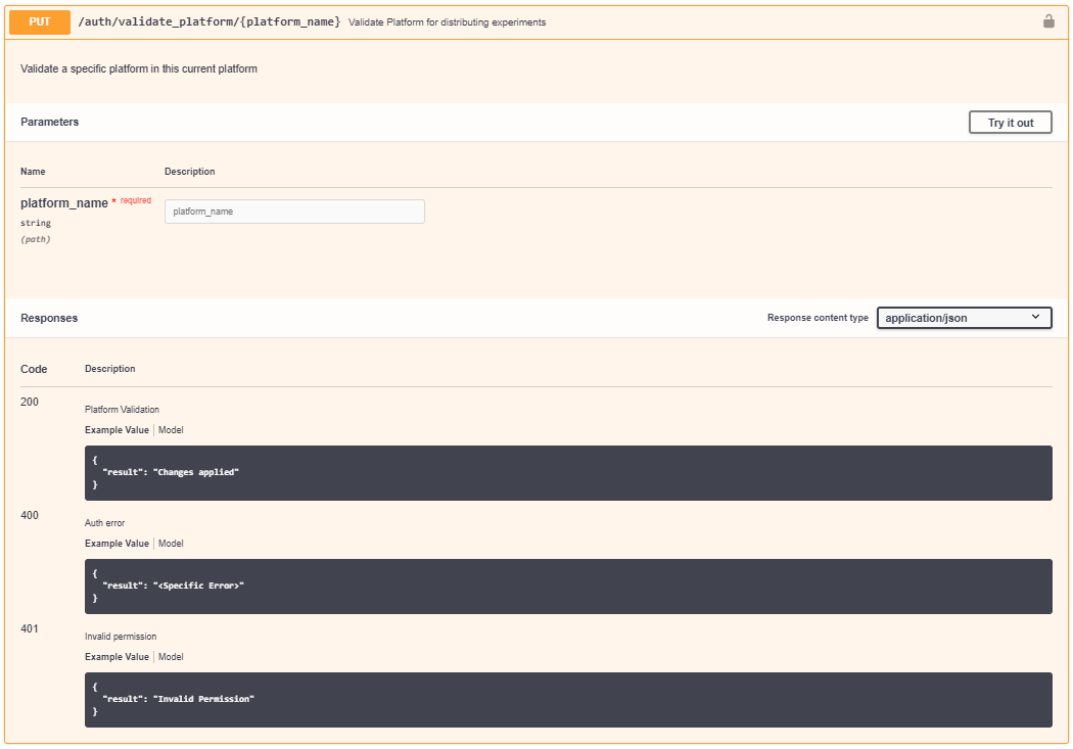


Figure 26: Open APIs validate platform specification

Show platform

It is possible to list all the available platforms registered. The validated and the not validated one. The request is the following:

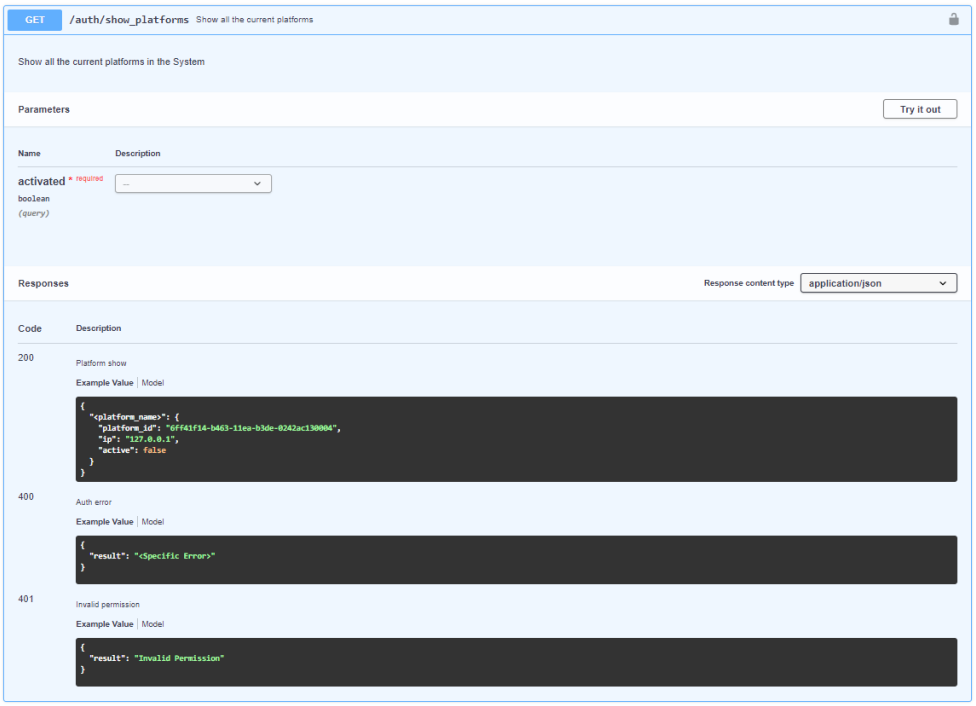


Figure 27: Show platforms

Delete platform

The Admin is able to delete the registered platforms, to not been used in distributed experiments, with the following request:

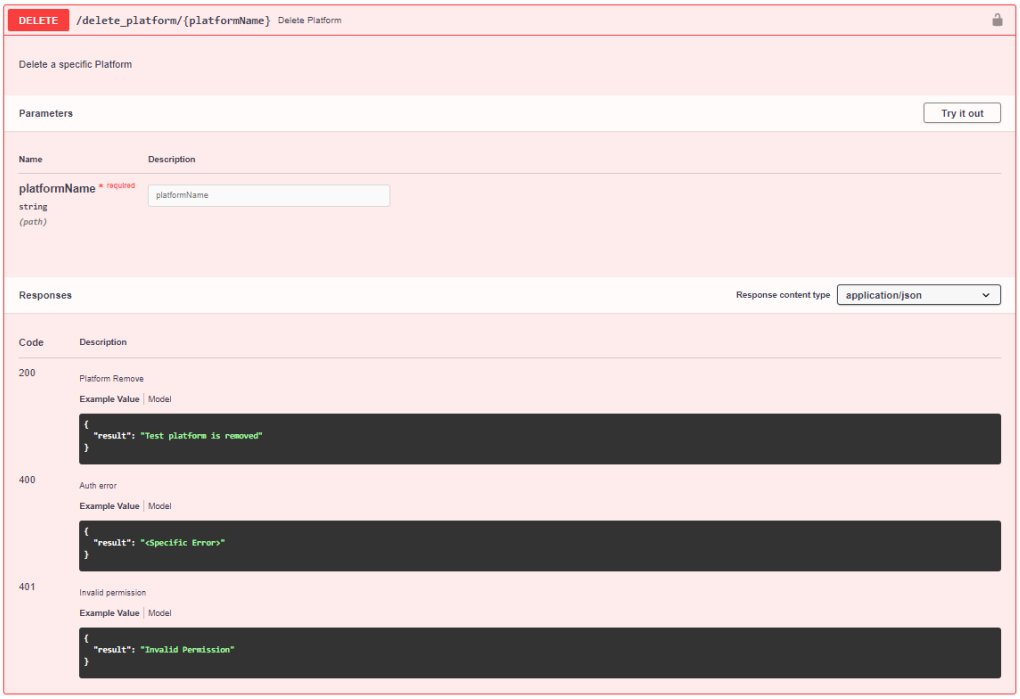


Figure 28: Open APIs delete platform specification

4.2. Mano Wrapper

The Dispatcher has the Mano interfaces embedded in charge of managing the VIM , NFVO and repository components. The Mano interfaces satisfies the CRUD operations required to manage and index the required artifacts (images, VNFs & NSs).

Open APIs Specifications

The API specifications are described and available as a client in the swagger service exposed in 5002 port. The Dispatcher has the following interfaces to be redirected to the MANO Wrapper component.

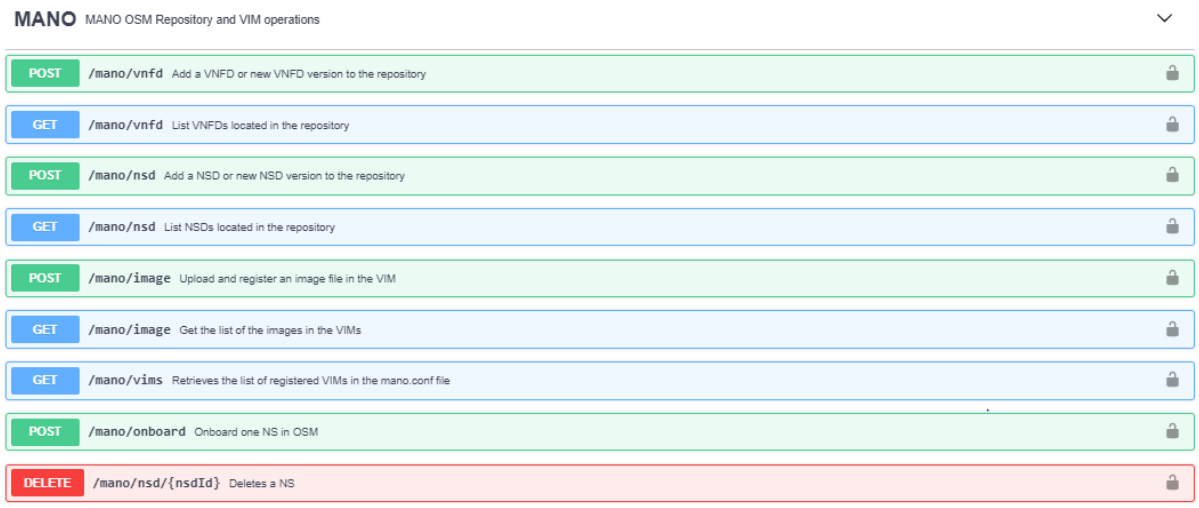


Figure 29: Open APIs Mano interfaces

The Open API specification is properly defined in the Open APIs description file. All of them are secured by the auth module and it requires a validated account to be used. There are endpoints to upload images and show the available resources in the VIM, index, des-index VNFs and NSs packages and retrieve the packages indexed in the repository.

4.3. Distributor

The Distributor is the component in charge of evaluating the Experiment Descriptor (ED) in order to identify whether the execution involves other 5GENESIS platform in the Experiment. Moreover, it distributes the experiments, by connecting them through the Dispatcher, where two platforms are involved in the execution of a defined experiment.

In addition, the Distributor is in charge also of authorizing the users to retrieve their experiments results and information. Facing those challenges, the Distributor is required to provide a solution over those issues. In this module, once a experiment has been created, the experiment ID is saved in the MongoDB Database with the username that created it in a pair form (experimentID, username). This correlation is necessary to allow the experiment owner to request and obtained the status of the experiment, cancel it or retrieve its results. All the experiments are considered private, just the experimenter who created the experiment can view and do operations over it.

4.4. ELCM

The interface oversees the experiment execution through the ELCM. This component has the functionalities to execute experiments by the Experimenter and operates them during the execution in the 5GENESIS platform. However, the ELCM has no authorization mechanisms to reject the requests that consults experiments that belongs to one specific user, relying on the Dispatcher the authorization of the user to access the resources.

Open APIs interfaces

The ELCM component has multiple operations described available in the swagger service, that will be authorized by the dispatcher. The endpoints are described by the Open APIs.

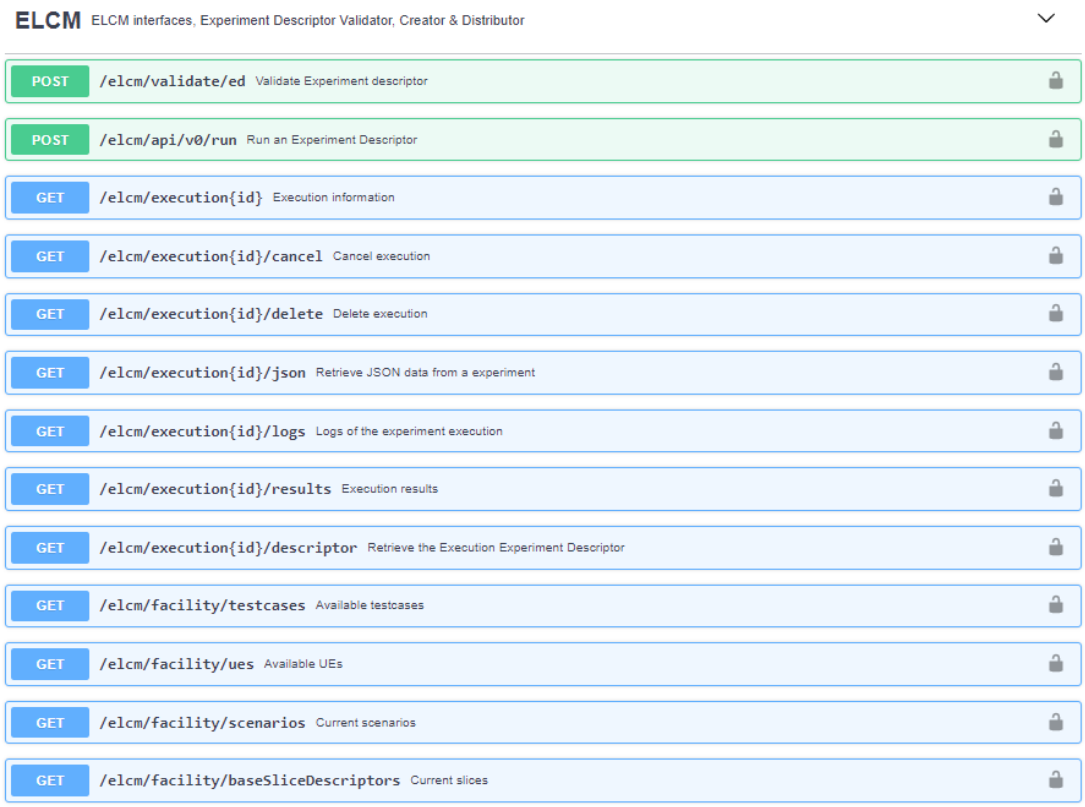


Figure 30: ELCM Open APIs interfaces

Those endpoints defined in the Open APIs specification are ELCM functionalities, except the first one, that is in charge of validating the ED and distribute the Remote Experiments, without synchronizing with the ELCM communication. Not all the previous requests are redirected to the ELCM directly, there are some endpoints that are redirected first to the Distributor module, that is in charge of distribute the required resources before the ELCM redirection.

There is a different redirection logic for the endpoints described. It is possible to split the requests interfaces in two types, the requests related to the CRUD experiments and the consult of the ELCM resources.

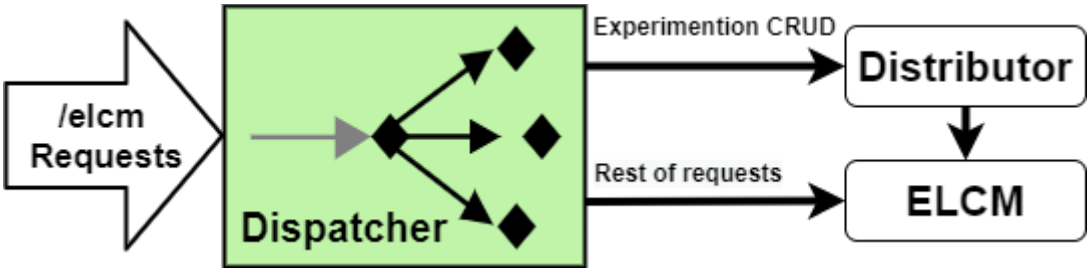


Figure 31: ELCM request split

The rest of the requests, out of experiments itself, are about the resources of the ELCM. Requests to retrieve the available slices, the different UEs, the defined scenarios and testcases.

- elcm/facility/baseSliceDescriptors
- elcm/facility/testcases
- elcm/facility/facilityUes
- elcm/facility/scenarios

From the experiment point of view, the experimentation CRUD operations are the following:

- elcm/api/v0/run
- elcm/execution/<id>
- elcm/execution/<id>/descriptor
- elcm/execution/<id>/logs
- elcm/execution/<id>/cancel
- elcm/execution/<id>/delete
- elcm/execution/<id>/json
- elcm/execution/<id>/results

The flow for running the experiment, the first request in the list, perform a different sequence in the Distributor. Therefore, it is possible to split again between the first request, '*elcm/api/v0/run*', and the rest that have the structure '*elcm/execution/<id>*'. The first one is in charge of the experiment creation and it requires a specific flow, as a prior validation of the ED and dependencies validation is required.

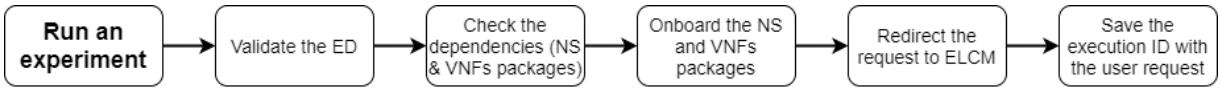


Figure 32: ELCM Run Experiment flow

During the Experiment flow, it is needed to validate the experiment descriptor; check the dependencies as the packages and the used images by the VNFs, if they are uploaded in the location indicated by the Experiment Descriptor; and onboard the NS required in the execution.

It is important to have into account that during the running experimentation request, the execution ID will be paired with the user that has requested the experimentation execution. This correlation will be saved in the mongo database. That user will be required to execute the rest of the CRUD requests that affects the experiment that they have created.

Also, another thing to have into consideration to enable "Run an experiment flow" is that the required NS selected during the experiment definition must be public or own by the experimenter. This information is stored in the NS Repository of the 5GENESIS platform, and is crosschecked during the validation of the ED to enable the execution of the experiment or rejected.

To execute the experiment CRUD operations, the flow is defined as the follow:

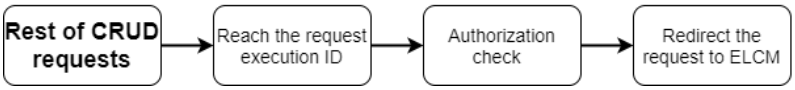


Figure 33: ELCM CRUD Experiment flow

In each CRUD request not involved in the creation of experiment there is a flow to authorize the user by the Distributor in which, if the user has already created the experiment, and redirect the request to the ELCM. Those requests start with the following path *"elcm/execution/*<id>*"*.

ED validation

The Experiment Descriptor (ED) is a JSON file with all the required information to execute an experiment by the ELCM. This descriptor is validated with a JSON schema in a semantic and syntactic manner.

```
{
 "Application": Optional[str],
 "Automated": bool,
 "ExclusiveExecution": bool,
 "ExperimentType": str,
 "Extra": Dict[str, str] (may be empty),
 "NSs": List[Tuple[str, str]] ((nsd id, vim location) may be empty),
 "Parameters": Dict[str, str] (may be empty),
 "Remote": Optional[str],
 "RemoteDescriptor": Optional[<Remote descriptor>],
 "ReservationTime": Optional[int],
 "Scenario": Optional[str],
 "Slice": Optional[str],
 "TestCases": List[str] (may be empty),
 "UEs": List[str] (may be empty),
 "Version": str
}
```

Figure 34: Experiment Descriptor

Once the descriptor is validated, it is required to verify that all the resources defined in the descriptor are available in the platform. These resources are the UEs, the Slice, the Scenario and the test cases form the ELCM resources, and the Network Services descriptors (NSD, VNFD) and all the VIM images that each VNF could require to instantiate the services from the MANO resources.

The UEs, Slices, Scenarios and testcases are retrieved from the ELCM. The validation is about to request the resources at the ELCM and verify that the experiment is able to be executed in the ELCM.

Another interesting point is the NSs parameter, in the validation it is needed to verify that the NS exists in the repository and also that the images used by the VNFs must be in the VIM location indicated.

Experiment Execution and NS onboarding

At this point, the 5GENESIS repository is full of NSs & VNFs. To keep the consistency throughout the facility, the necessary network services with its dependencies are onboarded onto the NFVO only after analysing the Experiment Descriptor. In this way, exclusively the required components of the NS will be onboarded instead of the whole set, making the process more effective and optimizing the resources in the platform, prioritizing them for the execution of the experiments. More importantly, this process is transparent to the user and to the rest of the system modules.

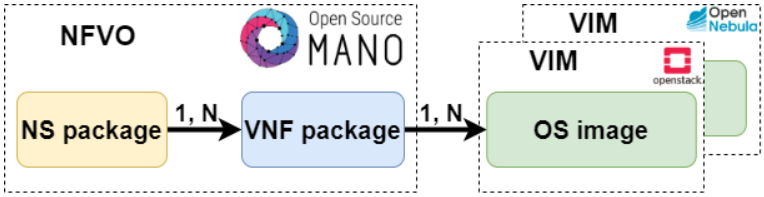


Figure 35: Packages needed to be onboarded

Once the ED is validated with the required dependencies for the execution of the experiment, it is then onboarded in the NFVO through the Distributor component, to be deployed once the instantiation request from the Slice Manager is received in the NFVO.

Experiment Distribution

The experiment distribution from the Dispatcher side is in charge of the Distributor component. Once the platform has been registered, it is possible to request on it for distributed experiment purposes.

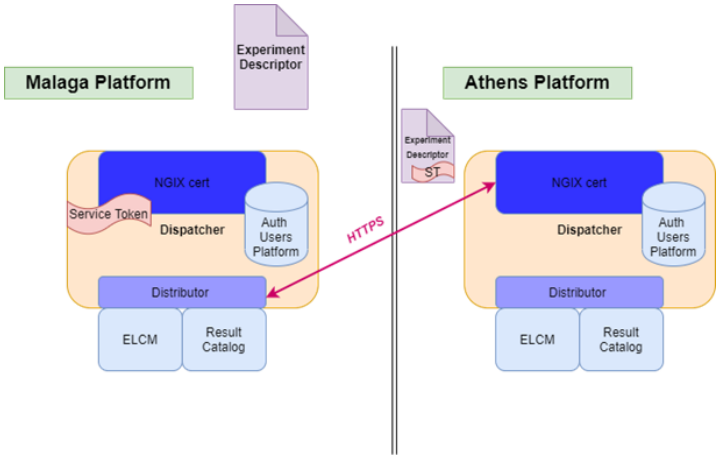


Figure 36: Interplatform experiment Architecture

The Distributor component is able to redirect the experiment to another platform in a secure way through HTTPS and a Service Token to be authorize.

The Distributor realizes the same actions of the standalone experimentation adding a pairing flow to link the two experimentation platforms. It is needed to validate and verify the experiment, then send the experiment to the ELCM. Each ELCM must know the execution id of the distributed experiment. The flow is defined is illustrated below in Figure 37:

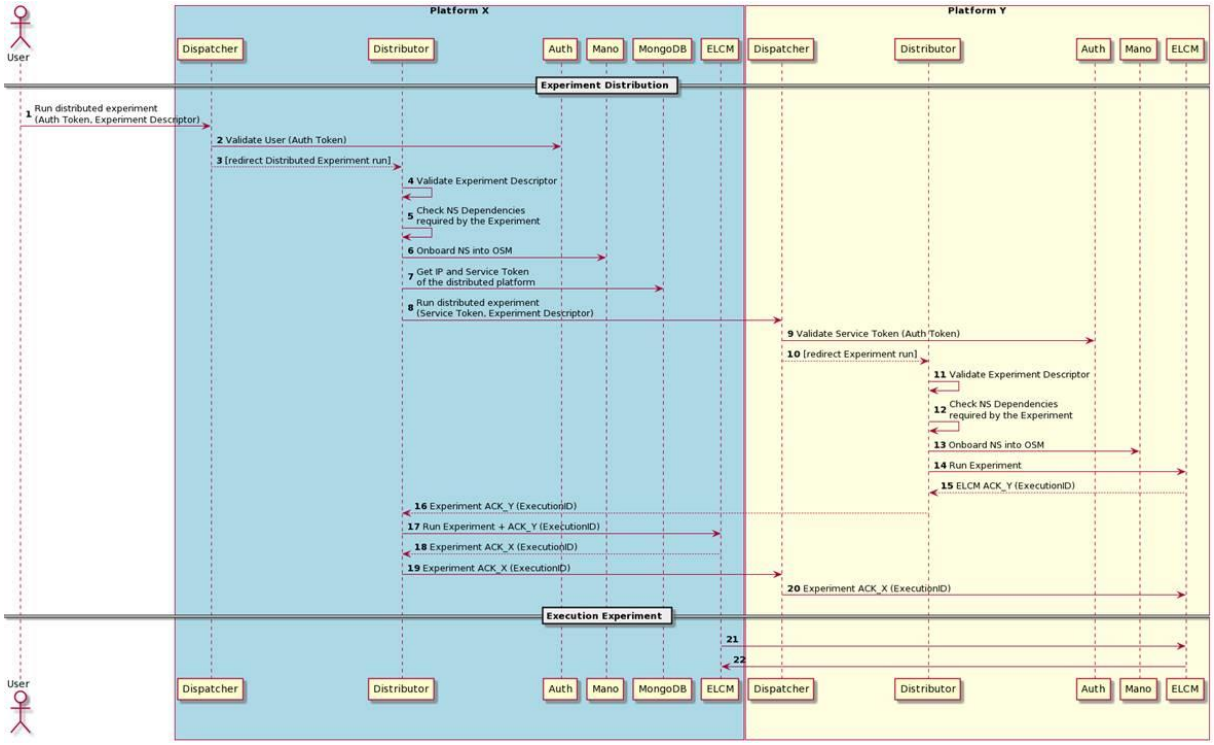


Figure 37: Experiment Distribution

The following picture (Figure 38) depicts the interaction flow when the Experimenter request the execution of the distributed experiment and the steps to be followed in the communication with the remote platform.

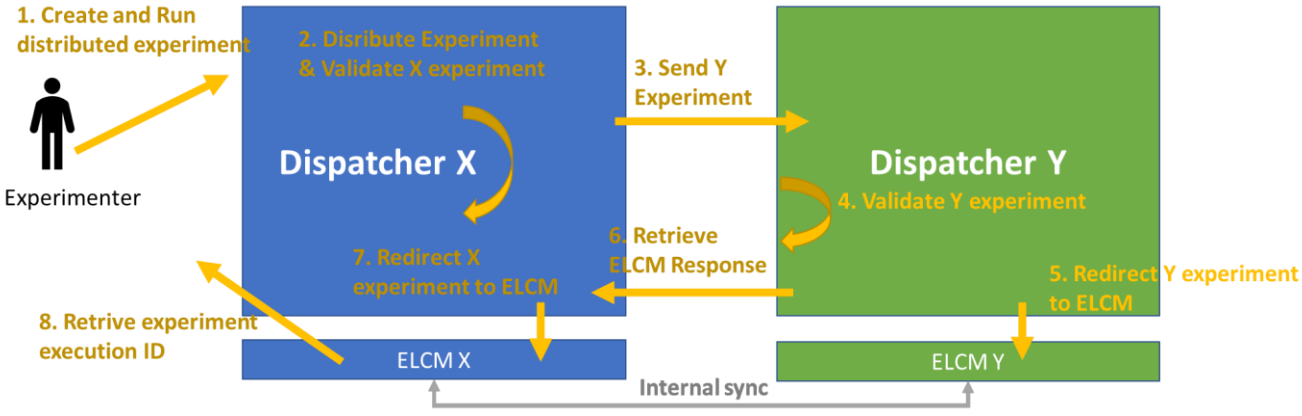


Figure 38: Experiment distribution flow

4.5. Result Catalog

The experimentation results will be provided by the analytics components through the Dispatcher to provide the security layer in the retrieve experiments. This component will retrieve the experimentation results of the experimentation in a raw-data and refined-data form. The Dispatcher authorizes the users to retrieve their own experiments results. This logic is realized by the Distributor component, and the endpoints are available from the Dispatcher through the Open APIs interfaces.

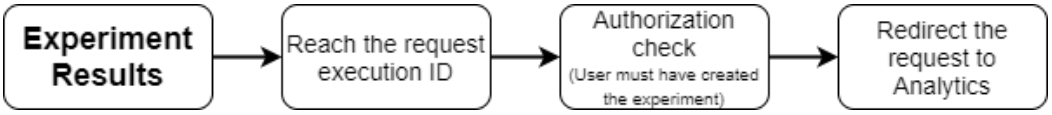
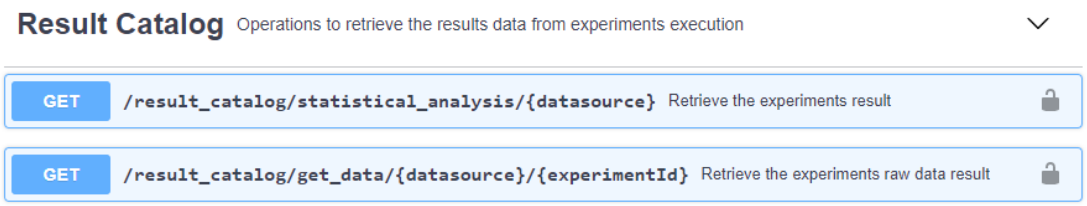


Figure 39: Experiments results flow

Open APIs interfaces

The interfaces to retrieve experiments are the following must follow the Open API specification described before.



The request to obtain the refined data comes from the statistical container from the analytics module. The interface is the following:

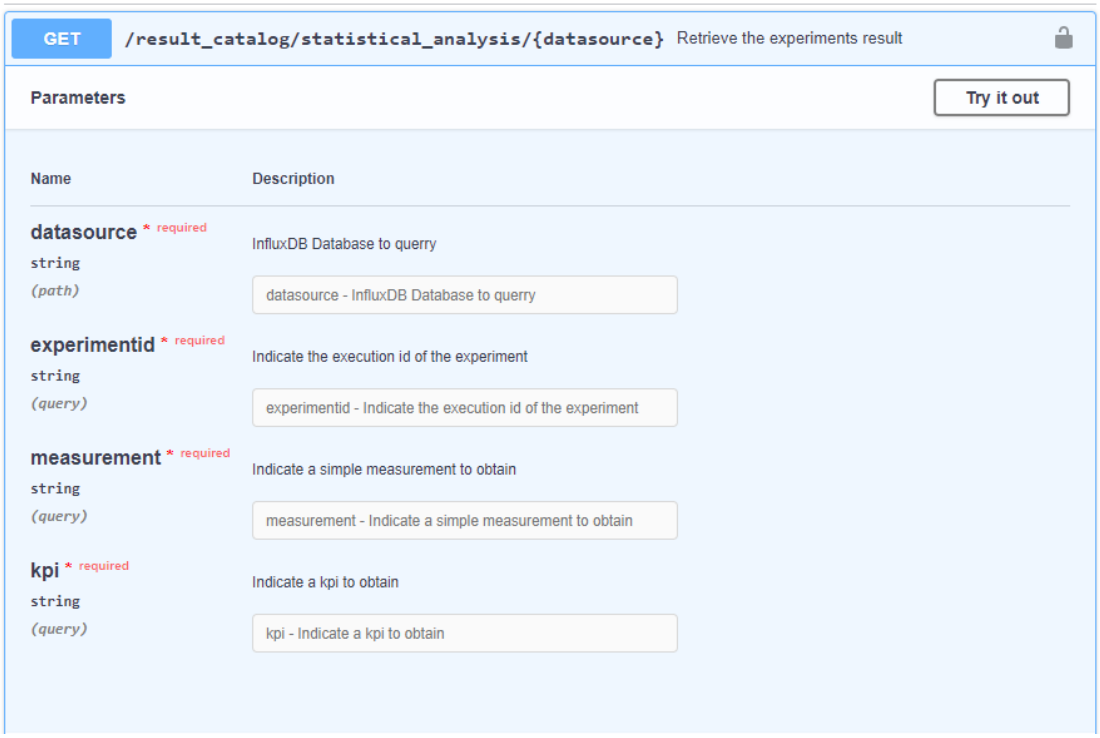


Figure 40: Retrieve experiment results interface through the Open API specification

The available responses could be the experiment results based on the request made by the user, or if the use is not authorized or the request is not found in the system.

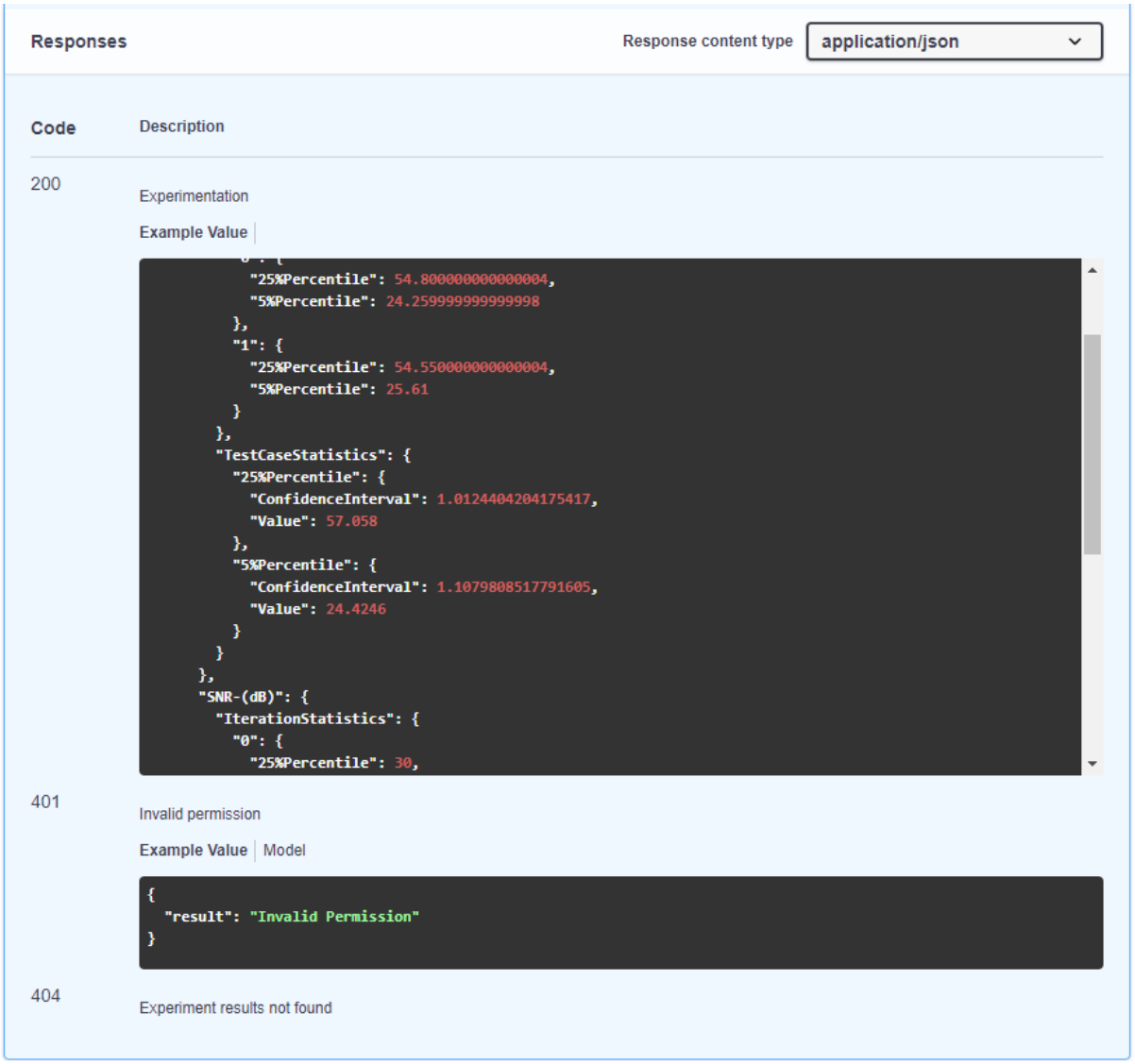


Figure 41: Retrieve experiment results response through the Open API specification

To obtain the raw data the following request is required.

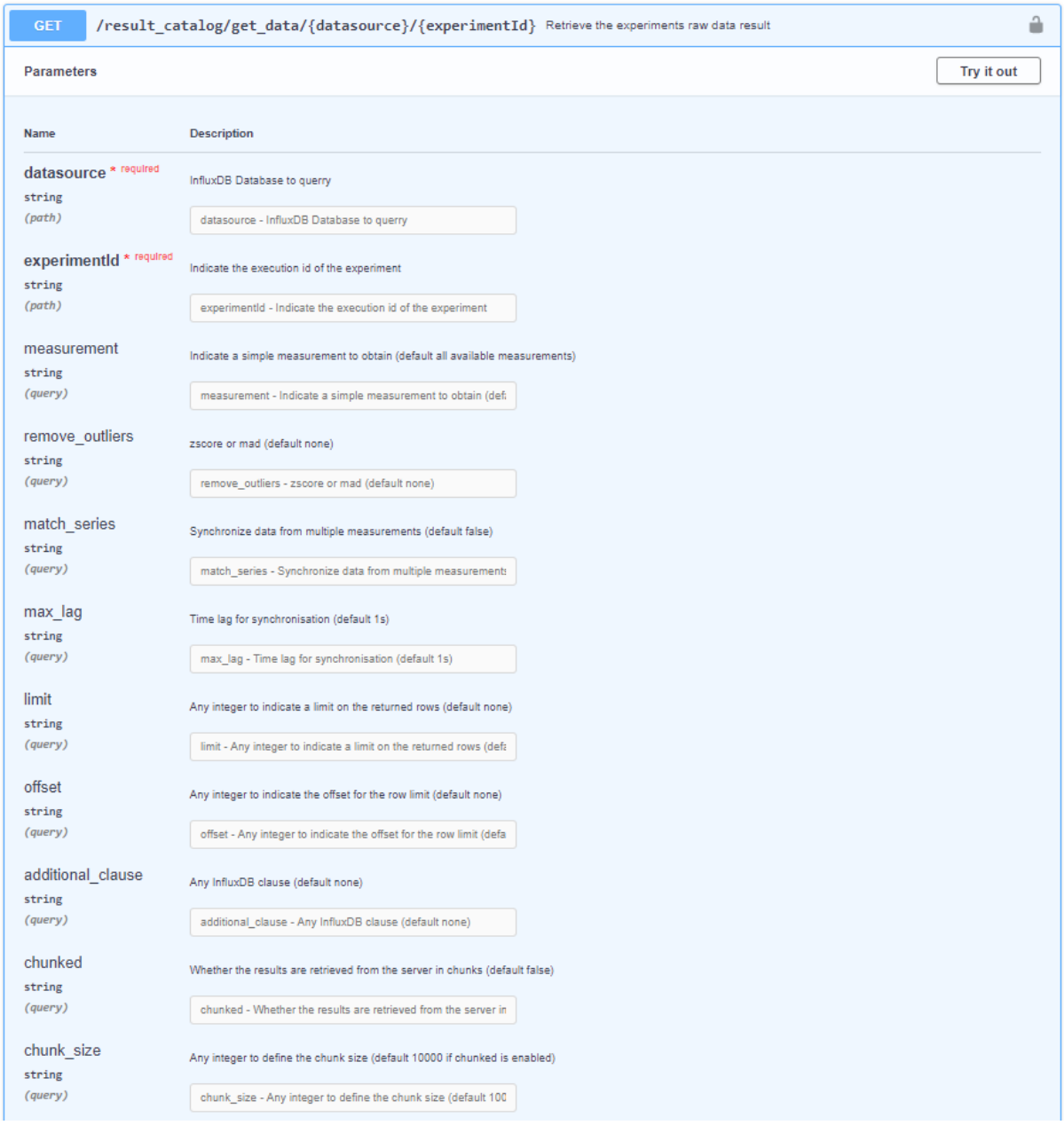


Figure 42: Raw data interface

The raw data interface returns the values used by the analytics components to plot graphs and calculate statistical operations. The returned values will be different depending of the filters showed in the previous Figure 42.

5. INSTALLATION & RUN

The following subsections detail the process of installing the Dispatcher from the 5GENESIS public repository [https://github.com/5genesis], including requirements, explanations, configuration, commands, etc.

5.1. Requirements

The machine (virtualised of not) where the Dispatcher is to be installed needs some previous software installed locally or remote but accessible for installing and running the 5Genesis Dispatcher, e.g., the NVFO or the ELCM services do not need to be installed on the same machine but have to http reachable from it. The physical requirements are the following [5]:

- 2 Cores
- 4GB of RAM
- 40GB of Disk

For installing and running the 5Genesis Dispatcher, you will need:

1. docker version >= 18.09.6
2. docker-compose version >= 1.17.1
3. NFVO + VIM +NS repository
4. ELCM
5. Analytics Container
6. Configuration files correctly filled up:
    - Dispatcher config file: *dispatcher.conf* [5]
    - MANO Wrapper module config [6]: configuration file (*mano.conf*) inside the *mano* folder

5.2. Pre-configuration and installation

The Dispatcher [5] needs to be configured properly before the containers are built. For that, a simplified configuration file is offered: *dispatcher.conf*, which will have to be edited and adapted. The file should contain information of all the modules the Dispatcher forwards information to (validator, mano, elcm, result_catalog, etc.) and how to do it. For each module, a new enabler will be added in the Dispatcher. It uses the following format:

```
[module_name]
PROTOCOL=[http|https]
HOST=x.x.x.x -> IP or DNS name of the host component
PORT=xxxx -> Port where the app API is available
PATH=/ -> Base path of the application ("/" by default)
```

The config file template already includes the *validator* and the *mano* modules as they are included within the *Dispatcher*. They are already configured and should not be touched.

During the installation process you will be asked for a couple of simple questions to re-verificate the configuration of the Dispatcher and the mano wrapper, and also some config for integration testing purposes.

Once edited properly, the configuration will be applied and the containers built, based on the config file we have just created (*dispatcher.conf*):

```
$ ./install.sh
```

Example:

```
[mano]
PROTOCOL=http
HOST=mano
PROTOCOL=5001
PATH=/
```

The start script will deploy and run the Dispatcher container, the Distributor, the MANO Wrapper and a Swagger environment to test the available features:

```
$ ./start.sh
```

- Dispatcher will be accessible through port 8082 through an SSL certificate, so HTTPS is required.
- Swagger environment will be accessible through port 5002.
- The SSL certificate is self-signed. It is not supervised by a Certification Authority because in this current moment is not possibly know where the platforms will be hosted.

For installing the certificate is required to open the internet navigator in https://<IP_OF_DISPATCHER>:8082

Something like this will appear, the web page format depends on the browser:

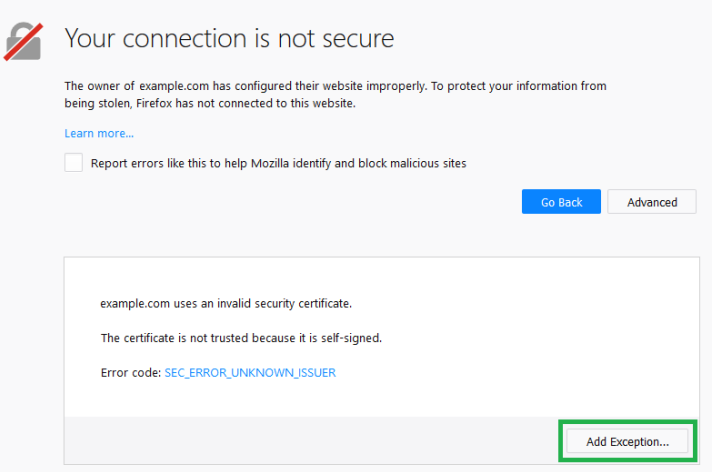


Figure 43: Allow self-signed certificate

You must add the exception/believe in the certificate or something like that. After this step, you can use the Dispatcher through SSL certification.

5.2.1. Dispatcher Manual

Once the installation is over, the Dispatcher with all the required components is up and running. The Dispatcher can manage and authorize every request though it reverse proxy.

All the possible requests available in the system are described in the Open APIs specification showed through an swagger client.

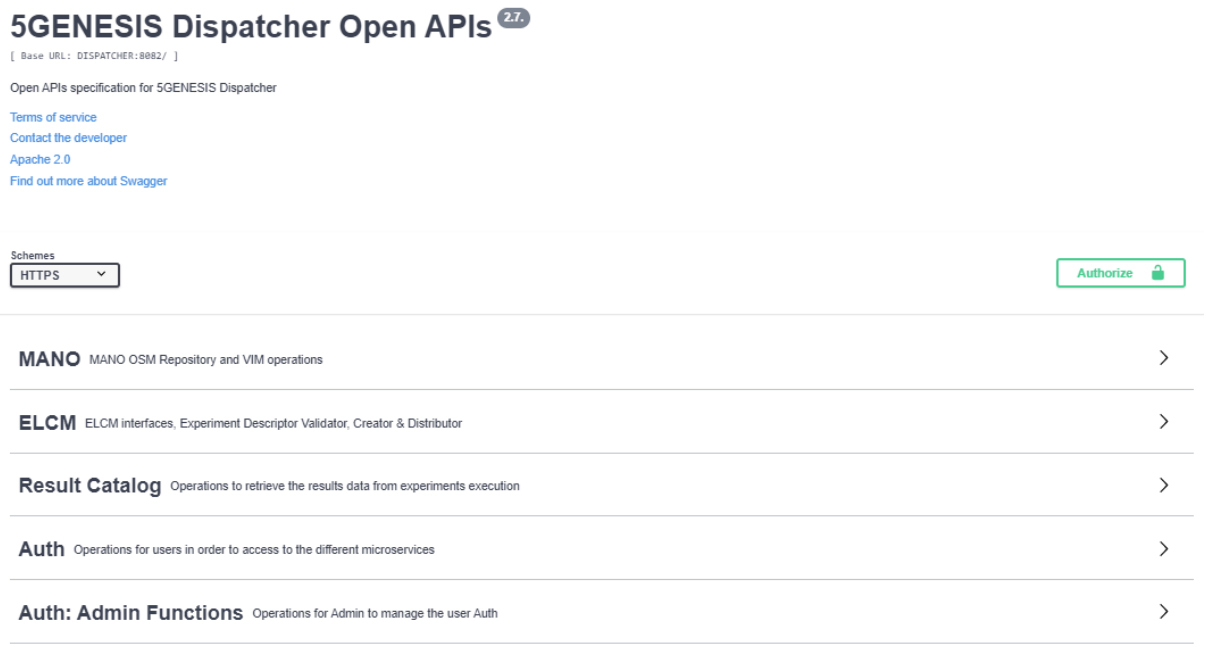


Figure 44: OPEN APIs interfaces

5.2.2. Auth module standalone installation

Although the *Authenticator* is installed along with the *Dispatcher*, it could be also installed in a standalone way to add a security layer to an external software. In the following section it is explained in deep the development over the *Authenticator* module, considering the dependencies, configuration, dockerisation and installation.

Requirements

The authorization module is developed in Python3. The required dependencies for building the project are the following:

```
jwcrypto==0.6.0
gevent==1.4.0
Flask==1.0.2
Flask-SQLAlchemy==2.1
requests==2.20.1
flask_mail==0.9.1
Flask-Cors==3.0.8
Flask-RESTful==0.3.7
pymongo==3.8.0
```

Module Structure

The module files are structured in the following way:

```
auth/                           Main Folder
│
├─ swagger/                     Swagger Folder
│   └──swagger.json             Open APIs Specification
│
├─ templates/                   Folder for different templates
│   ├─ recover.html             Recover password template
│   ├─ validate_platform.html   Validate platform template
│   └──validate_user.html       Validate users template
│
├─ auth.db                      SQL Database
├─ Auth.py                      Server
├─ auth_logic.py                Service logic
├─ auth_utils.py                Utils tools
├─ constants.py                 Constans file
├─ DB_Model.py                  Database Model
├─ DockerFile                   DockerFile for building the conatiner
├─ key.json                     Key for encrypt/desencrypt Tokens
├─ MailConfig.py                Mail config
├─ platform_name                Name of the platform, used in platform register
├─ platformID                   The platform ID autogenerated before the install
├─ requirements.txt             Python Dependencies
└─ settings.py                  Server settings
```

Configuration

For configuring the e-mail, the security token and some extra configuration, we can find the files below:

- Email config is defined in *MailConfig.py*
- *key.jso*n for encrypting and decrypting tokens.
- *Settings.py*
    - Setting the token timeout
    - Setting the suffix request for internal calls

Pre-requisites

For installing and running the 5Genesis *Auth* you will need:

- docker version >= 18.09.6
- docker-compose version >= 1.17.1
- MongoDB

Install & Run

Authorization is very easy to install and deploy in a Docker container. By default, the Docker will expose port 2000, so change this within the *Dockerfile* if necessary. When ready, simply use the *Dockerfile* to build the image.

```
cd auth
```

```
docker build -t auth .
```

This will create the *auth* image and pull in the necessary dependencies. Once done, run the Docker. For running the image and map the container port (2000) to whichever port you wish on your local host (2000 also in this case):

```
docker run -p 2000:2000 auth
```

6. TESTING

6.1. Automating Test Dispatcher

We have created a set of tests to audit the behaviour of the Dispatcher. These tests can be run by the developer/integrator, after the release of a new version to verify the previously implemented functions are still working and nothing is broken after the new commit; or by the user, to verify the Dispatcher is correctly installed, configured and connected with its neighbour elements.

The tests are written using Robot Framework [7], a generic open source automation framework which can be used for test automation and robotic process automation (RPA). Robot Framework is open and extensible and can be integrated with virtually any other tool to create powerful and flexible automation solutions. Being open source (Apache License 2.0) also means that Robot Framework is free to use without licensing costs.

The testing framework is executed in a separate container executing the file “runtest.sh” which also includes a web server that serves the testing reports after the execution of the set of tests is finalised [Figure 45]. This report is accessible via port *8200*.

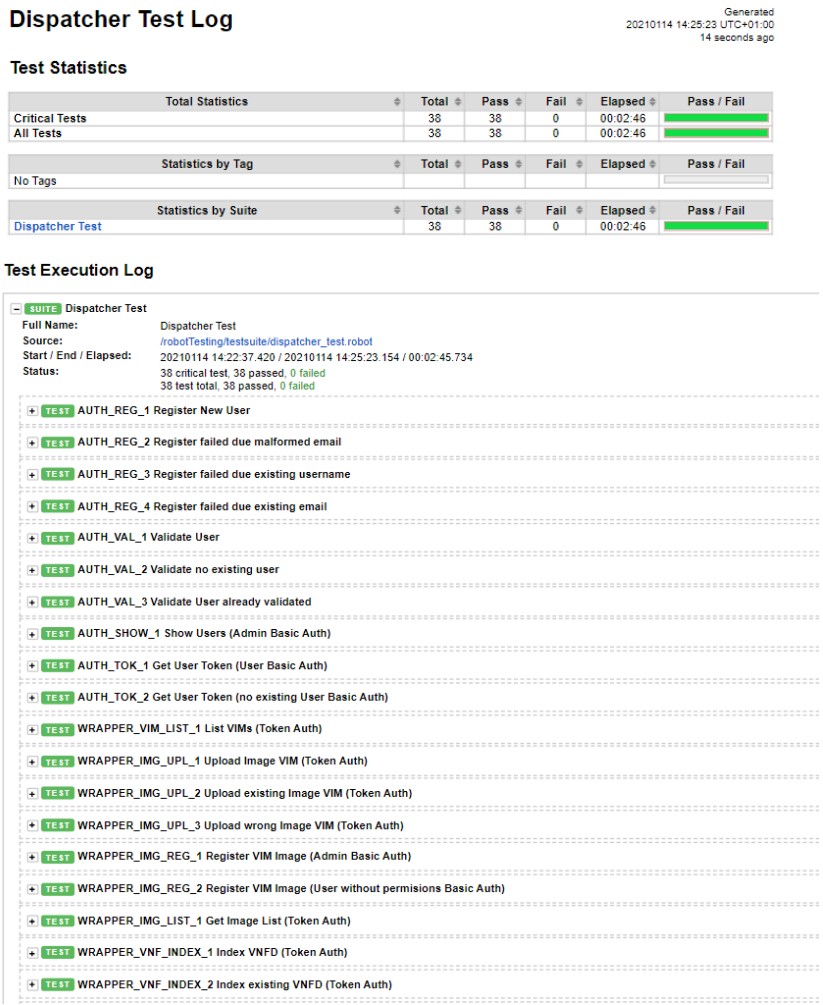


Figure 45 - Test report

Each endpoint is tested following the logical cycle of the application, just like the Experimenter would interact with the system. The Automated Dispatcher test also consider the error outputs of the system when the interaction with the system do not follow the system specifications.

The following table describe the template used to the test execution and the intermediate steps that pass the system internally.

Experimenter test

<table>
<tr><th>Identifier</th><th><Component>_<TestCase></th></tr>
<tr><td>Test Purpose & risks detected</td><td>Description of the test case and its importance due the risks that could exists if it is not verified and validate.</td></tr>
<tr><td>Configuration</td><td>Configuration of the component</td></tr>
<tr><td>Pre-test conditions</td><td>Preconditions in the test case (Populated databases, Application phases, authorized users…)</td></tr>
<tr><td>External components involved</td><td>
<table>
<tr><th>Component</th><th>Involved in</th><th>Mocked</th></tr>
<tr><td><Name></td><td><Usage in the Testcase></td><td>True | False</td></tr>
</table>
</td></tr>
<tr><td>Applicability</td><td>Component features in use</td></tr>
<tr><td>Test Cases</td><td>
<table>
<tr><th>ID</th><th>Description</th><th>Expected result</th></tr>
<tr><td>1</td><td><Testcase in success scenario></td><td><result></td></tr>
<tr><td>2</td><td><Testcase in failure scenario></td><td><result></td></tr>
<tr><td>3</td><td><Testcase in _____ scenario></td><td><result></td></tr>
</table>
</td></tr>
<tr><td>Test Sequences</td><td>
<table>
<tr><th>TC_ID</th><th>Sequence</th></tr>
<tr><td>1</td><td>< Sequence in a success scenario></td></tr>
<tr><td>2</td><td>< Sequence in failure scenario></td></tr>
<tr><td>3</td><td>< Sequence in _____ scenario></td></tr>
</table>
</td></tr>
</table>

- User Registration in the platform (AUTH_Reg)

Identifier	AUTH_REG
Test Purpose & risks detected	Register user in the platform to be validated by the Admin. The risks could be are three: • Registration of previously registered users

<table>
<tr><td></td><td>

- Registration with a wrong formed email

</td></tr>
<tr><td>Configuration</td><td>Component Auth is configured by default over the Dispatcher stack.</td></tr>
<tr><td>Pre-test conditions</td><td>The user’s database is empty, the only user able to perform actions in the platform is the Admin.</td></tr>
<tr><td>External components involved</td><td>
<table>
<tr><th>Component</th><th>Involved in</th><th>Mocked</th></tr>
<tr><td>Dispatcher</td><td>Reverse proxy in the REST Request</td><td>False</td></tr>
<tr><td>Gmail</td><td>Mail notification</td><td>False</td></tr>
</table>
</td></tr>
<tr><td>Applicability</td><td>Auth module provides authorization functions at Dispatcher level. This testcase validates the registration of users. Then the admin can validate them, and the user is able to use the platform.</td></tr>
<tr><td>Test Cases</td><td>
<table>
<tr><th>ID</th><th>Description</th><th>Expected result</th></tr>
<tr><td>1</td><td>Success Registration</td><td>User registered</td></tr>
<tr><td>2</td><td>Failed registration due to malformed email</td><td>Email malformed</td></tr>
<tr><td>3</td><td>Failed registration due to an existing username</td><td>Username already exists</td></tr>
<tr><td>4</td><td>Failed registration due an existing email</td><td>Email already exists</td></tr>
</table>
</td></tr>
<tr><td>Test Sequences</td><td>
<table>
<tr><th>TC_ID</th><th>Sequence</th></tr>
<tr><td>1</td><td>Inputs: username, email, password
I. Dispatcher redirects the request to auth module
II. Auth validates the imputs
III. Auth check if the user already exists in the database
IV. User is registered in the system.
V. Email notification to the platform Admin to validate the user</td></tr>
<tr><td>2</td><td>Inputs: username, malformed email, password
I. Dispatcher redirects the request to auth module
II. Auth rejects the email validation</td></tr>
<tr><td>3</td><td>Inputs: existing username, email, password
I. Dispatcher redirects the request to auth module
II. Auth validates the inputs
III. Auth rejects the registration due username already exists</td></tr>
</table>
</td></tr>
</table>

	4	Inputs: username, existing email, password I. Dispatcher redirects the request to auth module II. Auth validates the inputs III. Auth rejects the registration due email already exists

- Validate the account

Identifier	AUTH_VAL		
Test Purpose & risks detected	the Admin is able to validate users once a user is registered in the platform. The risks could be are three: • Try to validate users that does not exists • Validate a user previously validated		
Configuration	Component Auth is configured by default over the Dispatcher stack.		
Pre-test conditions	The user's database has one user pending to be validated, the only user able to perform actions in the platform is the Admin before the validation.		
External components involved	Component	Involved in	Mocked
	Dispatcher	Reverse proxy in the REST Request	False
	Gmail	Mail notification	False
Applicability	This testcase validates the validation of users. Once the admin validates the user, the user is able to use the platform.		
Test Cases	ID	Description	Expected result
	1	Success validation	User validated
	2	Failed validation due an inexistent user	User does not exist
	3	Validation over a validated user	No changes
Test Sequences	TC_ID	Sequence	
	1	Inputs: username (existing user and not validated yet) I. Dispatcher redirects the request to auth module II. Auth validates the inputs III. Auth check if the user already exists IV. User is validated in the system. V. Email notification to the user	
	2	Inputs: username (existing user) I. Dispatcher redirects the request to auth module II. Auth validates the inputs	

		III. Auth rejects because the user does not exist
	3	Inputs: username (existing user and validated) I. Dispatcher redirects the request to auth module II. Auth validates the inputs III. Auth check if the user already exists IV. User is already validated in the system

- Show users

Identifier	AUTH_SHOW		
Test Purpose & risks detected	The Admin is able to show the users registered in the platform.		
Configuration	Component Auth is configured by default over the Dispatcher stack.		
Pre-test conditions	The user's database has one user validated.		
External components involved	Component	Involved in	Mocked
	Dispatcher	Reverse proxy in the REST Request	False
Applicability	This testcase validates the show users' function.		
Test Cases	ID	Description	Expected result
	1	Success users retrieve	Users
Test Sequences	TC_ID	Sequence	
	1	Inputs: Basic admin Auth I. Dispatcher redirects the request to auth module II. Auth retrieve the users in the platform	

- Login platform

Identifier	AUTH_TOK
Test Purpose & risks detected	Authorization by Token is the best option to access in the Dispatcher service because a token is associated to an account and the token has a short life. However, the risks are the following: • Get an authorization token from a not validated user • Get an authorization token from a not existing user
Configuration	Component Auth is configured by default over the Dispatcher stack.

<table>
<tr><td>Pre-test conditions</td><td>The user's database has one user pending to be validated and other already validated.</td></tr>
<tr><td>External components involved</td><td>
<table>
<tr><th>Component</th><th>Involved in</th><th>Mocked</th></tr>
<tr><td>Dispatcher</td><td>Reverse proxy in the REST Request</td><td>False</td></tr>
</table>
</td></tr>
<tr><td>Applicability</td><td>Auth module provides authorization functions at Dispatcher level. This testcase validates the retrieve of token access to use the Dispatcher in a more secure way from Basic Auth.</td></tr>
<tr><td>Test Cases</td><td>
<table>
<tr><th>ID</th><th>Description</th><th>Expected result</th></tr>
<tr><td>1</td><td>Success token retrieve</td><td>Token</td></tr>
<tr><td>2</td><td>Failed token retrieve from a not existing user</td><td>Not existing user</td></tr>
</table>
</td></tr>
<tr><td>Test Sequences</td><td>
<table>
<tr><th>TC_ID</th><th>Sequence</th></tr>
<tr><td>1</td><td>Inputs: Basic Auth (Username, Password)
I. Dispatcher redirects the request to auth module
II. Auth check if the user already exists and its validated
III. Token is built and retrieved</td></tr>
<tr><td>2</td><td>Inputs: Basic Auth (Not existing Username, Password)
I. Dispatcher redirects the request to auth module
II. Auth check if the user already exists and its validated
III. Rejection of token built</td></tr>
</table>
</td></tr>
</table>

- List VIMs

<table>
<tr><th>Identifier</th><th>WRAPPER_VIM_LIST</th></tr>
<tr><td>Test Purpose & risks detected</td><td>The VIM is one of the main resources in the Mano Wrapper to deploy NS. A NS instantiation needs at least one to deploy the services.</td></tr>
<tr><td>Configuration</td><td>Mano component should be configured with at least one VIM.</td></tr>
<tr><td>Pre-test conditions</td><td></td></tr>
<tr><td>External components involved</td><td>
<table>
<tr><th>Component</th><th>Involved in</th><th>Mocked</th></tr>
<tr><td>Dispatcher</td><td>Reverse proxy in the REST Request</td><td>False</td></tr>
<tr><td>Auth</td><td>Admin authenticator</td><td>False</td></tr>
</table>
</td></tr>
<tr><td>Applicability</td><td>Mano module retrieves the VIM PoP available.</td></tr>
</table>

Test Cases	ID	Description	Expected result
	1	Success VIM resources retrieve	VIM resources
Test Sequences	TC_ID	Sequence	
	1	Inputs: Auth token I. Dispatcher redirects the request to auth module II. Auth check if the user already exists and its validated III. Dispatcher redirects the request to mano module IV. Mano retrieves the VIM PoPs and manager	

- Upload image VIM

Identifier	WRAPPER_IMG_UPL		
Test Purpose & risks detected	Mano is able to upload VIM images. The images could have several formats allowed by the VIM. Taking this into account we consider several risks: • Upload the same image twice • Upload an image with not allowed format		
Configuration	Mano component should be configured with at least one VIM.		
Pre-test conditions	The VIM has not images uploaded.		
External components involved	Component	Involved in	Mocked
	Dispatcher	Reverse proxy in the REST Request	False
	Auth	User authenticator	False
	VIM	Image Upload	False
Applicability	Mano requires VIM images to index VNFs in the NS repository. The images could be used for 0 to N VNFs.		
Test Cases	ID	Description	Expected result
	1	Success image upload	Image uploaded
	2	Failed to upload an existing image	Image not uploaded
	3	Failed Image upload due an unsupported image extension	Unsupported image
Test Sequences	TC_ID	Sequence	
	1	Inputs: Auth token + image file + selected VIM I. Dispatcher redirects the request to auth module	

		II. Auth check if the user already exists and its validated III. Dispatcher redirects the request to mano module IV. Mano validate the image file V. Mano save the image details in the DB VI. Mano upload the image in the VIM
	2	Inputs: Auth token + image file + selected VIM I. Dispatcher redirects the request to auth module II. Auth check if the user already exists and its validated III. Dispatcher redirects the request to mano module IV. Mano validate the image file V. Mano rejects the image file due the image already exists
	3	Inputs: Auth token + image file + selected VIM I. Dispatcher redirects the request to auth module II. Auth check if the user already exists and its validated III. Dispatcher redirects the request to mano module IV. Mano rejects the image file due the file extension is not supported

- Register VIM image

Identifier	WRAPPER_IMG_REG		
Test Purpose & risks detected	The VIM could have images preloaded or the process for the administrator could be easier to upload images directly to the VIM. In this case, the Mano is able to just register images to satisfy the VNF dependencies without uploading an image by the Mano wrapper. The risks detected are the following: • Users can not register images in the VIM		
Configuration	Mano component should be configured with at least one VIM.		
Pre-test conditions	Image to be registered is uploaded in the VIM		
External components involved	Component	Involved in	Mocked
	Dispatcher	Reverse proxy in the REST Request	False
	Auth	Admin authenticator	False
Applicability	Admin can register images to satisfy VNF dependencies.		
Test Cases	ID	Description	Expected result
	1	Success image register	Success registration

	2	Failed image register (try to be registered by one user)	Registration rejection

Test Sequences

TC_ID	Sequence
1	Inputs: Admin Basic Auth+ name image +vim I. Dispatcher redirects the request to auth module II. Auth check if admin user III. Dispatcher redirects the request to mano module IV. Mano register the image in the DB
2	Inputs: User Basic Auth I. Dispatcher redirects the request to auth module II. Auth check if the user already exists and its validated III. Dispatcher redirects the request to mano module IV. Mano rejects the registration due the user is not the Admin

- List images

Identifier	WRAPPER_IMG_LIST
Test Purpose & risks detected	The images located in the VIM are the main dependency for VNFs packages. Having a clear understanding of the current images in the VIM is an important knowledge for the experimenter.
Configuration	Mano component should be configured with at least one VIM.
Pre-test conditions	One or more images uploaded or registered in the VIM.

External components involved

Component	Involved in	Mocked
Dispatcher	Reverse proxy in the REST Request	False
Auth	Admin authenticator	False

Applicability	Mano module retrieves the VIM images available.

Test Cases

ID	Description	Expected result
1	Success VIM images retrieve	VIM images

Test Sequences

TC_ID	Sequence
1	Inputs: Auth token V. Dispatcher redirects the request to auth module VI. Auth check if the user already exists and its validated VII. Dispatcher redirects the request to mano module

	VIII. Mano retrieves the VIM images in all PoPs

- Index VNF

Identifier	WRAPPER_VNF_INDEX
Test Purpose & risks detected	Mano is able to create a repository of NS checking all the dependencies required to be instantiated. The first step to index in the repository is the VNFs. The VNFs must be validated and checked. Taking this into account we consider several risks: • Upload the same VNF twice • Upload a VNF with incorrect descriptor • Upload a VNF with an image dependency that does not exist
Configuration	Mano component should be configured with at least one VIM.
Pre-test conditions	The VNF repository is empty. The VIM has at least one image.
External components involved	Component: Dispatcher — Involved in: Reverse proxy in the REST Request — Mocked: False Component: Auth — Involved in: User authenticator — Mocked: False
Applicability	Mano can create a NS repository indexing VNFs and NSs artefacts. The first step is the Indexing of VNFs.
Test Cases	ID 1 — Description: Success VNF Indexing — Expected result: VNF Indexed ID 2 — Description: Failed VNF Index (VNF already exists) — Expected result: VNF not uploaded ID 3 — Description: Failed VNF Index (Wrong descriptor) — Expected result: VNFD wrong formed ID 4 — Description: Failed VNF Index (Image dependency does not exist) — Expected result: Image does not exist
Test Sequences	TC_ID 1 — Sequence: Inputs: Auth token + VNF I. Dispatcher redirects the request to auth module II. Auth check if the user already exists and its validated III. Dispatcher redirects the request to mano module IV. Mano validate the VNF and check its dependencies V. Mano index the VNF TC_ID 2 — Sequence: Inputs: Auth token + VNF (already indexed) I. Dispatcher redirects the request to auth module II. Auth check if the user already exists and its validated

		III. Dispatcher redirects the request to mano module IV. Mano rejects the indexing due the VNF already exists
	3	Inputs: Auth token + VNF (already wrong formed) I. Dispatcher redirects the request to auth module II. Auth check if the user already exists and its validated III. Dispatcher redirects the request to mano module IV. Mano rejects the indexing due the VNF is wrong formed
	4	Inputs: Auth token + VNF (missing image dependency) I. Dispatcher redirects the request to auth module II. Auth check if the user already exists and its validated III. Dispatcher redirects the request to mano module IV. Mano rejects the indexing due the VNF requires an image not uploaded or registered yet

- List VNF

Identifier	WRAPPER_VNF_LIST		
Test Purpose & risks detected	Mano is able to list the VNFs indexed in the repository. This action is performed by a regular user to know the current VNFs in the system.		
Configuration	Mano component should be configured with at least one VIM.		
Pre-test conditions	The VNF repository has at least one VNF indexed.		
External components involved	Component	Involved in	Mocked
	Dispatcher	Reverse proxy in the REST Request	False
	Auth	User authenticator	False
Applicability	Mano can list the VNF packages in the repository.		
Test Cases	ID	Description	Expected result
	1	Success VNF Listing	VNF Packages data
Test Sequences	TC_ID	Sequence	
	1	Inputs: Auth token I. Dispatcher redirects the request to auth module II. Auth check if the user already exists and its validated III. Dispatcher redirects the request to mano module IV. Mano list the VNFs available for the user (public packages and privates of the user) in the repository	

- Index NS

<table>
<tr><th>Identifier</th><th>WRAPPER_NS_INDEX</th></tr>
<tr><td>Test Purpose & risks detected</td><td>Mano is able to create a repository of NS checking all the dependencies required to be instantiated. The second step to index in the repository, after the VNFs, is the NSs. The NSs must be validated and checked. Taking this into account we consider several risks:
Upload the same NS twice
Upload a NS with incorrect descriptor
Upload a NS with a VNF dependency that does not exist</td></tr>
<tr><td>Configuration</td><td>Mano component should be configured with at least one VIM.</td></tr>
<tr><td>Pre-test conditions</td><td>The VNF repository has at least one VNF.</td></tr>
<tr><td>External components involved</td><td><table><tr><th>Component</th><th>Involved in</th><th>Mocked</th></tr><tr><td>Dispatcher</td><td>Reverse proxy in the REST Request</td><td>False</td></tr><tr><td>Auth</td><td>User authenticator</td><td>False</td></tr></table></td></tr>
<tr><td>Applicability</td><td>Mano can create a NS repository indexing VNFs and NSs artefacts. The second step is the Indexing of NSs.</td></tr>
<tr><td>Test Cases</td><td><table><tr><th>ID</th><th>Description</th><th>Expected result</th></tr><tr><td>1</td><td>Success NS Indexing</td><td>NS Indexed</td></tr><tr><td>2</td><td>Failed NS Index (NS already exists)</td><td>NS not uploaded</td></tr><tr><td>3</td><td>Failed NS Index (Wrong descriptor)</td><td>NSD wrong formed</td></tr><tr><td>4</td><td>Failed NS Index (VNF dependency does not exist)</td><td>VNF does not exist</td></tr></table></td></tr>
<tr><td>Test Sequences</td><td><table><tr><th>TC_ID</th><th>Sequence</th></tr><tr><td>1</td><td>Inputs: Auth token + NS
VI. Dispatcher redirects the request to auth module
VII. Auth check if the user already exists and its validated
VIII. Dispatcher redirects the request to mano module
IX. Mano validate the NS and check its dependencies
X. Mano index the NS</td></tr><tr><td>2</td><td>Inputs: Auth token + NS (already indexed)
V. Dispatcher redirects the request to auth module
VI. Auth check if the user already exists and its validated
VII. Dispatcher redirects the request to mano module
VIII. Mano rejects the indexing due the NS already exists</td></tr><tr><td>3</td><td>Inputs: Auth token + NS (already wrong formed)
V. Dispatcher redirects the request to auth module</td></tr></table></td></tr>
</table>

		VI. Auth check if the user already exists and its validated VII. Dispatcher redirects the request to mano module VIII. Mano rejects the indexing due the NS is wrong formed
	4	Inputs: Auth token + VNF (missing image dependency) V. Dispatcher redirects the request to auth module VI. Auth check if the user already exists and its validated VII. Dispatcher redirects the request to mano module VIII. Mano rejects the indexing due the NS requires an VNF not indexed yet

- List NS

Identifier	WRAPPER_NS_LIST		
Test Purpose & risks detected	Mano is able to list the NSDs indexed in the repository. This action is performed by a regular user to know the current NSs in the system.		
Configuration	Mano component should be configured with at least one VIM.		
Pre-test conditions	The VNF repository has at least one NS indexed.		
External components involved	Component	Involved in	Mocked
	Dispatcher	Reverse proxy in the REST Request	False
	Auth	User authenticator	False
Applicability	Mano can list the NS packages in the repository.		
Test Cases	ID	Description	Expected result
	1	Success NS Listing	NS Packages data
Test Sequences	TC_ID	Sequence	
	1	Inputs: Auth token V. Dispatcher redirects the request to auth module VI. Auth check if the user already exists and its validated VII. Dispatcher redirects the request to mano module VIII. Mano list the NSs available in the repository (public packages and privates of the user)	

- Onboard NS

Identifier	WRAPPER_NS_ONBOARD
Test Purpose & risks detected	Mano is able onboard in the NFVO the required NS and VNF packages. Taking this into account we consider several risks: • Onboard the NS or VNF twice

<table>
<tr><td></td><td>

- Onboard a no existing NS

</td></tr>
<tr><td>Configuration</td><td>Mano component should be configured with at least one NFVO.</td></tr>
<tr><td>Pre-test conditions</td><td>The NS repository has at least one NS.</td></tr>
<tr><td>External components involved</td><td>
<table>
<tr><th>Component</th><th>Involved in</th><th>Mocked</th></tr>
<tr><td>Dispatcher</td><td>Reverse proxy in the REST Request</td><td>False</td></tr>
<tr><td>Auth</td><td>User authenticator</td><td>False</td></tr>
<tr><td>OSM (NFVO)</td><td>NFVO to onboard the NS</td><td>False</td></tr>
</table>
</td></tr>
<tr><td>Applicability</td><td>Mano onboard the required NS artefacts in the NFVO to allow the NFVO to deploy those network services.</td></tr>
<tr><td>Test Cases</td><td>
<table>
<tr><th>ID</th><th>Description</th><th>Expected result</th></tr>
<tr><td>1</td><td>Success NS onboarding</td><td>NS onboarded</td></tr>
<tr><td>2</td><td>Failed NS onboarding (NS already onboarded)</td><td>NS not onboarded</td></tr>
<tr><td>3</td><td>Failed NS onboarding (NS do not exist)</td><td>Not found NS</td></tr>
</table>
</td></tr>
<tr><td>Test Sequences</td><td>
<table>
<tr><th>TC_ID</th><th>Sequence</th></tr>
<tr><td>1</td><td>Inputs: Auth token + NS (Already indexed)
I. Dispatcher redirects the request to auth module
II. Auth check if the user already exists and its validated
III. Dispatcher redirects the request to mano module
IV. Mano checks if the NS is already onboarded
V. Mano finds the NS and check its dependencies
VI. Mano onboard into NFVO the VNFs and NS and return the Ns_id (onboarded)</td></tr>
<tr><td>2</td><td>Inputs: Auth token + NS
I. Dispatcher redirects the request to auth module
II. Auth check if the user already exists and its validated
III. Dispatcher redirects the request to mano module
IV. Mano rejects the onboarding due NS already exists</td></tr>
<tr><td>3</td><td>Inputs: Auth token + NS (Not existing)
I. Dispatcher redirects the request to auth module
II. Auth check if the user already exists and its validated
III. Dispatcher redirects the request to mano module
IV. Mano rejects the onboarding due NS does not exist</td></tr>
</table>
</td></tr>
</table>

- Delete NS

Identifier	WRAPPER _NS_DELETE		
Test Purpose & risks detected	Mano is able to delete the NS packages from the repository and OSM. Taking this into account we consider several risks: • Delete a non-existing NS		
Configuration	Mano component should be configured with at least one NFVO.		
Pre-test conditions	The NS repository has at least one NS and the NFVO could have the NS onboarded.		
External components involved	Component	Involved in	Mocked
	Dispatcher	Reverse proxy in the REST Request	False
	Auth	User authenticator	False
	OSM (NFVO)	NFVO to onboard the NS	False
Applicability	Mano NS deletion allows remove the NS package from the repository and from the NFVO if it is onboarded.		
Test Cases	ID	Description	Expected result
	1	Success NS deletion	NS deleted
	2	Failed NS deletion (NS does not exist)	NS not deleted
Test Sequences	TC_ID	Sequence	
	1	Inputs: Auth token + NS id (Exists and onboarded) I. Dispatcher redirects the request to auth module II. Auth check if the user already exists and its validated III. Dispatcher redirects the request to mano module IV. Mano checks if the NS exists, if it has been indexed by the user and if it is already onboarded V. Mano remove the onboarded and the indexed NS	
	2	Inputs: Auth token + NS id (Not exists) I. Dispatcher redirects the request to auth module II. Auth check if the user already exists and its validated III. Dispatcher redirects the request to mano module IV. Mano rejects the deletion because NS does not exist	

- ED validation

Identifier	DISTR _ED_VALIDATION

<table>
<tr><td>Test Purpose & risks detected</td><td>Distributor component is able to verify and validate the Experiments Descriptors. Taking this into account we consider several risks:
Validation of an ED with missing parameters
Validation of an ED with wrong values types (lists instead strings)
Validation of an ED with wrong experiment distributed</td></tr>
<tr><td>Configuration</td><td>ELCM endpoint must be configured in the dispatcher.conf file.</td></tr>
<tr><td>Pre-test conditions</td><td>The ELCM is up and running.</td></tr>
<tr><td>External components involved</td><td>
<table>
<tr><th>Component</th><th>Involved in</th><th>Mocked</th></tr>
<tr><td>Dispatcher</td><td>Reverse proxy in the REST Request</td><td>False</td></tr>
<tr><td>Auth</td><td>User authenticator</td><td>False</td></tr>
<tr><td>ELCM</td><td>Check of the ELCM resources</td><td>True</td></tr>
</table>
</td></tr>
<tr><td>Applicability</td><td>Distributor is able to validate the EDs (Experiment Descriptor) and test if the dependencies are available.</td></tr>
<tr><td>Test Cases</td><td>
<table>
<tr><th>ID</th><th>Description</th><th>Expected result</th></tr>
<tr><td>1</td><td>Success validation</td><td>ED validated</td></tr>
<tr><td>2</td><td>Failed validation due missing parameters in the ED</td><td>ED not validated</td></tr>
<tr><td>3</td><td>Failed validation due wrong types in the ED</td><td>ED not validated</td></tr>
<tr><td>4</td><td>Failed validation due wrong distributed ED</td><td>ED not validated</td></tr>
</table>
</td></tr>
<tr><td>Test Sequences</td><td>
<table>
<tr><th>TC_ID</th><th>Sequence</th></tr>
<tr><td>1</td><td>Inputs: Auth token + ED
I. Dispatcher redirects the request to Auth module
II. Auth check if the user already exists and its validated
III. Dispatcher redirects the request to Distributor module
IV. Distributor validates the ED, the ELCM and the MANO NS and VIM dependencies.
V. Distributor retrieve an ok with the validation.</td></tr>
<tr><td>2</td><td>Inputs: Auth token + ED
I. Dispatcher redirects the request to Auth module
II. Auth check if the user already exists and its validated
III. Dispatcher redirects the request to Distributor module</td></tr>
</table>
</td></tr>
</table>

		IV. Distributor detects that the ED is not properly composed due missing parameters. V. Distributor retrieve a not ok with the validation.
	3	Inputs: Auth token + ED I. Dispatcher redirects the request to Auth module II. Auth check if the user already exists and its validated III. Dispatcher redirects the request to Distributor module IV. Distributor detects that the ED is not properly composed due bad types of any parameter. V. Distributor retrieve a not ok with the validation.
	4	Inputs: Auth token + ED I. Dispatcher redirects the request to Auth module II. Auth check if the user already exists and its validated III. Dispatcher redirects the request to Distributor module IV. Distributor detects that the ED is not properly composed due missing distributed parameter. V. Distributor retrieve a not ok with the validation.

7. PORTAL

The 5GENESIS Portal leverages the use of the Open APIs for providing a more user-friendly web interface for experimenters to interact with the 5GENESIS Facilities. Aside from the capabilities provided during Release A of this component, which have been refined and improved according to the feedback received during the previous experimentation phase, the following features has been included as part of the Release B development cycle:

- Support for defining experiments with a customized set of parameters.
- Support for the definition of experiments based on the use of MONROE nodes.
- Implementation of the East/West interface for communicating two Portal instances during the definition of distributed experiments.
- Integration with the Dispatcher component, which provides the required privacy and security management.
- A complete overhaul of the network service's onboarding process, integrated with the Dispatcher's catalogue, as well as the network slicing configuration for experiments.

The 5Genesis Portal is available as open-source software since May 2020, being hosted as a GitHub repository [8].

7.1. Experiment definition

Users registered in the Portal can create experiments, which will be linked to their account for easier management. Each experiment execution will also be registered in the Portal. Four kinds of experiments can be created through the Portal interface:

- Standard experiments, where the experimenter selects a number of standard test cases as provided by the facility, which are executed without modification.
- Custom experiments, which are very similar to standard experiments, but allow the configuration of certain parameters. A custom experiment definition screen can be seen on Figure 46.

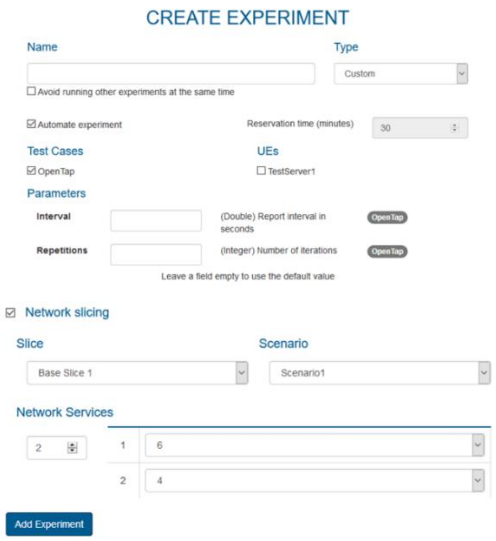


Figure 46: Custom experiment configuration parameters

- MONROE experiments, which are executed using a MONROE node and are configured by providing the name of the Application (MONROE container) to use, and the set of parameters that will be sent to the node Figure 47.

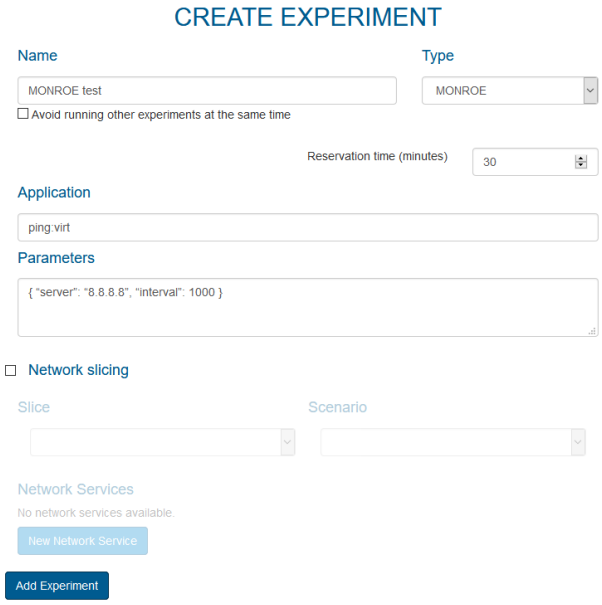


Figure 47: MONROE experiment definition

- Distributed experiments, where the execution is coordinated between two 5Genesis platforms. In this case, experimenters first configure all the parameters on the first platform and select a remote platform from the available ones (Figure 48, left). On a second step, experimenters configure the parameters for the remote platform, which the Portal transparently retrieves using the East/West interface (Figure 48, right).

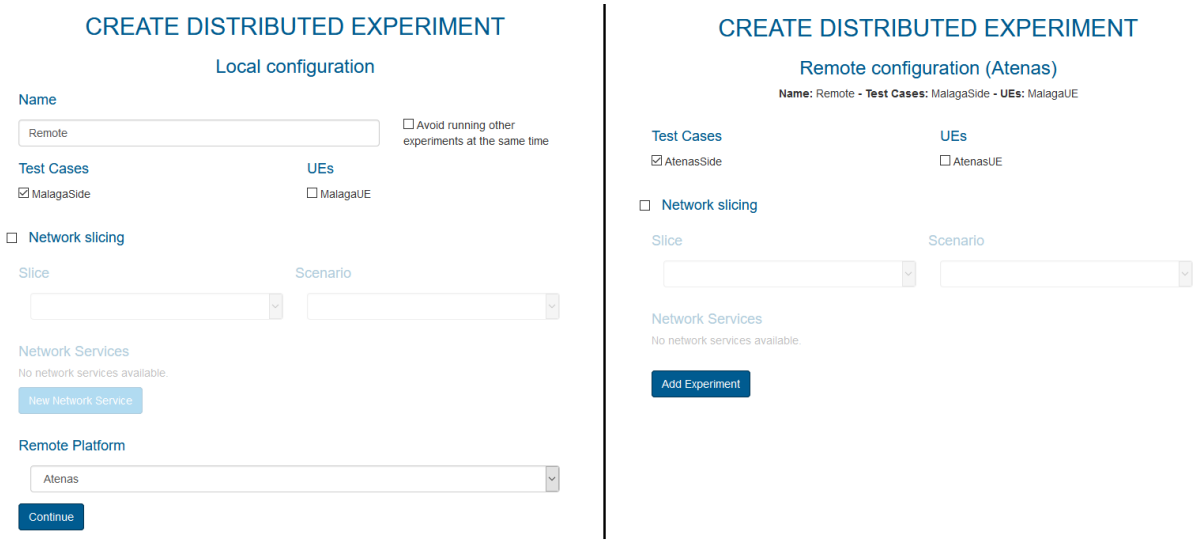


Figure 48: Distributed experiment definition

7.2. Experimenter dashboard

Once created, experiments are visible on the experimenter's dashboard (Figure 49), where the user can request the execution of the experiment or review the results and logs of the previous executions. In the case of distributed experiments, the user has access to the logs of both platforms (Figure 50).

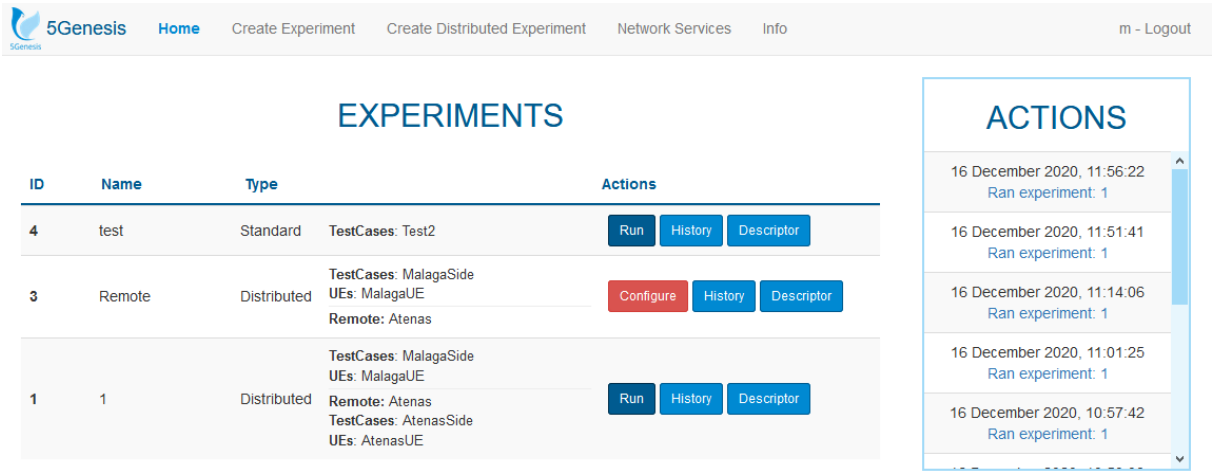


Figure 49: Experimenter dashboard

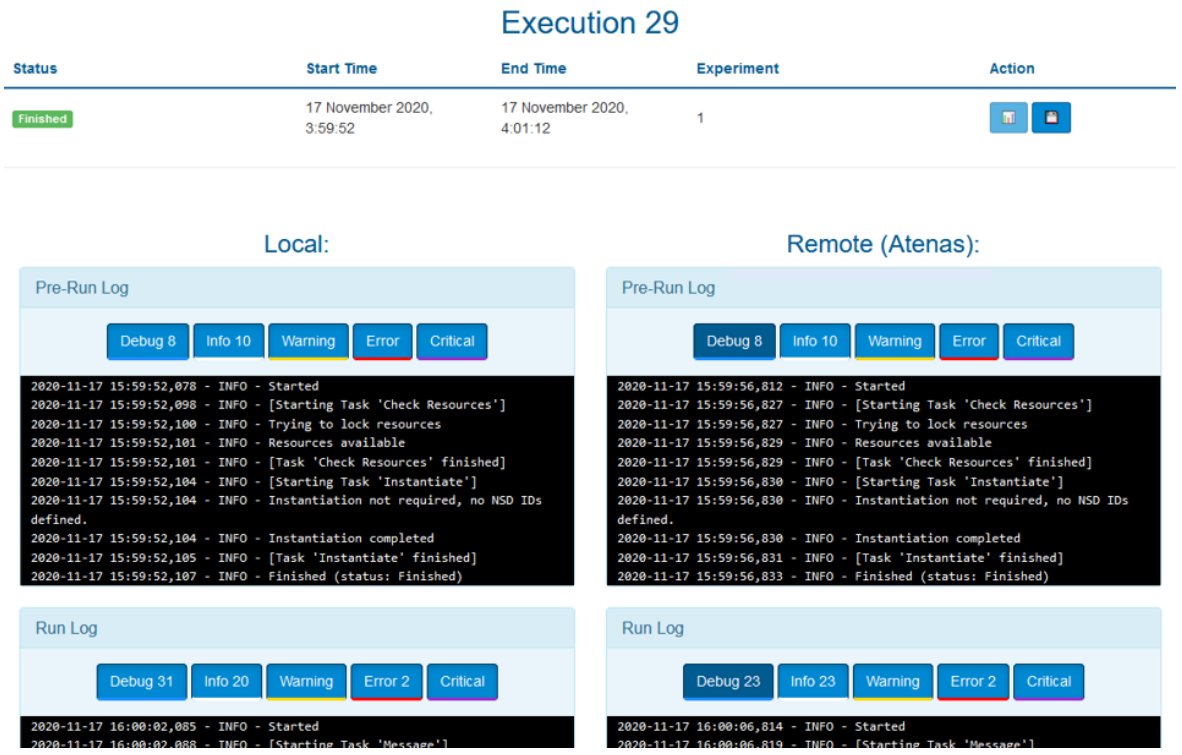


Figure 50: Execution logs (distributed experiment)

For each experiment execution, the Portal also provides a link to a customized Grafana dashboard (Figure 51) that displays the most important raw results obtained during the experiment execution as well as access to the Analytics dashboard, where the user can perform a more in-depth analysis of the results.

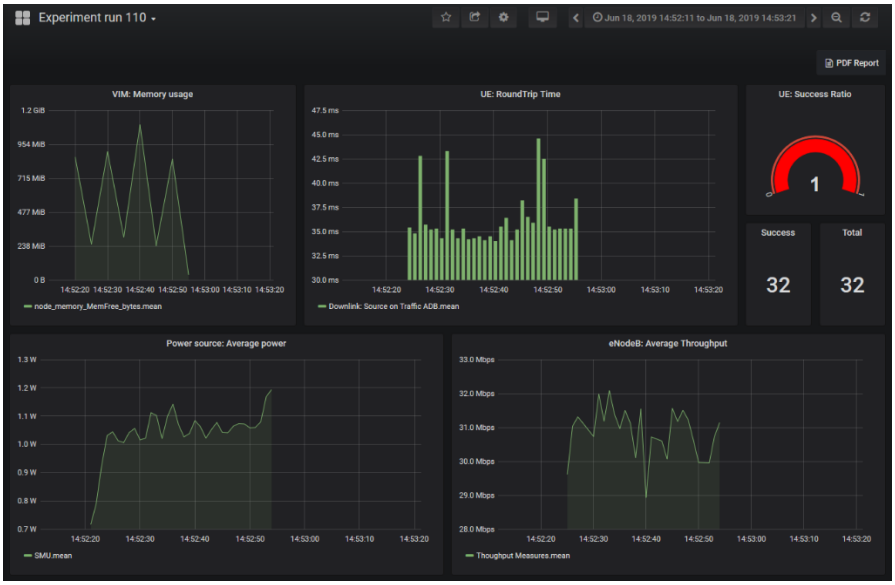


Figure 51: Grafana dashboard

7.3. Network services onboarding

The network service onboarding capabilities of the Portal has been re-built from the ground up since Release A, in order to take advantage of the additional capabilities provided by the network services repository available through the Dispatcher. The network service onboarding interface can be seen in Figure 52.

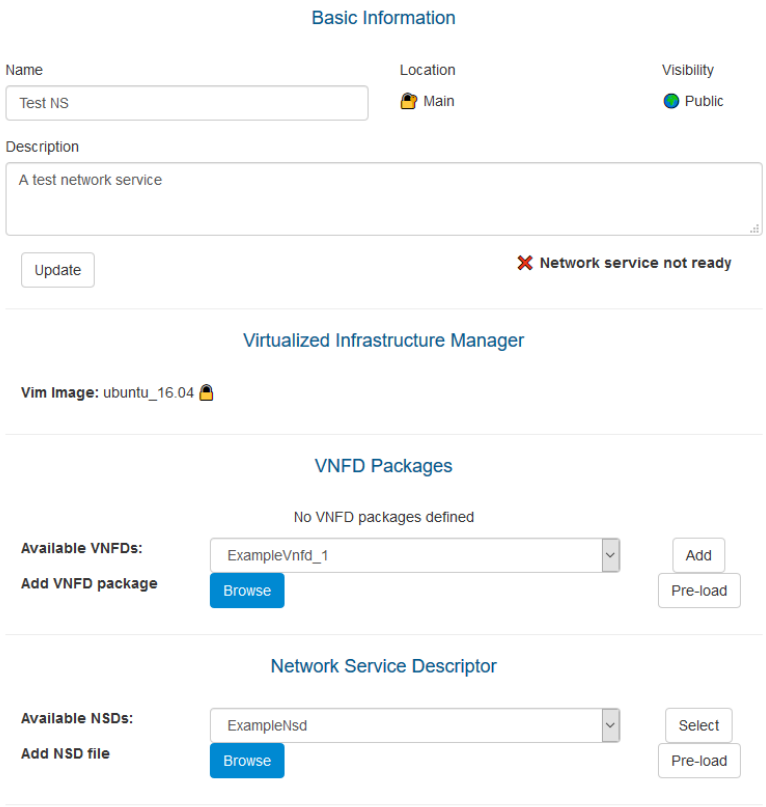


Figure 52: Network services onboarding

All network services available to the experimenter (which includes public network services as well as those defined by the experimenters themselves) are visible in the network services dashboard (Figure 53). From this interface, the user can continue editing previously created network services and easily see which ones can be used while defining a new experiment, since only those that are marked as ready are selectable in the experiment definition interface.

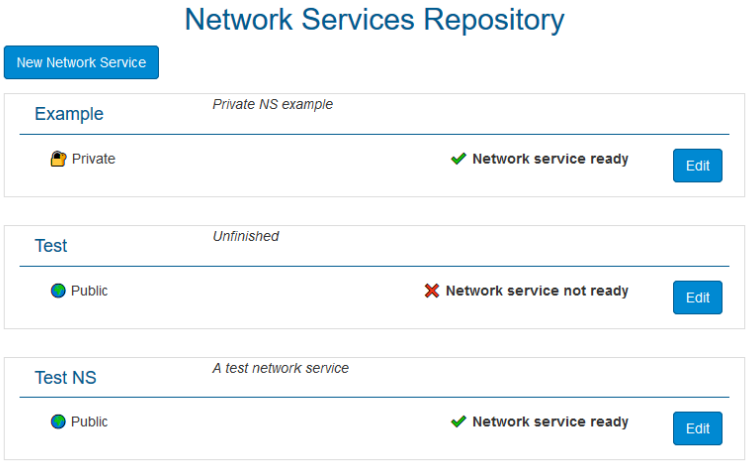


Figure 53: Available network services on the user account

7.4. Portal implementation

The 5Genesis Portal has been implemented using the Flask framework for Python, along with Bootstrap for the front-end rendering. Internally, the Portal makes use of the functionality provided by the Open APIs, as well as several additional data structures and logic in order to provide a more user-friendly interface for the experimenters.

7.4.1. User authentication

The Portal makes use of the user management capabilities of the Open API while providing an additional layer of security to user authentication. Portal accounts are automatically linked to a new Open API account upon registration, which effectively creates a two-step authentication process to every request performed through the Portal, while allowing us to implement additional functionality in the Portal independently or the features exposed by the Open API.

When a user registers through the Portal, an initial account is created in the Portal database, and then, the provided credentials are used for automatically generating a new account through the Open APIs. This initial account is unusable until the user activation has been completed by an administrator, given that every login attempt is performed in two steps:

- First, the credentials are compared with those in the Portal database. If the user exists and the credentials are correct the login process can continue.
- Then, the Portal checks the user authentication using the Open APIs. If the user is active, then they can access to the Portal.

This login step is performed by using basic authentication (i.e. sending the username and hashed password to the Open APIs), however, once the credentials are verified access tokens are retrieved through the Open APIs and used for every subsequent request.

The existence of the user's Portal account on top of the account on the Open APIs allows us to provide some additional features, like saving, for each user independently, their collection of defined experiments and network services, and provide easier access to the information of previous experiment executions.

Table 1 shows the endpoints of the Open APIs that are used for managing authentication:

Endpoint (Dispatcher)	Method	Notes
/auth/register	POST	Requests the creation of a new user.
/auth/get_token	GET	Retrieves a new access token for the user.

Table 1 Authentication endpoints used by the Portal

7.4.2. Experiment definition and execution

During an experiment definition, the Portal retrieves the platform configuration (available test cases, UEs, etc.) by communicating directly with the ELCM component on the same platform (Table 2). The Portal cannot make use of the related functionality in the Open APIs since this information is retrieved once during the start-up process and saved globally in the Portal instance. When generating the views for a specific user the Portal filters the information so that only the values available for that particular experimenter are visible.

Endpoint (ELCM)	Method	Notes
/facility/ues	GET	Retrieves a list of available UEs in the platform
/facility/testcases	GET	Retrieves a list of available test cases in the platform, along with additional information for each of them.
/facility/baseSliceDescriptors	GET	Retrieves a list of available base slice descriptors in the platform
/facility/scenarios	GET	Retrieves a list of available scenarios in the platform

Table 2 Facility information endpoints used by the Portal

In the case of a distributed experiment, makes use of the East/West interface for communicating with the remote platform. This communication is performed during the definition of the remote part of the experiment. Since the user does not necessarily have a user in the remote platform, and given that distributed experiments can only make use of public resources it is also not possible to retrieve this information using the Open APIs. The endpoints exposed by the East/West interface of the Portal can be seen in Table 3.

Endpoint (Remote Portal)	Method	Notes

/distributed/ues	GET	Retrieves a list of available UEs in the remote platform
/distributed /testcases	GET	Retrieves a list of available test cases in the remote platform, in this case only public test cases.
/distributed/baseSliceDescriptors	GET	Retrieves a list of available base slice descriptors in the remote platform
/distributed/scenarios	GET	Retrieves a list of available scenarios in the remote platform
/distributed/networkServices	GET	Retrieves a list of public network services in the remote platform

Table 3 East/West interface endpoints used by the Portal

Experiment execution, however, is performed by using the experimenter's credentials. In this case the Portal generates an Experiment Descriptor by using the values selected by the user and sends the request to the Open APIs, which perform the initial validation. This validation includes the user access to the selected test cases as well as the correct onboarding of the required network service's artifacts. Additionally, the execution logs are also retrieved through the Open APIs. The endpoints used can be seen in Table 4.

Endpoint (Dispatcher)	Method	Notes
/elcm/api/v0/run	POST	Requests the execution of an experiment, using an experiment descriptor
/elcm/execution/<id>/logs	GET	Retrieves the logs from an experiment execution

Table 4 Experiment execution endpoints used by the Portal

During the execution of the experiment the ELCM sends real time status information, such as the percentage of completion or the current stage of the experiment, to the Portal, which is displayed in the experimenter dashboard.

7.4.3. Network services onboarding

The definition of a complete network services is a complex task that involves the creation of several artifacts that need to be onboarded on the Management and Orchestration layer. The Portal provides an easy to use interface for performing this process in an organized manner, making use of the MANO wrapper provided by the Dispatcher component.
The definition of the network service can be completed step by step at different times, while the Portal keeps track of the current status. For the selection of each component the Portal provides a list of existing artifacts that are available, or allows the user to upload a new one.

Table 5 Shows the endpoints used during the onboarding process:

Endpoint (Dispatcher)	Method	Notes
/mano/vims	GET	Retrieves a list of available VIMs
/mano/image	GET	Retrieves the list of available images in each VIM

	POST	Onboards a new VIM image
/mano/vnfd	GET	Retrieves the list of available VNFDs
	POST	Onboards a new VNFD
/mano/nsd	GET	Retrieves the list of available NSDs
	POST	Onboards a new NSD

Table 5 Network service onboarding endpoints used by the Portal

8. CONCLUSIONS

The report presented the activities performed during the design and development of the Open APIS interface of the 5GENESIS facility [10]. In briefly presented the architecture and internal components that will manage the access of the Experimenter to the platform. Over the Release B implementation of the Open APIs, was extending the features and functionalities implemented during the first cycle refined, based on the feedback produced by the WP2 Facility Requirements and Specifications, improving the performance and security of the Open APIs. The authenticator module secures the access to the facility, enabling the communication with the Dispatcher component, that is the engine module that will redirect the Experimenter request to the underlying modules that communicates with the Enablers.

Moreover, the role of the Dispatcher in the Distribution of the experiments in a remote platform is also described together with the diagrams and flow for synchronizing the execution of the experiment and KPIs result gathering.

With the goal of offering an open and common method to interact with the Facility for experimentation, 5GENESIS exposes the Open API to two clients, by command line or by offering a Portal with more user-friendly Web UI. The current implementation of the Portal has been also presented in this document.

This document is delivered at month 33 of the project, as final report of the activities performed in Task 3.4 Open APIs, service level functions and interfaces for verticals, in the overall Openness Framework and Integral Components of the Facility in WP3.

REFERENCES

[1] D2.4 Final report on facility design and experimentation planning https://5genesis.eu/wp-content/uploads/2020/07/5GENESIS_D2.4_v1.0.pdf

[2] TM FORUM API references https://www.tmforum.org/open-apis

[3] *Customer Management API* https://projects.tmforum.org/wiki/display/API/Open+API+Table

[4] ETSI standards NFV-SOL 005 API - https://www.etsi.org/deliver/etsi_gs/NFV-SOL/001_099/005/02.07.01_60/gs_NFV-SOL005v020701p.pdf

[5] 5GENESIS Dispatcher Github repository: https://github.com/5genesis/Dispatcher/blob/master/dispatcher.conf

[6] Wrapper configuration file MANO module: https://github.com/5genesis/Dispatcher/blob/master/mano/README.md#config-file

[7] Robot Framework https://robotframework.org

[8] 5GENESIS Portal Repository https://github.com/5genesis/Portal

[9] Keycloak User Autentication https://ncarlier.gitbooks.io/oss-api-management/content/howto-kong_with_keycloak.html

[10] H. Koumaras et al., "5GENESIS: The Genesis of a flexible 5G Facility," 2018 IEEE 23rd International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD), Barcelona, Spain, 2018, pp. 1-6, doi: 10.1109/CAMAD.2018.8514956.