5G ExPerimentation Infrastructure hosting Cloud-nativE Network Applications for public proTection and disaster RElief

Innovation Action – ICT-41-2020 - 5G PPP – 5G Innovations for verticals with third party services

D3.2: 5G EPICENTRE Front-end components

Delivery date: June 2023

Dissemination level: Public

Project Title:	**5G-EPICENTRE - 5G ExPerimentation Infrastructure hosting Cloud-nativE Network Applications for public proTection and disaster RElief**
Duration:	**1 January 2021 – 31 December 2023**
Project URL	https://www.5gepicentre.eu/

This project has received funding from the European Union's Horizon 2020 Innovation Action programme under Grant Agreement No 101016521.

Document Information

Deliverable	D3.2: 5G EPICENTRE Front-end components
Work Package	WP3: Front-end components and 5G ExaaS APIs
Task(s)	T3.2: Novelty experiment insights visualization tools
Type	Report
Dissemination Level	Public
Due Date	M30, June 30, 2023
Submission Date	M31, July 31, 2023 M36, December 31, 2023 (revision)
Document Lead	Konstantinos C. Apostolakis (FORTH)
Contributors	Stefania Stamou (FORTH) Sozos Karageorgiou (EBOS) Christos Skoufis (EBOS) Daniel del Teso (NEM) Antonio Zanesco (YBQ) Manos Kamarianakis (ORAMA) Antonis Protopsaltis (ORAMA) Almudena Díaz Zayas (UMA) Ankur Gupta (HHI) Dani Alcaraz Mora (RZ) André Gomes (ONE) Rainer Wragge (OPTO) Carlos Martins Marques (ALB) Laurent Drouglazet (ADS)
Internal Review	IST ORAMA

Document history

Version	Date	Changes	Contributor(s)
V0.1	27/03/2023	Initial deliverable structure	Konstantinos Apostolakis (FORTH)
V0.2	08/05/2023	50% of the deliverable content	Konstantinos Apostolakis (FORTH) Stefania Stamou (FORTH)
V0.3	06/06/2023	Northbound Configuration dashboard Section (Section 2.2.2) initial draft.	Sozos Karageorgiou (EBOS) Christos Skoufis (EBOS)
V0.4	12/06/2023	Northbound Configuration dashboard Section (Section 2.2.2) updated final draft.	Sozos Karageorgiou (EBOS) Christos Skoufis (EBOS)
V0.5	16/06/2023	90% of the deliverable content	Konstantinos Apostolakis (FORTH) Stefania Stamou (FORTH)
V1.0	19/06/2023	Internal Review Version	Konstantinos Apostolakis (FORTH)
V1.1	24/06/2023	1st version with suggested revisions	Antonis Protopsaltis (ORAMA) Manos Kamarianakis (ORAMA)
V1.2	27/06/2023	2nd version with suggested revisions	Anna Maria Spagnolo (IST)
V1.5	05/07/2023	First revisions after internal review, including formatting and proof-reading	Konstantinos Apostolakis (FORTH)
V1.6	07/07/2023	Second revisions after internal review	Anna Maria Spagnolo (IST) Antonis Protopsaltis (ORAMA) Konstantinos Apostolakis (FORTH)
V2.0	10/07/2023	Final version for submission	Konstantinos Apostolakis (FORTH)
V2.1	26/12/2023	Implemented revisions after project periodic review and gathering additional inputs from partners – marked changes for 2nd round internal review.	Konstantinos Apostolakis (FORTH) Daniel del Teso (NEM) Antonio Zanesco (YBQ) Antonis Protopsaltis (ORAMA) Almudena Díaz Zayas (UMA) Ankur Gupta (HHI) Dani Alcaraz Mora (RZ) André Gomes (ONE) Rainer Wragge (OPTO) Carlos Martins Marques (ALB) Laurent Drouglazet (ADS)
V2.2	27/12/2023	Suggested revisions after internal review, final formatting and proof-reading (quality check)	Anna Maria Spagnolo (IST) Manos Kamarianakis (ORAMA) Konstantinos Apostolakis (FORTH)
V3.0	28/12/2023	Final revised version for submission	Konstantinos Apostolakis (FORTH)

Project Partners

Logo	Partner	Country	Short name
	AIRBUS DS SLC	France	**ADS**
	NOVA TELECOMMUNICATIONS SINGLE MEMBER S.A.	Greece	**NOVA**
	Altice Labs SA	Portugal	**ALB**
	Fraunhofer-Gesellschaft zur Förderung der angewandten Forschung e.V.	Germany	**HHI**
	Foundation for Research and Technology Hellas	Greece	**FORTH**
	Universidad de Malaga	Spain	**UMA**
	Centre Tecnològic de Telecomunicacions de Catalunya	Spain	**CTTC**
	Istella SpA	Italy	**IST**
	One Source Consultoria Informatica LDA	Portugal	**ONE**
	Iquadrat Informatica SL	Spain	**IQU**
	Nemergent Solutions S.L.	Spain	**NEM**
	EBOS Technologies Limited	Cyprus	**EBOS**
	RedZinc Services Limited	Ireland	**RZ**
	OptoPrecision GmbH	Germany	**OPTO**
	Youbiquo SRL	Italy	**YBQ**
	ORamaVR SA	Switzerland	**ORAMA**
	Hewlett-Packard Italiana Srl	Italy	**HPE**

List of abbreviations

Abbreviation	Definition
3GPP	3rd Generation Partnership Project
5G PPP	5G Public Private Partnership
5QI	5G QoS Indicator
AdE	Adaptation Engine
AF	Application Function
API	Application Programming Interface
AuthM	User Authentication and Management
CI/CD	Continuous Integration/Continuous Delivery
CNF	Containerised Network Function
CORS	Cross-Origin Resource Sharing
CSV	Comma Separated Values
DL	Downlink
DOM	Document Object Model
eMBB	Enhanced Mobile Broadband
EPI	Experiment Planning Interface
ExCom	Experiment Composer
GA	Grant Agreement
GUI	Graphical User Interface
HSPF	Holistic Security and Privacy Framework
IoT	Internet of Things
ITools	Insights Tools

JSON	JavaScript Object Notation
JWT	JSON Web Token
KPI	Key Performance Indicator
mMTC	Massive Machine Type Communications
MQTT	Message Queuing Telemetry Transport
nappD	Network Applications Creation and Management Dashboard
NCD	Northbound Configuration Dashboard
NF	Network Function
NSBR	Network Service Browser
QoS	Quality of Service
RBAC	Role-Based Access Control
REST	Representational State Transfer
UAO	Upgradeable, Acceptable, Optimal
UC	Use Case
UI	User Interface
UL	Uplink
URLLC	Ultra-Reliable Low Latency Communications
VIS	Visualisation Solution
VM	Virtual Machine
VNF	Virtual Network Function

Executive summary

This deliverable constitutes the final report on the 5G-EPICENTRE Front-end Layer architectural components of the 5G-EPICENTRE platform. Thereby, it constitutes the conclusive report on Work Package 3 activities.

All systems elaborated in the present document constitute user-facing applications which aim at facilitating the end-user experience with the 5G-EPICENTRE platform in a user-friendly and accessible manner. The current document also serves as a means to report on additional functionalities that have been implemented since M14 (referring to the preceding deliverables D3.1 *"5G EPICENTRE Northbound API"* and D3.3 *"5G EPICENTRE User Interface"*); which are now deployed and offered through the 5G-EPICENTRE Portal and Northbound configuration dashboard components.

WP3 has placed emphasis in the development of a richly-featured functional user-facing online platform for onboarding, testing and eventually, experimenting with vertical and network application software components for public protection and disaster relief, towards effectively executing experiments in 5G settings. The present document describes how these features have been implemented and integrated: from uploading experiment artefacts (as Helm chart packages) and reserving (*i.e.*, "booking") testbed resources for experimentation, to real-time monitoring and visualizing outcomes of the scheduled experiment. Following an intent-based user interface design philosophy, the components of this web portal interface with the Experiment Coordinator (described in deliverable D2.5) to deliver together the necessary system capabilities with respect to user expectations

In addition to a conclusive report, D3.2 can also be viewed as a manual that can be used by those who wish to interact with the 5G-EPICENTRE platform (in particular, project target third party experimenters), with the aim to test and validate their PPDR based solutions over fully featured 5G testbed environments. For this reason, instructions are included in the form of visual walkthroughs on performing the following series of actions: i) experiment definition, ii) on-boarding of experiment artefacts, iii) calibration of platform Network Applications and traffic simulation conditions, and iv) experiment reporting & visualization. Thereby, the current document includes a section devoted to instructions on the steps that the end-user can follow when they access the Portal as an experimenter. These contents will be complemented with an instructional video tutorial, which will be produced as part of the project's outreach activities in WP6.

Table of Contents

List of Figures

List of Tables

1 Introduction

Within the overall vision of 5G-EPICENTRE in accelerating 5G innovations for Public Protection and Disaster Relief (PPDR) verticals (*i.e.*, innovators and communications technology vendors), it is important to equip target end users[1] with tools for effortless interaction with the underlying testbed environments. In such manner, the platform's front-end components abstract several system functions and technologies (*e.g.*, Kubernetes, Karmada, JFrog, iPerf, RabbitMQ, *etc*.) that are integrated into the 5G-EPICENTRE experimentation workflow and each require different sets of commands, parameterization and Application Programming Interfaces (APIs) to work. By doing so, the front-end component facilitate interaction with the underlying systems and testbeds through a visually rich web-based environment. Thereby, actions such as deploying experiments, delegating resources as well as executing and retrieving experiment results over the four heterogeneous testbeds federated under 5G-EPICENTRE are streamlined in just a few mouse clicks. In the present document, all Graphical User Interface (GUI) components of the 5G-EPICENTRE integrated experimentation platform are described, intending to provide experimenters with means of managing experiment lifecycles, requests, and experimental conditions. The final version of the 5G-EPICENTRE Portal is presented, which constitutes the main access point to the 5G-EPICENTRE platform. Through the Portal the end-user can easily define experiments, monitor their execution and visualise measurements. In addition, a specialized Northbound Configuration Dashboard (NCD) is presented for allowing experimenters to interact with the underlying network control plane in a more natural manner, and within predefined sets of values.

The current deliverable is a culmination of all Task efforts in Work Package (WP) 3, and naturally complements and updates on the contents of prior WP deliverables D3.3 *"5G EPICENTRE User Interface"* and D3.*1 "5G EPICENTRE Northbound API"*. It also takes into account the most recent project Grant Agreement (GA) amendments and 5G Public Private Partnership (5G-PPP) Software Network Working Group white paper on Network Applications [1], which both reflect the project's alignment to the common definition of Network Applications and their proposed delivery models. Further, the deliverable is informed by the latest work carried out in WP1, namely, D1.4 *"Experimentation requirements and architecture specification final version"*, which elaborates on the role of the platform front-end components within the whole 5G-EPICENTRE architectural stack; D1.2 *"5G-EPICENTRE experimentation scenarios final version"*, which defines the various 5G experimentation scenarios that are possible through the four federated infrastructures; and D1.6 *"Experiment evaluation strategy and experimentation plan"*, which defined commonalities and nomenclature between the different project Use Cases (UCs) with relation to defining experimentation Key Performance Indicators (KPIs). Security-wise, the front-end components described in this document adhere to the design regulation put forward in D1.5 *"Security-by-design toolkit"*, and integrate with the front-end solutions described in D2.7 *"Cloud-native security intermediate version"*, regarding the platform's Network Intrusion and Detection Network Application. With respect to platform component integration aspects, the portal conforms to the integration guidelines laid out in D4.4 *"5G-EPICENTRE experimentation facility preliminary version"*.

In turn, this deliverable provides input to WP2 deliverables D2.4 *"5G-EPICENTRE service deployment"*; D2.5 *"5G-EPICENTRE Experiment execution"*; and D2.6 *"5G-EPICENTRE Analytics Engine"*, regarding the interfacing with their respective modules and subsystems, as well as the final technical interpretation of the 5G-EPICENTRE architecture in D4.5 *"5G-EPICENTRE experimentation facility final version"*. Its contents will also be useful in the context of WP5, particularly in the context of provisioning external (to the project) PPDR vertical application experimenters with tutorial content and Portal usage guidelines.

[1] The final target end user audience for the 5G-EPICENTRE front-end components (derived from the D3.3 "Portal actors" Table 2, p.16) includes: a) (third-party) ***experimenters***, who will utilize the platform for gaining easy access to experiment definition and deployment functionalities; and b) ***testbed owners***, who utilize the platform as a means to manage and negotiate the deployment requests from experimenters.

The remainder of this deliverable is organised as follows: Section 2 elaborates on the mappings drawn between the described components and functionalities and the project elicited requirements and technical component specifications. Section 3 delivers a comprehensive technical report on the 5G-EPICENTRE Portal implementation. Section 4 describes the processes which the front-end modules support, alongside the manner in which information flows through the system in each case. Finally, Section 5 presents indicative usage examples, complete with lots of visual references.

1.1 Mapping of project's outputs

The purpose of this section is to map the undertaken 5G-EPICENTRE GA commitments regarding the descriptions, against the project's respective outputs and work performed in the scope of WP3.

Table 1: Adherence to 5G-EPICENTRE's GA Tasks Descriptions

5G-EPICENTRE Task	Respective Document Chapters	Justification
Task 3.1: Northbound API information models and implementations *"[…] partners involved in this Task will determine and identify any existing API definitions (data models) and implementations that can satisfy the requirements of the selected experimentation scenarios (T1.1), proposing where applicable, any necessary extensions".*	2.2.2 – Northbound configuration dashboard	This Section provides insights regarding the functionality of the NCD and its underlying Network application. It further describes how users interact with the NCD by submitting requests through its front-end GUI. It elaborates on how users can select network properties they need to configure, how the NCD triggers the proper Service API calls needed to configure the network properties, and eventually, the display of the status of the request and feedback given to the user.
Task 3.1: Northbound API information models and implementations *[…] define new data models for the remaining necessary Northbound APIs in accordance to the experimental applications' requirements; […] develop a complete implementation of each of the APIs defined by the project partners".*		
T3.2: Novelty experiment insights visualization tools *"The purpose of this Task is the development of appropriate experiment KPI visualization tools, as well as, the orchestration of the information flow toward the end-users."*	4.3 – Presenting data generated at the	This Section presents the technical details on how experiment KPI visualization is achieved in the Portal, including the means by which KPI information enters, and is processed by the Portal backend and frontend subsystems.

T3.2: Novelty experiment insights visualization tools *"Hence, user interfaces will be targeted that foster usability via user experience design methods and support tools, for the purposes of visualization, interaction and comprehension of big data, toward allowing the user to process information much faster and retain it for longer".*	4.3.2 – 5G-EPICENTRE frontend visualization components	In this Section, the various visualization components are elaborated, with brief description of each one, and their pertinence to 5G-EPICENTRE generated insight and objectives.
	5.1.4 – Experiment execution	This Section provides a step-by-step visual walkthrough on how the visualization environment can be accessed and utilized by experimenters.
T3.3: Experiment planning dashboard *"This Task will deliver a web-based, information-rich graphical user interface (GUI)-driven portal for […] requesting execution of experiments to the 5G-EPICENTRE infrastructure. Through this interface, high-level administrator access will be provided for both first party, as well as third-party experimenters, who will be able to define and set-up an experiment".*	4.2– Calibrating underlying infrastructure components	The Section presents technical details on how experiment processes are implemented, as well as security mechanisms to ward off against unauthorized access.
	5.1.2 – Scheduling an experiment	This Section provides a step-by-step visual walk-through on how third-party experimenters can define and request experiment execution.
	5.1.3 – Experiment deployment	The Section provides a visual walk-through on how testbed administrators can manage the deployment of requested experiments.
T3.3: Experiment planning dashboard *"In addition, the user interface will allow experimenters to browse existing open VNF/Network Applications packages from centralized repositories (T4.2) so as to select desired Network Applications to be deployed across the federated Kubernetes clusters federated under the platform".*	4.2.1 – Delegating the vertical application components	This Section presents technical details on how vertical application component delegation processes are implemented within the Portal.
	5.1.1 – Delegating vertical application components to the testbed operator	This Section provides a step-by-step visual walk-through on how experimenters can request delegation of their vertical application artefacts onto any one of the four 5G-EPICENTRE testbeds.
T3.3: Experiment planning dashboard *"The dashboard will further provide KPI and insights visualization leveraging on the plugins developed in the context of T3.2".*	4.3.2 – 5G-EPICENTRE frontend visualization components	In this Section, the various visualization components are elaborated, with brief description of each one, and their pertinence to 5G-EPICENTRE generated insight and objectives.

1.2 Adherence to reviewers' comments and recommendations on D3.3

In addition, efforts were spent to address comments made by the project monitors with respect to D3.3 toward the improvement of the present document. More specifically, Table 2 below, aims at explicitly clarifying how review report comments from the second project review have been addressed.

Table 2: Amendments made to address each reviewers' comments

Review comment(s) (as provided by the reviewers)	5G-EPICENTRE Adherence and Document Update (short reply and reference to the chapter that details the reply)
User experience must be assessed throughout all use of the technology, and adaptations should be made to address poor UI experience.	Section 5 provides usage examples for each of the four indicative platform use cases presented in D1.4. Its purpose is to demonstrate streamlined processes for calibrating the underlying infrastructures, as well as experiment deployment in a few comprehensive steps.
If understood correctly, the PPDR user will request an experiment to be run by defining the Network Applications but actually 5G-EPICENTRE partners must prepare the experiment before PPDR users can run it. The reasoning for this is partly clear but limits the impact of the project to the project execution unless a sustainable business model is planned after the project.	The Portal enables (3rd party) experimenters to request execution of their experiments in 4 simple steps, thereby simplifying interaction with the testbed operators. In addition, verticals are allowed to delegate their vertical application components to the project artefact repository, actively opting for solutions that can be publicised to others, or kept internally, for a private experimental run. Deliverable D6.10 provides insight into possible business models for remunerating such interactions.
It is understandable that users are not allowed to modify the network infrastructure, but it is not clear why they are not allowed to configure it within predefined sets of values.	Integration of the Portal with the project defined vertical-agnostic platform Network Applications opens up the possibility to modify, to an extent, the underlying infrastructure. In particular, experimenters are granted some extent of programmability over the network control plane, *e.g.*, to request higher priority and guaranteed QoS for particular flows (*i.e.*, via the Configurator, see Section 2.2.2), as well as control over security policies (via the Network Intrusion and Detection integration), KPI reporting, and network traffic simulation.
It is not clear if PPDR users can act as function developers since running a Network Application for their experimentation might require some functionalities that are not available in the testbeds and that cannot be provided by the project partners. This should be possible to maximize the impact of the project but requires establishing clear guidelines and interfaces to develop and test these modules prior to their onboarding on the platform.	The project and its Portal adopt the 'hybrid' option for interaction between the vertical application provider and the testbed operator, thereby enabling verticals to delegate part of, or their entire vertical application as part of the 5G-EPICENTRE platform (*i.e.*, hosted on the same cluster lifecycle as the 5G-EPICENTRE platform components), allowing them to also make them available to others. The delegation procedure ensures that testbed operators have full control over what components are orchestrated on their platforms.

2 Adherence to project specifications

In this Section, we explore how the current implementation of the 5G-EPICENTRE front end components supports project documentation with respect to requirements (Section 2.1) and architecture specifications (Section 2.2), as elaborated in the latest version of the 5G-EPICENTRE specifications (D1.4 *"Experimentation requirements and architecture specification final version"*).

2.1 5G-EPICENTRE requirements adherence

The 5G-EPICENTRE Portal design has been informed by the requirements first specified in D1.3, and validated in D1.4. We briefly discuss adherence to the project elicited requirements (including stakeholders' and platform requirements) in Table 3. In this Table, requirements are identified using the codes QR, FR, NFR defined in D1.3 The definition of these codes is given below:

- QR: stakeholders' requirement
- FR: functional requirement
- NFR: non-functional requirement

Table 3: 5G-EPICENTRE Portal adherence to stakeholders' and platform requirements

Req.	Description	Adherence	Referring document Sections
Stakeholders' requirements			
QR3	The platform should provide network resource repository and the ability to use.	The 5G-EPICENTRE Portal supports the capacity to use the Network Service Repository, by supporting all of its functions and API methods (see also D4.3 *"Curated Network Application image repository"*).	Section 4.2.1 Section 5.1
QR4	The platform should provide VNF/Network Applications repository and the ability to use.	The 5G-EPICENTRE Portal supports the capacity to use the Network Service Repository, by supporting all of its functions and API methods (see also D4.3 *"Curated Network Application image repository"*).	Section 4.2.1 Section 5.1
QR5	The platform should provide service on-boarding / parametrisation functionality.	Through the 5G-EPICENTRE Portal, the function developer is given the means to delegate their vertical application experimentation components to the most appropriate testbed, whereby it provides also methods for parameterising the underlying testbed environment.	Section 4.2.2 Section 5.1.2
QR6	Friendly user interface, guiding to perform testing.	The 5G-EPICENTRE Portal has been designed following fundamental principles of effective web design, informed by the user requirements.	Section 5

QR7	Visualization of KPIs measure and automated analysis.	The Portal supports different visualization components for KPIs presentation, alongside automated routines for generating reports pertaining to specific experiment execution requests.	Section 4.3 Section 5.1.4
QR11	On-line tools for one-stop reservation of facilities.	The 5G-EPICENTRE Portal is a web-based system developed for requesting, and monitoring execution of a 5G experiment on any one of the four 5G-EPICENTRE federated testbeds.	Section 2.2.1 Section 3 Section 4 Section 5
QR12	Avoid complexity and over-dimensioning.	Following fundamental principles of effective web design, the 5G-EPICENTRE Portal presents an intuitive user experience, abstracting much of the complexity involved in networking infrastructure configuration. It has been designed with the purpose of enabling users to order an experiment, without possessing knowledge on virtual or physical network infrastructure.	Section 5
QR13	Provide technical support for network configuration and service operation	The 5G-EPICENTRE Portal displays tooltip dialogue boxes, along with "Help" items to provide basic guidelines throughout the experiment journey. In addition, print documentation will be delivered to third parties during their experimentation phase.	Section 5
QR16	Remote monitor of testbed execution	The Portal provides means to monitor execution of an experiment in real time, in terms of collecting and provisioning data in rich visualisation structures.	Section 4.3 Section 5.1.4
QR20	Compliance to standards	The 5G-EPICENTRE Portal supports the RFC 7519 standard [1] for URL-safe user authentication.	Section 3.2.1 Section 4.1
		Platform requirements	
FR1	The system must allow a user to define experiments	The 5G-EPICENTRE Portal implements functionality for an experimenter to prepare an experiment, resulting in the generation of an experiment descriptor.	Section 4.2.2 Section 5.1.2
FR2	The system must allow a user to define experiment-specific KPIs	The 5G-EPICENTRE Portal supports experimentation in four pre-defined PPDR 5G scenarios, which involve particular KPIs gener-	Section 5

		ated at each testbed level based on the project use cases spawning these scenarios. Usage of the Analytics services network application allows experimenters to further supply the system with custom KPI calculations (see also D1.4).	
FR3	The system must support onboarding of network functions (NFs).	The 5G-EPICENTRE Portal implements functionality to delegate vertical application components (as Helm charts) on the specified infrastructure.	Section 4.2.1 Section 5.1
FR6	The system must support service function chaining of NFs into end-to-end services (Network Applications).	The 5G-EPICENTRE Portal implements functionality for accommodating the platform network applications, as defined in D1.4.	Section 4.2.2
FR10	The system should support remote access to the definition and monitoring of experiments	[**Direct relation to QR16**] The Portal provides means for experimenters to calibrate the underlying infrastructures towards defining their experiment deployments; as well as means to monitor execution of an experiment in real time, in terms of collecting and provisioning data in rich visualisation structures.	Section 4.2 Section 4.3 Section 5.1.4
FR11	The system should provide a proper abstraction of the underlying network technologies.	[**Direct relation to QR12**] Following fundamental principles of effective web design, the 5G-EPICENTRE Portal presents an intuitive user experience, abstracting much of the complexity involved in networking infrastructure configuration. It has been designed with the purpose of enabling users to order an experiment, without possessing knowledge on virtual or physical network infrastructure.	Section 5
FR12	The system should expose easy-to-consume APIs toward the experimenter.	The Portal consumes APIs by other 5G-EPICENTRE functional blocks, encapsulating API calls in simple GUI operations. In addition, the Portal implements several API calls for supporting its own internal structures.	Section 3.2
FR16	The system should expose a requirements catalogue for the underlying network resources.	[**Direct relation to QR3**] The 5G-EPICENTRE Portal supports the capacity to use the Network Service Repository, by supporting all of its functions and API methods (see also	Section 4.2.1 Section 5.1

		D4.3 *"Curated Network Application image repository"*).	
FR17	The system should provide a user with appropriate network resource inventories and means to configure and (re)use them.	[**Direct relation to QR4**] The 5G-EPICENTRE Portal supports the capacity to use the Network Service Repository, by supporting all of its functions and API methods (see also D4.3 *"Curated Network Application image repository"*).	Section 4.2.1 Section 5.1
FR18	The system should allow experimenters to repeat and re-parameterise experiments.	The 5G-EPICENTRE Portal enables the user to store experiments on their profile, which can be repeated, modified, deleted or used as templates for future experiments (*e.g.*, when an application is updated).	Section 4.2.2
FR19	The system should provide means to customise network slices for Network Application requirements under eMBB, URLLC and mMTC service types.	The 5G-EPICENTRE Portal implements functionality for accommodating the platform network applications, as defined in D1.4.	Section 4.2.2
FR25	The system should translate analytics into visualization formats suitable for interpretation by a human being.	The Portal supports different visualization components for KPIs presentation, alongside automated routines for generating reports pertaining to specific experiment execution requests.	Section 4.3 Section 5.1.4
FR26	The system should support the capability of producing customized reports based on users' needs and preferences.	The Portal supports different visualization components for KPIs presentation, alongside automated routines for generating reports pertaining to specific experiment execution requests.	Section 4.3 Section 5.1.4
FR27	The system should provide guidance to the user in order to train them on using it.	[**Direct relation to QR13**] The 5G-EPICENTRE Portal displays tooltip dialogue boxes, along with "Help" items to provide basic guidelines throughout the experiment journey.	Section 5
FR30	The system could enable the calibration of individual testbed components from a singular control point.	Through the 5G-EPICENTRE Portal, experimenter is provisioned the means to delegate their vertical application experimentation components to the most appropriate testbed. The Portal further provides methods for parameterising the underlying testbed environment for creating custom experimental conditions.	Section 4.2

FR31	The system could support role-based access control (RBAC) policies.	The 5G-EPICENTRE Portal implements RBAC by means of the user authentication and authorization routines and APIs.	Section 3.2.1 Section 4.1
NFR1	The system must be secure.	[**Direct relation to FR31**] The 5G-EPICENTRE Portal implements RBAC by means of the user authentication and authorization routines and APIs.	Section 3.2.1 Section 4.1
NFR3	The system must be privacy-compliant.	The 5G-EPICENTRE Portal supports compliance with privacy regulations by means of its RBAC policy, and ensures that there is not "leak" of information related to an experiment or experimenter in a way that compromises personal data or intellectual property.	Section 3.2.1 Section 4.1 Section 4.2
NFR4	The system must be performant/responsive.	All actions that can be performed inside the 5G-EPICENTRE Portal have been designed to be responsive and clearly present outcomes of such actions in a manner expected by the end-user, to facilitate their experience.	Section 5
NFR6	The system should be user-friendly.	[**Direct relation to QR6**] The 5G-EPICENTRE Portal has been designed following fundamental principles of effective web design, informed by the user requirements.	Section 5
NFR9	The system could be documented.	The present deliverable, designated as 'public', provides the system documentation.	Section 5

2.2 Relation to the 5G-EPICENTRE architecture

With respect to the latest 5G-EPICENTRE architecture documentation (D1.4), the core front-end functional blocks are depicted in Figure 1.

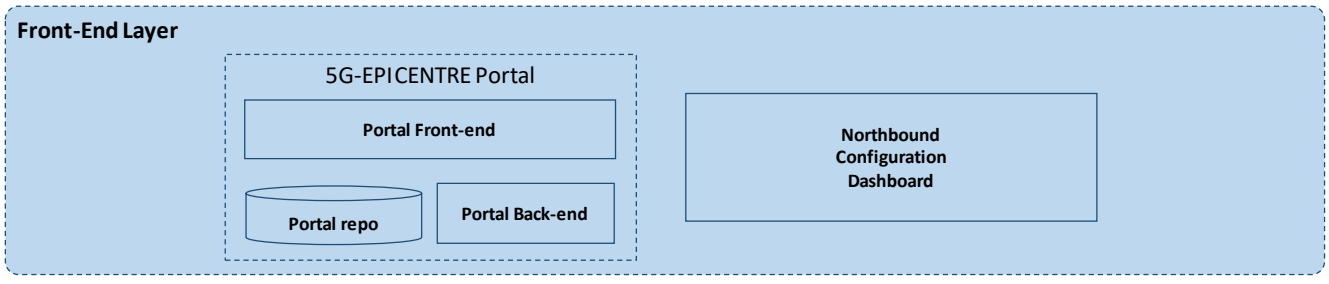

Figure 1: 5G-EPICENTRE Front-end components in the overall 5G-EPICENTRE architecture (D1.4).

2.2.1 5G-EPICENTRE Portal

The Portal is seen as the overall solution to enable the user to carry out the four (4) indicative functional scenarios of the 5G-EPICENTRE platform, *i.e.*, 1) **Delegating vertical application components to the testbed operator**; 2) **Scheduling an experiment**; 3) **Experiment deployment**; and 4) **Experiment execution**.

The Portal implements a client/server design architecture, consisting of the following applications:

- The **client (frontend - not to be confused with the 5G-EPICENTRE Front-end Layer)**, which is a single-page client application, which uses HTML and TypeScript, running on the end-user's browser and providing the graphical user interface (GUI) environment for enabling interaction with the system's controls. Built using the Angular framework, the Portal client is a collection of Angular Components (representing entire pages, or partial components that are embedded into others), Services and auxiliary modules, whilst also making use of the user's machine temporary local storage.
- The **server (backend – not to be confused with the 5G-EPICENTRE Back-end Layer)**, is a Node.js application that serves the requests from the frontend. It leverages several frameworks, most notably the *Express*[2] web application framework for robust API creation, *Mongoose*[3] for modelling application data and mapping to MongoDB document collections for persistent storage, and *mqtt*[4] and *jsonwebtoken*[5] library packages for implementing the MQTT OASIS standard messaging protocol (for asynchronous communication with the 5G-EPICENTRE Back-end Layer services that use it); and JSON Web Token (JWT) open standard for secure transmission of information between the Portal client and server.

The high-level architecture of the Portal implementation is illustrated in Figure 2. As a core software element of this deliverable, the Portal is described in more detail in Section 3.

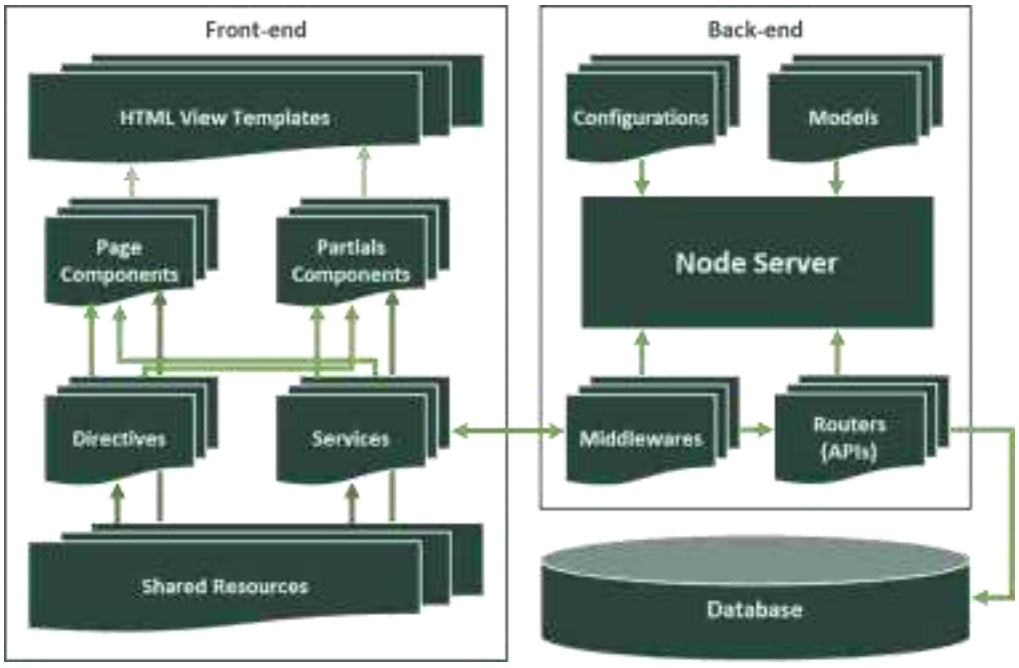

Figure 2: High-level architecture of the 5G-EPICENTRE Portal

[2] https://expressjs.com/
[3] https://mongoosejs.com/
[4] https://www.npmjs.com/package/mqtt
[5] https://www.npmjs.com/package/jsonwebtoken

2.2.2 Northbound configuration dashboard

The Northbound Configuration Dashboard (NCD) is a Network Application specifically designed to empower experimenters with the ability to request and configure specific Quality of Service (QoS) parameters based on the 5G QoS Indicator (5QI) standards for running their experiments. The NCD serves as a crucial tool for experimenters, enabling them to ensure optimal performance, reliability, and efficiency of their experiments within the network environment.

In 5G networks, 5QI is a parameter that is used to classify and prioritise different types of network traffic based on their QoS requirements. It helps in defining the service level and the treatment of data packets within the network. The 5QI value is an indicator that ranges from 1 to 255 and represents different QoS levels. A higher 5QI value generally corresponds to a higher priority or better QoS treatment for the associated traffic. The specific mapping of 5QI values to QoS characteristics, such as latency, packet loss, and throughput, is defined by the 3rd Generation Partnership Project (3GPP) standards organisation [2]. By assigning appropriate 5QI values to different types of traffic, 5G networks can efficiently allocate network resources, prioritise critical applications, and deliver an optimal QoS experience for various services, including voice, video, Internet of Things (IoT), and other data applications.

The ability to request specific QoS settings through the NCD is of paramount importance for several reasons. Firstly, experiments often have unique requirements for network performance, such as low latency, high bandwidth, or minimal packet loss. By providing experimenters with the option to request specific QoS parameters, the NCD allows them to tailor the network conditions precisely to their experiment's needs. This customisation ensures that the experimenters can accurately replicate real-world scenarios and obtain accurate results.

Secondly, the use of the 5QI standards in the NCD brings a standardised approach to QoS management in 5G networks. Experimenters can leverage the defined 5QI values to classify and prioritise their traffic, ensuring consistent and predictable QoS treatment within the network. This standardisation promotes interoperability, simplifies configuration, and enhances overall network management efficiency.

Moreover, the NCD plays a vital role in ensuring fair resource allocation and efficient network utilisation. By enabling experimenters to request specific QoS parameters, the NCD facilitates resource allocation decisions based on experiment priorities. It allows experimenters to optimise their resource utilisation and obtain the necessary network conditions to conduct their experiments effectively.

Furthermore, the NCD enhances the overall user experience and satisfaction by minimising unexpected network performance issues during experiments. Experimenters can proactively request and configure the desired QoS parameters through the NCD, reducing the risk of encountering suboptimal network conditions that could adversely impact their experiments' outcomes.

In summary, the NCD serves as a Network Application that empowers experimenters to request, via a user-friendly interface, specific QoS settings based on the 5QI standards. By offering this capability, the NCD enables experimenters to achieve precise control over network conditions, replicate real-world scenarios, promote standardisation, optimise resource allocation, and ensure reliable and efficient experimentation outcomes.

Step 1: In the "5QI Request" tab, the experimenter selects from a drop-down of predefined 5QI values the appropriate 5QI value for their experiment, inputs the experiment id and submits the request. The NCD contacts the Npcf_PolicyAuthorization Service API[6] to submit the request for processing. The experimenter can also view the history of requests, 5QI values requested and associated QoS settings (Figure 3 and Figure 4).

[6] https://jdegre.github.io/editor/?url=https://raw.githubusercontent.com/jdegre/5GC_APIs/master/TS29514_Npcf_PolicyAuthorization.yaml

Figure 3: Northbound Configuration Dashboard mock-up for Portal integration – Selecting an applicable value (1/2)

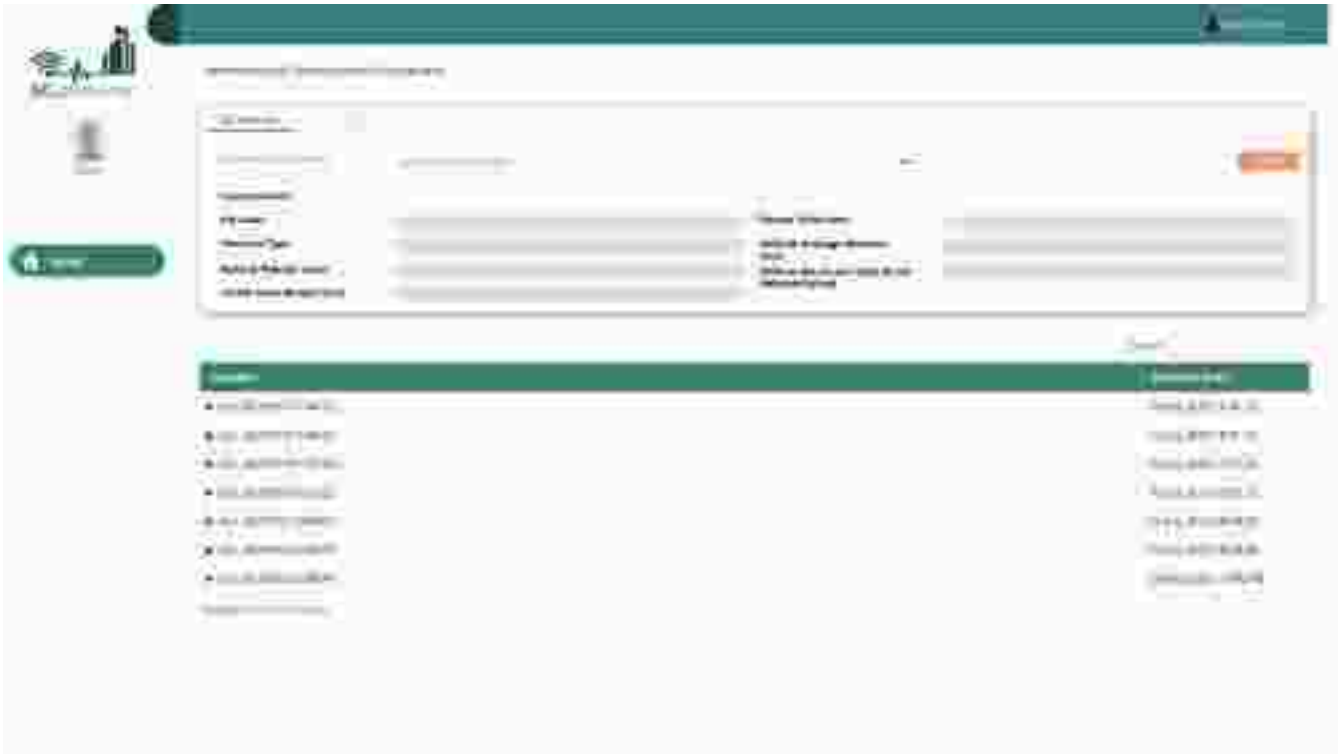

Figure 4: Northbound Configuration Dashboard mock-up for Portal integration – Selecting an applicable value (2/2)

Step 2: The experimenter can access the "Notes" tab to view the "Standardized 5QI to QoS characteristics mapping" to understand which 5QI value is the most appropriate for their experiments (Figure 5).

Figure 5: Northbound Configuration Dashboard mock-up for Portal integration – Notes tab

3 5G-EPICENTRE Portal implementation

This Section presents the fully realised final version of the 5G-EPICENTRE Portal at M30, and provides details on its final implementation. The 5G-EPICENTRE Portal (henceforth referred to as *"The Portal"*) is a distributed web-based client/server application developed for requesting, and monitoring execution of a 5G experiment on any one of the four 5G-EPICENTRE federated testbeds.

The project was built using several open-source and source-available technologies, including the **Angular**[7] web framework (v15), **Node.js**[8] server environment (v20), and **MongoDB**[9] database program.

3.1 Frontend

The Portal frontend was generated with Angular CLI version 15.2.5. It is written entirely in TypeScript and HTML/CSS. It delivers the client application logic mapping onto the HTML document object model (DOM) programming interface, thus defining all the HTML elements to be displayed (*i.e.*, when the page is loaded onto a browser), along with the properties and behaviours of those elements.

Like all Angular projects, the Portal frontend application is organised into a list of components, which define application data, logic and screen visual output. Following Angular coding best practices, the Portal frontend components are organised into *pages* and *partials*. Page components implement functionality delivered inside an entire page, which can be navigated to using the browser address bar (or via the Angular router service module). A Partial component, on the other hand, implements self-contained functionality that controls the look and behaviour of a particular region of the screen (HTML DOM element), which can be embedded inside other partial or page components (a search bar, stylised button, widget, *etc.*).

In addition, the Portal implements a number of Angular injectable services, which handle functionality such as determining visibility of partial components (*e.g.*, a loading screen whilst HTTP requests are executing), or sending and retrieving data to and from the backend. Finally, additional code implements necessary models, constants, validation mechanisms and Angular interceptors (*i.e.*, injectable structures that intercept HTTP requests and process requests and responses before they arrive at their intended destination, *i.e.*, the frontend or the backend side).

In addition to Angular core modules (*NgModules*), the frontend application imports functionality from the following external resources, leveraging open-source Angular NgModules:

- **ApexCharts** (*NgApexchartsModule*)[10], which streamlines creation of responsive charts, diagrams, graphs, and other information visualizations.
- **Leaflet**[11], which provides a JavaScript library for interactive map visualizations.

3.1.1 Pages

The Portal frontend consists of the following Page components:

- **Home:** Landing page for signed-in users, it provides a dashboard GUI for conveying latest key information to the user in one convenient place.
- **Login:** landing page for non-signed-in users, it provides a login form, alongside means to register a new user account to gain access to Portal services as an experimenter.

[7] https://angular.io/
[8] https://nodejs.org/en
[9] https://www.mongodb.com/
[10] https://apexcharts.com/
[11] https://www.npmjs.com/package/leaflet

- **Notifications:** Displays a list of all the automated notifications that the Portal has generated for this user, alongside means to manage them (*e.g.*, delete, mark them as "read", or access their contents).
- **Experiments:** Displays a list of all the experiments that the experimenter has requested using the Portal. In the case of testbed owner accounts, the page displays all experiments that have been requested to be deployed on the testbed with which the account is associated with. Alongside access to each experiment (see **Experiment** page below), this page further provides means to quickly discard, or otherwise manage experiments (*e.g.*, download descriptor information).
- **Experiment:** Displays detailed information about a requested experiment, alongside access to the experiment's descriptor file. It further includes a submission history and access to the experiment cancellation, or reviewing (in case of a testbed owner account) interface.
- **New Experiment:** Implements a "wizard" that guides the user (experimenter) through a series of steps on requesting a new experiment execution.
- **Revise experiment:** A redacted version of the **New Experiment** page, where the user (experimenter) can modify information about their previously requested experiments.
- **Experiment insights:** Displays detailed information about an experiment execution, either in real-time (*i.e.*, while the experiment is still being executed, and metrics are being generated and sent to the Portal), or for any past experiment that remains accessible through the Portal environment (*i.e.*, the user has not deleted the experiment page from their profile).
- **Resources:** Displays a list of all the vertical application artefacts (which, as specified in D1.4, are stored within the platform in the form of Helm charts) that the experimenter has requested to delegate to testbed owners using the Portal. In the case of testbed owner accounts, the page displays all vertical application artefact requests. Alongside access to each artefact (see 'Resource' page below), this page further provides means to quickly discard, or otherwise manage artefact requests (*e.g.*, download the artefact's Helm chart). It is worth noting that, whereas experiments are addressed to each testbed owner independently (based on PPDR scenario selection), artefact delegation requests are visible by all testbed owner accounts, since each should verify artefact deployment in their own testbed.
- **Resource:** Displays detailed information about a requested vertical application deployment package (*i.e.*, Helm chart), alongside access to the actual downloadable compressed file. It further includes a submission history and access to the cancellation, or reviewing (in case of a testbed owner account) interface.
- **Resource delegation:** Implements a 'wizard' that guides the user (experimenter) through a series of steps on requesting a new vertical application delegation request. It further integrates revision functionality, allowing experimenters to modify information about their previously requested artefact delegations.

3.2 Backend

The Portal backend is a Node.js application that implements core functionalities pertaining to Portal Tasks (such as maintaining the database for persistence of data created by frontend users) and makes these resources available to the client side. It utilises the express framework for streamlining API implementation, alongside Cross-Origin Resource Sharing (CORS, via the *cors* node.js package[12]) to enable otherwise restricted resources (such as those coming from the 5G-EPICENTRE Portal Back-end Layer) to be accessed from the Portal's domain. The backend further implements the Portal's authentication mechanism (see also Section 4.1) and the function to connect to the database. Through the mongoose library, it is possible for the Portal to create and manage collections (a *collection* in MongoDB is the entity that stores one or more data records, known as *documents*). It finally implements an MQTT client for connecting to the external RabbitMQ[13] message broker published to by

[12] https://www.npmjs.com/package/cors
[13] https://www.rabbitmq.com/

the Analytics Aggregator component (see Section 4.3.1, and deliverable D1.4), subscribing to its topic, and consuming messages from its queue.

Practically, the Portal backend can be interpreted as one more 5G-EPICENTRE Back-end Layer service that the frontend client connects to, in addition to the Experiment Coordinator and Network Service Repository.

The Portal backend exposes several Representational State Transfer (REST) APIs towards the Portal frontend, which are documented in the following Sub-sections.

3.2.1 User APIs

User APIs implement an express router with all endpoints following **"/api/users"**. They define all APIs used to authenticate and manage user logins and registrations. By design, the Portal does not allow multiple accounts to be created with one email address, therefore enabling unique user search to be based on email credentials.

3.2.1.1 Login user

This API is used to authenticate a user in the Portal, using a set of stored credentials. The API documentation is summarised in Annex I.

It first retrieves the request parameters *"email"* and *"password"* from the request body, and attempts to locate the user, whose *"email"* matches the ones in the request body in the *"users"* collection. If such a user exists in the database, and their password matches the one encrypted in the database, a JWT is generated for the user.

The response for this API contains a custom *'User'* Mongoose model (a *Model* in Mongoose defines and stores a JSON copy of a document), together with the signed token.

3.2.1.2 Register new user

This API is used to register a user in the Portal, using a set of provided credentials. The API documentation is summarised in Annex I.

It first de-structures the request body, and attempts to locate a user, whose *"email"* matches the one in the request body in the *"users"* collection. If such a user does not exist in the database, the provided password is encrypted using a cryptographic hashing algorithm[14], and a new User Model is created based on inputs from the request body. The Model is used to create the database record (document) in the *"users"* collection.

The response for this API contains this User model, together with the signed token (similar to the login API, effectively logging the user in upon successful registration).

3.2.2 Experiments API

The Experiments API implements an express router with all endpoints following **"/api/experiments"**. It defines all APIs used to manage experiment requests.

3.2.2.1 Get all

This API is used to retrieve all the experiments that are stored in the *"experiments"* collection inside the database, and which are associated with this user. The API documentation is summarised in Annex I.

The server first determines whether the requesting user is a testbed owner account, by checking the *"isAdmin"* property of the signed JWT included in the request header. In case this property is set to *true*, a *find()* mongoose query is created for retrieving a list of documents for which the *"testbed"* field matches the *"testbed"* field in the signed JWT of the user querying for those data. If the "*isAdmin"* property is set to *false* (*i.e.*, the requesting user

[14] https://www.npmjs.com/package/bcryptjs

is an experimenter), a different *find*() mongoose query is created for retrieving a list of documents for which the *"user"* field matches the *"id"* field in the signed JWT of the user querying for those data.

In either case, the response for this API contains a list of documents (formatted using a custom *'Experiment'* Mongoose model) that match the *find()* query. If the query is not successful, the response contains an empty array ([]).

3.2.2.2 Get by search term

This API is used to retrieve all the experiments that are stored in the *"experiments"* collection inside the database, which are associated with this user, and which match a particular given search term. The API documentation is summarized in Annex I.

The server first constructs a regular expression from the search term parameter, and makes it case-insensitive. It then determines whether the requesting user is a testbed owner account, by checking the *"isAdmin"* property of the signed JWT included in the request header. In case this property is set to *true*, a *find()* mongoose query is created for retrieving a list of documents for which the *"testbed"* field matches the *"testbed"* field in the signed JWT of the user querying for those data; and where the *"title"* field matches the pattern in the regular expression. If the "*isAdmin"* property is set to *false* (*i.e.*, the requesting user is an experimenter), a different *find*() mongoose query is created for retrieving a list of documents for which the *"user"* field matches the *"id"* field in the signed JWT of the user querying for those data (similarly matching the *"title"* parameter to the regular expression pattern).

In either case, the response for this API contains a list of documents (formatted using the Experiment Mongoose model) that match the *find()* query. If the query is not successful, the response contains an empty array ([]).

3.2.2.3 Get by ID

This API is used to retrieve a single experiment stored in the *"experiments"* collection inside the database, which matches a particular given string identifier. The API documentation is summarised in Annex I.

The server extracts the experiment ID from the request parameters, and creates a *findById()* mongoose query for retrieving a single document whose *"_id"* field matches the "experimentId" request parameter.

The response for this API contains a single document (formatted using the Experiment Mongoose model) that matches the *findById()* query. If the query is not successful, the response contains an *undefined* variable.

3.2.2.4 Submit an experiment request

This API is used to submit a new experiment request to the Portal. The API documentation is summarised in Annex I.

It first de-structures the request body, and attempts to locate whether there is already an experiment with this name for this user in the database (querying by the experiment title). If the experiment does not exist, a new Experiment Model is created, based on the inputs from the request body, and is assigned the user's id from the *"id"* property of the signed JWT included in the request header. The new Model is then stored as a document into the *"experiments"* collection, inserting thus the new document into the database.

The response for this API contains the newly created Experiment model.

3.2.2.5 Update an experiment request

This API is used to submit a revision to an existing experiment request created using the Portal. The API documentation is summarised in Annex I.

It first de-structures the request body, and attempts to locate whether there is already an experiment with this id in the database (querying by the experiment id). If the experiment exists, an *updateOne()* mongoose query is

created for updating the document for which the *"_id"* field matches the *"id"* field in the request body, and setting all specified parameters to their intended new values supplied in the request body.

The response for this API contains the result of the update operation.

3.2.2.6 Remove an experiment request

This API is used to remove/delete an existing experiment request created using the Portal. The API documentation is summarized in Annex I.

It first de-structures the request body, and attempts to locate whether there is already an experiment with this id in the database (querying by the experiment id). If the experiment exists, the backend deletes the first document that matches the previous search criteria.

The response for this API contains the newly deleted Experiment model.

3.2.3 Resources API

The Resources API implements an express router with all endpoints following **"/api/resources"**. They define all APIs used to manage vertical application artefact delegation requests.

3.2.3.1 Get all

This API is used to retrieve all the vertical application artefact delegation requests that are stored in the *"helm-chart-artefacts"* collection inside the database, and which are associated with this user. The API documentation is summarised in Annex I.

The server first determines whether the requesting user is a testbed owner account, by checking the *"isAdmin"* property of the signed JWT included in the request header. In case this property is set to *true*, a blank *find()* mongoose query is created for retrieving all data (since application artefact delegation requests are addressed to all testbed owners regardless of the testbed which they administrate). If the "*isAdmin"* property is set to *false* (*i.e.*, the requesting user is an experimenter), a different *find*() mongoose query is created for retrieving a list of documents for which the *"user"* field matches the *"id"* field in the signed JWT of the user querying for those data.

In either case, the response for this API contains a list of documents (formatted using a custom *'HelmChart'* Mongoose model) that match the *find()* query. If the query is not successful, the response contains an empty array ([]).

3.2.3.2 Get by search term

This API is used to retrieve all the vertical application artefact delegation requests that are stored in the *"helm-chart-artefacts"* collection inside the database, which are associated with this user, and which match a particular given search term. The API documentation is summarised in Annex I.

The server first constructs a regular expression from the search term parameter, and makes it case-insensitive. It then determines whether the requesting user is a testbed owner account, by checking the *"isAdmin"* property of the signed JWT included in the request header. In case this property is set to *true*, a *find()* mongoose query is created for retrieving a list of documents for which the *"title"* field matches the pattern in the regular expression. If the "*isAdmin"* property is set to *false* (*i.e.*, the requesting user is an experimenter), a different *find*() mongoose query is created for retrieving a list of documents for which the *"user"* field matches the *"id"* field in the signed JWT of the user querying for those data, also matching the *"title"* parameter to the regular expression pattern.

In either case, the response for this API contains a list of documents (formatted using the HelmChart Mongoose model) that match the *find()* query. If the query is not successful, the response contains an empty array ([]).

3.2.3.3 Get by ID

This API is used to retrieve a single vertical application artefact delegation request stored in the *"helmchart-artefacts"* collection inside the database, which matches a particular given string identifier. The API documentation is summarised in Annex I.

The server extracts the resource ID from the request parameters, and creates a *findById()* mongoose query for retrieving a single document whose *"_id"* field matches the *"resourceId"* request parameter.

The response for this API contains a single document (formatted using the HelmChart Mongoose model) that matches the *findById()* query. If the query is not successful, the response contains an *undefined* variable.

3.2.3.4 Get by tag name

This API is used to retrieve all the resources that are stored in the *"helmchart-artefacts"* collection inside the database, which are associated with this user, and for which the user-defined *"category"* parameter includes elements that match a particular given tag name. The API documentation is summarised in Annex I.

The server first determines whether the requesting user is a testbed owner account, by checking the *"isAdmin"* property of the signed JWT included in the request header. In case this property is set to *true*, a *find()* mongoose query is created for retrieving a list of documents for which the *"category"* field contains elements that match the given *"tagName"* value in the request parameters. If the "*isAdmin"* property is set to *false* (*i.e.*, the requesting user is an experimenter), a different *find*() mongoose query is created for retrieving a list of documents for which the *"user"* field matches the *"id"* field in the signed JWT of the user querying for those data, also matching the *"category"* parameter elements to the *"tagName"* value in the request parameters.

The response for this API contains a list of documents (formatted using the HelmChart Mongoose model) that match the *find()* query. If the query is not successful, the response contains an empty array ([]).

3.2.3.5 Submit a vertical application artefact delegation request

This API is used to submit a new vertical application artefact delegation request to the Portal. The API documentation is summarised in Annex I.

It first de-structures the request body, and attempts to locate whether there is already a vertical application artefact with this name for this user in the database (querying by the vertical application artefact *"title"*). If the resource does not exist, a new HelmChart Model is created, based on the inputs from the request body, and is assigned the user's id from the *"id"* property of the signed JWT included in the request header. The new Model is then stored as a document into the "*helmchart-artefacts*" collection, inserting thus the new document into the database.

The response for this API contains the newly created HelmChart model.

3.2.3.6 Update a resource request

This API is used to submit a revision to an existing vertical application artefact delegation request created using the Portal. The API documentation is summarised in Annex I.

It first de-structures the request body, and attempts to locate whether there is already an existing vertical application artefact delegation request with this id in the database (querying by the *"id"* parameter in the request body). If the vertical application artefact delegation request exists, an *updateOne()* mongoose query is created for updating the document for which the *"_id"* field matches the *"id"* field in the request body, and setting all specified parameters to their intended new values supplied in the request body.

The response for this API contains the result of the update operation.

3.2.3.7 Remove a resource request

This API is used to remove/delete an existing vertical application artefact delegation request created using the Portal. The API documentation is summarised in Annex I.

It first de-structures the request body, and attempts to locate whether there is already a resource with this id in the database (querying by the "id" parameter in the request body). If the vertical application artefact delegation request exists, the backend deletes the first document that matches the previous search criteria.

The response for this API contains the result of the delete operation.

3.2.3.8 Upload temporary chart package

This API is used to upload a file to a temporary storage in the Portal backend *(/uploads* folder). The API documentation is summarised in Annex I.

The server retrieves the file object, and checks the filename extension to validate it as a helm chart (only allowing file extensions like .zip, .tgz and .gz). It then creates the local path string where the file will be moved, and proceeds to send the file from the temporary path to the newly created path within the /uploads folder.

The response for this API contains the full path to the /uploads folder, where the file is temporarily stored.

3.2.3.9 Download temporary chart package

This API is used to download a file from the temporary storage in the Portal backend *(/uploads* folder). The API documentation is summarised in Annex I.

The server retrieves the filename from the request parameters and re-creates the local path string where the file is located. It then proceeds to transfer the file at the given path as an attachment to the response.

3.2.3.10 Delete temporary chart package

This API is used to delete a file from the temporary storage in the Portal backend *(/uploads* folder). The API documentation is summarised in Annex I.

The server retrieves the filename from the request parameters and re-creates the local path string where the file is located. It then proceeds to asynchronously remove the file at the given path.

The response for this API contains the full path to the /uploads folder, where the file was temporarily stored.

3.2.4 Reports API

Implements an express router with all endpoints following **"/api/ reports"**. It defines all APIs used to manage experiment reports created automatically whenever metrics are sent to the Portal by the Aggregator Back-end Layer component.

3.2.4.1 Get by ID

This API is used to retrieve a single experiment report stored in the *"experiment-reports"* collection inside the database, which matches a potential set of given string identifiers. The API documentation is summarised in Annex I.

The server extracts the IDs from the request parameters, and creates a *findOne()* mongoose query for retrieving a single document whose *"experiment_id"* field matches the *"experimentId"* request parameter, and (potentially) the *"execution_id"* field matches the *"executionId"* request parameter.

The response for this API contains a single document (formatted using a custom *'ExperimentReport'* Mongoose model) that matches the *findOne()* query. If the query is not successful, the response contains an *undefined* variable.

3.2.4.2 Delete report

This API is used to remove/delete an existing experiment report from the *"experiment-reports"* collection inside the database. The API documentation is summarised in Annex I.

The server extracts the IDs from the request parameters, and creates a *deleteOne()* mongoose query for removing a single document whose *"experiment_id"* field matches the *"experimentId"* request parameter, and (potentially) the *"execution_id"* field matches the *"executionId"* request parameter.

The response for this API contains the result of the delete operation.

3.2.5 Notifications API

Implements an express router with all endpoints following **"/api/notifications"**. It defines all APIs used to manage Portal notifications.

3.2.5.1 Get all

This API is used to retrieve all the notifications addressed to this user, that are stored in the 'notifications' collection inside the database. The API documentation is summarized in Annex I.

The server first determines whether the requesting user is a testbed owner account, by checking the *"isAdmin"* property of the signed JWT included in the request header. In case this property is set to *true*, a *find()* mongoose query is created for retrieving a list of documents for which the *'testbed'* field either matches the 'testbed' field in the signed JWT of the user querying for those data, or is set to *'None'* (for vertical application artefact delegation requests). In addition, the query checks whether the notification has been marked for deletion by the testbed owner (*"admin_remove"* parameter must be *false*).

If the "*isAdmin"* property is set to *false* (*i.e.*, the requesting user is an experimenter), a different *find*() mongoose query is created for retrieving a list of documents for which either the 'user' field matches the 'id' field in the signed JWT of the user querying for those data, or the 'correspondent' field matches the 'username' field in the JWT. In addition, the query checks whether the notification has been marked for deletion by the experimenter (*"origin_remove"* parameter must be *false*).

In either case, the response for this API contains a list of documents (formatted using the Notification Mongoose model) that match the *find()* query. If the query is not successful, the response contains an empty array ([]).

3.2.5.2 Get all unread

This API is used to retrieve all the unread notifications addressed to this user, and which are stored in the 'notifications' collection inside the database. The API documentation is summarized in Annex I.

The server first determines whether the requesting user is a testbed owner account, by checking the *'isAdmin'* property of the signed JWT included in the request header. In case this property is set to *true*, a *find()* mongoose query is created for retrieving a list of documents for which the *'testbed'* field either matches the 'testbed' field in the signed JWT of the user querying for those data, or is set to *'None'* (for vertical application artefact delegation requests). In addition, the query checks whether the notification has been marked for deletion by the testbed owner (*"admin_remove"* parameter must be *false*), and also, whether the *'admin_read'* field (denoting whether the testbed owner has read the contents of the notification, or has marked them as 'read') is set to *false*.

If the '*isAdmin'* property is set to *false* (*i.e.*, the requesting user is an experimenter), a different *find*() mongoose query is created for retrieving a list of documents for which either the 'user' field matches the 'id' field in the signed JWT of the user querying for those data (or the 'correspondent' field matches the 'username' field in the JWT). In addition, the query checks whether the notification has been marked for deletion by the experimenter

(*"origin_remove"* parameter must be *false*), and also, whether the *'origin_read'* field (denoting whether the experimenter has read the contents of the notification, or has marked them as 'read') is set to *false*.

In either case, the response for this API contains a list of documents (formatted using the Notification Mongoose model) that match the *find()* query. If the query is not successful, the response contains an empty array ([]).

3.2.5.3 Get by search term

This API is used to retrieve all the notifications addressed to this user, that are stored in the 'notifications' collection inside the database, and which match a particular given search term. The API documentation is summarized in Annex I.

The server first constructs a regular expression from the search term parameter, and makes it case-insensitive. It then determines whether the requesting user is a testbed owner account, by checking the *'isAdmin'* property of the signed JWT included in the request header. In case this property is set to *true*, a *find()* mongoose query is created for retrieving a list of documents for which the *'testbed'* field either matches the 'testbed' field in the signed JWT of the user querying for those data, or is set to *'None'* (for vertical application artefact delegation requests). In addition, the *'text'* field should match the pattern in the regular expression, and the notification should not have been marked for deletion by the testbed owner (*"admin_remove"* parameter must be *false*).

If the '*isAdmin'* property is set to false (*i.e.*, the requesting user is an experimenter), a different *find*() mongoose query is created for retrieving a list of documents for which the 'user' field matches the 'id' field in the signed JWT of the user querying for those data (or the 'correspondent' field matches the 'username' field in the JWT), and similarly matching the 'text' parameter to the regular expression pattern. In addition, the notification should not have been marked for deletion by the experimenter (*"origin_remove"* parameter must be *false*).

In either case, the response for this API contains a list of documents (formatted using the Notification Mongoose model) that match the *find()* query. If the query is not successful, the response contains an empty array ([]).

3.2.5.4 Send notification

This API is used to create a new notification in the database, making it available to the correspondents that should be alerted to it. The API documentation is summarized in Annex I.

It creates a new Notification Model based on the inputs from the request body, and assigns it the user's *"id"* and *"username"* properties of the signed JWT included in the request header. The new Model is then stored as a document into the *"notifications"* collection, inserting thus the new document into the database.

The response for this API contains the newly created Notification model.

3.2.5.5 Update all notifications

This API is used to mark all notifications addressed to this user as "read", from their point of view. The API documentation is summarised in Annex I.

The server first determines whether the requesting user is a testbed owner account, by checking the *"isAdmin"* property of the signed JWT included in the request header. In case this property is set to *true*, an *updateMany()* mongoose query is created for updating all the documents for which the *"testbed"* field matches the *"testbed"* field in the signed JWT of the user querying for those data, by setting their *"admin_read"* value to true. If the "*isAdmin"* property is set to *false* (*i.e.*, the requesting user is an experimenter), a different *updateMany*() mongoose query is created for updating all the documents for which either the *"user"* field matches the *"id"* field in the signed JWT of the user querying for those data, or the *"correspondent"* field matches the *"username"* field in the JWT. In this case, the updated documents will have their *"origin_read"* value set to *true*.

In either case, the response for this API contains a JSON Object that indicates (among other info) the number of modified documents in the database.

3.2.5.6 Update notification

This API is used to mark a particular notification accessed by this user as "read", from their point of view. The API documentation is summarised in Annex I.

The server first determines whether the notification exists, by constructing a *findOne()* mongoose query for retrieving a single document for which the *"_id"* field matches the *"id"* field of the notification inside the request body. If the notification exists, a check is made for whether the requesting user is a testbed owner account, by checking the *"isAdmin"* property of the signed JWT included in the request header. In case this property is set to *true*, an *updateOne()* mongoose query is created for updating the document for which the *"_id"* field matches the "id" field in the request body, and setting its *"admin_read"* value to *true*. If the "*isAdmin"* property is set to *false* (*i.e.*, the requesting user is an experimenter), a different *updateOne*() mongoose query is created for updating the document, which will have its *"origin_read"* value set to *true*.

In either case, the response for this API contains a JSON Object that indicates (among other info) the result of the update operation in the database.

3.2.5.7 Mark for removal/Remove a notification

This API is used to remove/delete a notification created using the Portal. The action will result in the omission of this notification from all GET operations that this user will request in the future. For a notification to be completely removed from the database, both correspondents (testbed owner and experimenter) must have marked the same notification for permanent removal. The API documentation is summarized in Annex I.

The server first determines whether the notification exists, by constructing a *findOne()* mongoose query for retrieving a single document for which the *"_id"* field matches the *"id"* field of the notification inside the request body. If the notification exists, a check is made for whether the requesting user is a testbed owner account, by checking the *"isAdmin"* property of the signed JWT included in the request header. In case this property is set to *true*, an *updateOne()* mongoose query is created for updating the document for which the *"_id"* field matches the *"id"* field in the request body, and setting its *"admin_remove"* value to *true*. If the "*isAdmin"* property is set to *false* (*i.e.*, the requesting user is an experimenter), a different *updateOne*() mongoose query is created for updating the document, which will have its *"origin_remove"* value set to *true*.

If following the operation, both *"admin_remove"* and *"origin_remove"* values are set to *true*, the backend issues a *deleteOne()* query that deletes the document with the specified *"_id"* parameter. Depending on the final operation, the response for this API contains the result of that operation.

3.3 Deployment

Both Portal applications (frontend and backend) can be found in the 5G-EPICENTRE GitLab source code repository platform (at https://gitlab.5gepicentre.eu/kapostol/5gepicentre-portal.git), alongside comprehensive instructions for installing both applications in a production environment.

In accordance with the specifications of the deployment architectural view presented in D1.4, both Portal applications have been deployed on the UMA platform.

4 Portal processes and information flows

In this Section, the basic principles and workflows of the Portal will be discussed with respect to its supported functions and platform downstream/upstream information flows (see also D4.4 and D4.5). Specifically, the following Sub-sections will discuss how the Portal implements and ensures processes of authentication and authorisation as part of the holistic security considerations (see also D2.7), as well as how it supports integration with other components of the 5G-EPICENTRE architectural stack for setting up and calibrating underlying infrastructure components (*downstream* flow); along with presenting data generated at the infrastructures to the end user (*upstream* flow). For the latter case, the platform's visualisation components relating to the work carried out in T3.2 will also be individually elaborated.

4.1 Authentication & authorisation

Security mechanisms are in place at the Portal to enforce access control, including features for single sign-on, credential authentication and assessment of APIs requests. Specifically, the Portal backend implements the JWT standard to securely create and exchange data between the two Portal applications (front-end and back-end, see Section 3). Password encryption is further employed to store user account passwords as hashed passwords rather than plain text (implemented by means of the *bcrypt* function [3]). Finally, RBAC is enforced through the assignment of distinct roles to system users, and restricting specific platform functionalities (such as propagating an experiment descriptor to the Experiment Coordinator module at the Back-end Layer) only to those roles (*i.e.*, testbed owners) that should have access to them.

The Portal is meant to be used by users that identify themselves as ***experimenters***, *i.e.*, they provide their vertical application components to the platform for the purpose of running a 5G experiment. Each new user is able to register a new account (see also Section 3.2.1.2) to access platform functionalities, such as requesting the delegation of vertical application components, or the execution of an experiment. The platform further supports privileged user accounts (***testbed owners***, or ***admins***), which grant additional rights to their users, such as access to the platform external communication functions**.**

Successfully signing up a new account with the Portal, or signing in using the login API (see also Section 3.2.1.1) invokes the platform's authentication routine, which (if the user's identity is verified) will return a signed JWT containing the user's information with a 30-day expiry date. Every other function in the Portal is then subject to the authorisation routine, which dictates that each request requires an HTTP authorisation request header to be used, so that the user sends their access token to both authenticate them with the backend, and determine their access rights to invoke the requested function (see examples in Sections 3.2.2, 3.2.3, and 3.2.3.8). The token is further used to create and (locally) store a User Model at the frontend client-side storage, where its access rights are verified to display page content (*i.e.*, notifications pulled from the server), as well as granting access rights to frontend functions (*i.e.*, experimenters being able to request execution of a new experiment, whereas testbed owners are able to review and authorise such requests).

It is important to note here that **the Portal cannot be used to create privileged accounts** via its registration API; instead, they are created directly by the 5G-EPICENTRE platform administrators through a special seeding API provisioned to them by the Portal developer (not part of the publicly available documentation). Therefore, administrator privileges are ensured to be outside the reach of third-party experimenter accounts, and RBAC is thus enforced.

4.2 Calibrating underlying infrastructure components

One of the core responsibilities of the Portal is to expose rich and comprehensive user controls to manage how the underlying 5G-EPICENTRE testbed infrastructures shall execute a vertical experiment. Within the overall security architecture of the Portal, such controls are exposed to experimenter accounts, to customise every aspect

of the experiment environment. However, it is the testbed owner that authorises the experiment execution, and the deployment of third-party software on their testbeds' Kubernetes cluster environment.

4.2.1 Delegating the vertical application components

As specified in D1.4, with respect to its Network Application approach, 5G-EPICENTRE adopts the 'Hybrid' option of interaction between the testbed operator and the experimenter (see also [1], and its upcoming v2.0), since in complex PPDR scenarios, often part of the vertical application should be delegated to (and subsequently managed by) the testbed owner/telco operator (for instance, parts of the application needing to be deployed on the edge). This requires the experimenter to delegate part (or parts) of the vertical application to the testbed owner, so that they work in tandem with the platform's Network Applications (see Figure 6).

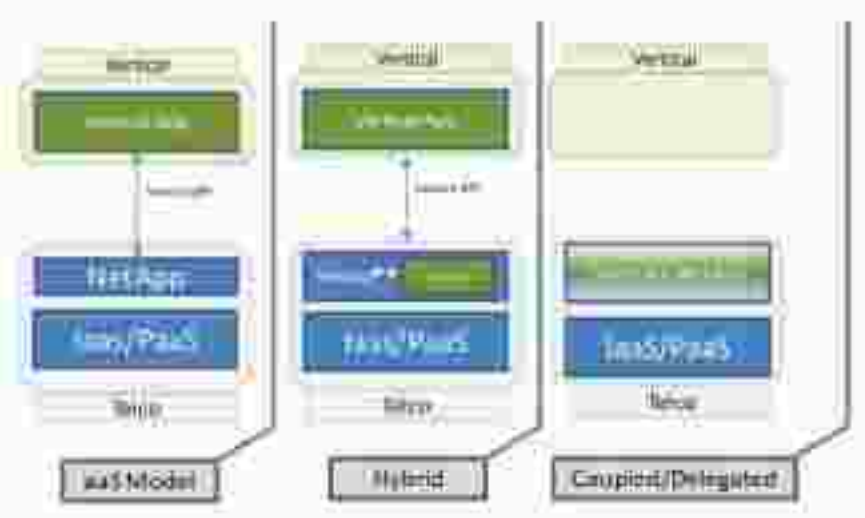

Figure 6: Options of interaction between vertical applications and Network Applications (Image retrieved from [1], licenced under CC BY 4.0).

The Portal supports a GUI workflow for quick and easy vertical application component delegation in the form of Helm chart packages. An experimenter can initiate the procedure by uploading a valid Helm chart package, and providing some metadata for search optimization purposes. A package is temporarily stored on the server backend side, where it is meant to remain until it is either i) deleted by an administrator, or the requesting user; or ii) it is irrevocably accepted or rejected by the testbed owner. In case of acceptance, the package is uploaded onto the Network Service Repository (see D4.3), and becomes "part" of the platform (under the Hybrid option of interaction) for as long as the experimenter/testbed owner allow it to. The chart package is no longer downloadable after it has been delegated to the Repository.

One thing to note here is that all vertical application package delegation requests are forwarded to all testbed owner accounts, so that each can decide whether it will be deployable on top of their own testbed. Any package accepted by a testbed owner account immediately renders that package deployable on their own testbed (only), regardless of the decision made by other testbeds. Thereby, packages could be deployed across infrastructures, provided that the owners of those infrastructures have accepted the delegation request.

It is important to also note that, as is the case with all Portal functions, experimenters have limited access rights to actually place a Helm chart inside the Repository themselves -–only testbed owners retain that authority. This reinforces security of the system, as potentially harmful files can be identified early on and dealt with, without entering the Back-end Layer components. In addition, the whole process of delegating and re-delegating application packages is encapsulated into a single page, streamlining the user experience. A usability evaluation study with indicative end-users will be carried out in the context of T4.5, and will be reported in D4.7 *"Integration, Verification and Testing Report final version"*, due in M34.

4.2.2 Creating an experiment request

Experiment creation involves an experimenter invoking the "wizard" (new experiment page) and following through the process in four simple steps (screens):

- **Scenario selection:** This screen prompts the user to categorise their vertical application under one of the four PPDR experimentation scenarios, as defined in D1.2 *"5G-EPICENTRE experimentation scenarios final version"*, *i.e.*:
    - **5G for video & throughput optimisation**, mapped onto the strengths and features provided by the ***Aveiro*** testbed, operated by 5G-EPICENTRE partner ALB.
    - **5G in drone management environments**, mapped onto the strengths and features provided by the ***Berlin*** testbed, operated by 5G-EPICENTRE partner HHI.
    - **5G slicing and QoS control and management**, mapped onto the strengths and features provided by the ***Málaga*** testbed, operated by 5G-EPICENTRE partner UMA.
    - **Latency management and re-instantiation procedures in 5G**, mapped onto the strengths and features provided by the ***Barcelona*** testbed, operated by 5G-EPICENTRE partner CTTC.

  Scenario selection is a mandatory and irreversible step (once the experiment is requested), as it maps the vertical application experiment execution request to a (one or more) testbed owner(s), who will need to review and authorise that request. It further determines the look and feel of the eventual visualisation environment during the actual execution, since each scenario promotes specific Key Performance Indicator (KPI) calculations that are of particular interest to that scenario, based on the identified commonalities of the project UCs (which provided the blueprints for the aforementioned scenarios), and the agreed upon coordinated and standardised experimentation strategy elaborated in D1.6 *"Experiment evaluation strategy and experimentation plan"*.[15]
- **Experiment information form:** This screen allows the experimenter to provide information on the experiment (*i.e.*, a title and description), alongside scheduling information – when the experiment is desired be deployed. To ensure resource availability, scheduling allows experimenters to designate a time window (by specifying a start and end date for the experiment), in which the Experiment Coordinator's internal Scheduler module should query the testbed for resources availability. If the Coordinator is unable to schedule the experiment between the start date and the end date, it will return an error message and terminate the experiment.
- **Experiment artefacts selection:** In this screen, the experimenter specifies all the artefacts that facilitate the deployment of the vertical application under the specifically desired test conditions, predicating the creation of all necessary Kubernetes objects in the designated testbed cluster. There are two types of deployment artefacts to choose from:
    - **Vertical application artefact(s):** They are the vertical application components that are delegated to the testbed during the experiment execution, following the "Hybrid" option for interaction between the vertical and operator (see also D1.4, or the 2022 5G-PPP Software Network Working Group White paper on *"Network Applications: Opening up 5G and beyond networks"* [1]). They represent vertical application deployment Helm charts that have been previously uploaded onto the Network Service Repository ("the Repository"), starting with a request by the experimenter (see also Section 4.2.1). Selection of (at least) one such artefact is mandatory – an experiment cannot be executed if the vertical application is not specified. The list of available Helm charts in the Repository, as well as the contents of the Helm Charts' Chart.yaml files, are integrated via an ***external repository*** Angular service, which issues requests to the Repository's

[15] For instance, video & throughput scenarios prioritize KPI visualizations related with live video contribution from disaster sites; drone management scenarios prioritize KPIs related to control and operation of drones and other unmanned vehicles; QoS and Slicing scenarios prioritize quality indicators and the demonstration of quality even in challenging network conditions; and Instantiation and latency scenarios prioritize KPI visualization related to deployment metrics and latencies.

OpenAPI Server proxy, in accordance with the documentation in D4.3 *"Curated Network Application image repository"*.

- **Platform and Project Use Case Network Application artefact(s):** These are the Network Applications, network and application functions (NFs/AFs) offered by the 5G-EPICENTRE Consortium to PPDR experimenters. Thereby, third-party experimenters can request the deployment of an instance of an offered application (or use it, if already deployed at the testbed) if they are interested in experimenting with the solution itself, or with their own application. Indicative examples of such software include the three vertical-agnostic ("platform") Network Applications described in D1.4, for exerting control over the underlying network control plane's functions, testbed entities and (security) policies. Through this list of available Network Applications, NFs and AFs (see also D4.2 *"Network functions implementation"*), developers can easily and effortlessly chain their application to a variety of pre-deployed services offered to them, as well as configure the network to fit their needs during the experiment execution: they can prioritise traffic flows and/or guarantee QoS (***Configurator*** Network Application, see also Section 2.2.2); detect and react to outside malicious interference (***Network Intrusion Detection*** Network Application, see also D2.7 *"Cloud-native security intermediate version"*); and obtain custom visual reports of experimenter-specific KPIs directly on the platform Portal (***Analytics Services*** Network Application[16], to be reported in D2.6 *"5G-EPICENTRE Analytics Engine"*, M36). Each Network Application creates additional, optional parameters in the experiment descriptor, declaring specification for deploying additional Helm charts (inside the Network Service Repository) alongside the vertical application one. Platform Network Applications are non-mandatory – the experimenter may opt not to include them in her experimentation environment.

Finally, this screen enables experimenters to indirectly interact with the 5G Traffic Simulation Manager ("the Traffic Simulator", see also D1.4), by selecting a pre-determined traffic profile to launch the experiment with. Traffic profiles are an easy way for experimenters to quickly establish the test simulation conditions under which the network and vertical application execution will be stressed, abstracting the majority of the underlying iPerf[17] framework programmability options – a particularly useful feature for experimenters with little to no knowledge of the iPerf tool. The profiles are defined as follows:

- **Light traffic:** A single iPerf client/server probe pair is initiated, with pre-determined options that aim at simulating typical traffic alongside traffic generated by the vertical application.
- **Crowded network:** A single iPerf client/server probe pair is initiated, with pre-determined options for simulating increased traffic. The pair generates more requests to be transferred across the network, thus creating more realistic conditions for a majority of PPDR events.
- **Disaster:** A single iPerf client/server probe pair is initiated, with pre-determined options that aim at simulating overloaded network conditions. This profile was created to emulate network conditions at the onset of a major catastrophic incident, and is aimed at testing vertical application behaviour in extreme conditions.

At present, the pre-determined options agreed for each profile per testbed, are presented in Table 4.

- **Confirmation:** This screen presents a comprehensive summary of the requested experiment, as dictated by the experimenter's inputs in the prior screens. It also enables the experimenter to review the experiment descriptor (see Table 5 for more details). From this screen, the experimenter invokes the "Submit experiment" API, as detailed in Section 3.2.2.4.

[16] It is worth noting that the Analytics Engine components are by default instantiated in the Kubernetes cluster at any particular testbed, and is therefore not deployed at the request of the vertical. The Vertical can only subscribe and pre-configure the KPI calculation using the Analytic Services Network Application API.

[17] https://iperf.fr/

Table 4: Characterization of traffic simulation parameters/conditions per traffic profile, defined for each of the testbeds

Testbed	Traffic Simulation parameter	Light Traffic	Crowded network	Disaster
UMA	Signal strength	-65 dBm	-80 dBm	-95 dBm
	Background traffic (% of the available bandwidth at the application level)	10%	80%	50%
	Downlink (DL) Modulation	Adaptive	Adaptive	Adaptive
	Uplink (UL) Modulation	16 QAM	16 QAM	16 QAM
	MAC re-transmissions	3	3	5
ALB	Signal strength	-65 dBm	-80 dBm	-95 dBm
	Background traffic (% of the available bandwidth at the application level)	10%	80%	50%
	Downlink (DL) Modulation	Adaptive to 256 QAM	Adaptive to 256 QAM	Adaptive to 256 QAM
	Uplink (UL) Modulation	Adaptive to 64 QAM	Adaptive to 64 QAM	Adaptive to 64 QAM
	MAC re-transmissions	8	8	8
HHI	Signal strength	-55 dBm	-70 dBm	-75 dBm
	Background traffic (% of the available bandwidth at the application level)	10%	80%	50%
	Downlink (DL) Modulation	Adaptive	Adaptive	Adaptive
	Uplink (UL) Modulation	Adaptive max. 64 QAM	Adaptive max. 64 QAM	Adaptive max. 64 QAM
	MAC re-transmissions	5	5	5
CTTC	Signal strength	-65 dBm	-80 dBm	-95 dBm
	Background traffic (% of the available bandwidth at the application level)	10%	80%	50%
	Downlink (DL) Modulation	Adaptive	Adaptive	Adaptive
	Uplink (UL) Modulation	16 QAM	16 QAM	16 QAM
	MAC re-transmissions	3	3	5

With the level of abstraction employed, alongside the sleek design of the Portal's GUI, the process of experiment requesting can be streamlined to a matter of minutes, thus greatly alleviating the risk of third parties disengaging midway through, due to overt complexity of the platform. A usability evaluation study with indicative end-users will be carried out in the context of T4.5, and will be reported in D4.7 *"Integration, Verification and Testing Report final version"*, due in M34.

Table 5: Experiment descriptor used as payload in the POST HTTP request for creating and queueing a new experiment execution in the Experiment Coordinator

Field	Type	Requisitness	Description
Application	String	REQUIRED	This is an unused field within 5G-EPICENTRE and its value should always be set to *null*.
Automated	Boolean	REQUIRED	This value indicates whether the experiment execution should be automatically deployed, or whether the experimenter is expected to manually start the deployment. The default value is *true*.
ExclusiveExecution	Boolean	REQUIRED	This value indicates whether the experiment execution should take place exclusively, or whether the experiment may be executed concurrently with other experiments on the platform. The default value is *true*.
ExperimentType	String	REQUIRED	This is an unused field within 5G-EPICENTRE and its value should always be set to *Standard*.
NSs	Array	REQUIRED	This is an unused field within 5G-EPICENTRE and its value should always be set to *[]*.
Remote	String	REQUIRED	This is an unused field within 5G-EPICENTRE and its value should always be set to *null*.
RemoteDescriptor	String	REQUIRED	This is an unused field within 5G-EPICENTRE and its value should always be set to *null*.
ReservationTime	Array <Number>	REQUIRED	This value incorporates the EPOCH timestamp (in milliseconds) for the desired start of the experiment (start time), along with the EPOCH timestamp (in milliseconds) for the last moment in time where it would be acceptable for the experiment to start (end time).
Scenario	String	REQUIRED	This is an unused field within 5G-EPICENTRE and its value should always be set to *null*.
Slice	String	REQUIRED	This is an unused field within 5G-EPICENTRE and its value should always be set to *null*.
TestCases	Array <String>	REQUIRED	This is a pre-configured field for use within 5G-EPICENTRE and its value should always be set to [*Helm Agent*].

UEs	Array	REQUIRED	This is an unused field within 5G-EPICENTRE and its value should always be set to *[]*.
Version	String	REQUIRED	This value indicates the version of the descriptor. The default value is *"2.1.0"*.
Extra	Object	OPTIONAL	A JSON Object holding parameters corresponding to the iPerf process to be executed, matching the traffic profile/testbed pairing: **Url** [String] REQUIRED Denotes the Unified Resource Locator where the 5G Traffic Simulator Manager instance of the specified testbed is hosted. **ServerProbes** [Array<Object>] REQUIRED Denotes server probe parameters, including origin of the traffic (*e.g.*, "UE", user and experiment identifiers, whether to publish traffic data to the Publisher module – see D1.4 – and iPerf parameters to add to the iPerf Agent request communication[18]). **ClientProbes** [Array<Object>] REQUIRED Denotes client probe parameters, including origin of the traffic (*e.g.*, "UE", user and experiment identifiers, whether to publish traffic data to the Publisher module – see D1.4 – and iPerf parameters to add to the iPerf Agent request communication).
Parameters	Object	REQUIRED	JSON Object holding experiment execution parameters: **Action** [String] REQUIRED Denotes the action that the Experiment Coordinator should perform, *i.e.*, *"deploy"*, or *"delete"*. **Filename** [String] REQUIRED Denotes the filename of the Helm chart in which the vertical application is packaged. **Namespace** [String] REQUIRED Denotes the namespace to execute the experiment. By default, it will be the name of the testbed matching the

[18] The reader is referred to the iPerf online documentation for a comprehensive overview of the iPerf parameters in this field (https://iperf.fr/iperf-doc.php)

			scenario selection, *i.e.*, *"aveiro"*, *"berlin"*, *"malaga"*, or *"barcelona"*.
			Network Application_id [String] REQUIRED A unique identifier for the experiment. It is created automatically for the experiment's document entry into database, and passed to this parameter only after the testbed owner "accepts" the experiment (by default, its value is "*undefined*", making the descriptor invalid).
			HSPF [String] REQUIRED A pseudo-Boolean string value indicating whether the Network Intrusion Detection Network Application is to be deployed in tandem with the vertical application. Its value is either "*yes*" or "*no*".
			HSPF_microservices [Array<String>] REQUIRED An array of microservices names, retrieved from the vertical application's Chart.yaml file, and which the experimenter has selected to inject Network Intrusion Detection sidecars to (see also deliverable D2.7). By default, its value is *[]*.

4.2.3 Authorising an experiment request

Every experiment request created through the aforementioned process is stored in document format inside the database ('experiments' collection), where its 'testbed' parameter renders it visible to testbed owner accounts associated with that testbed (*i.e.*, no other testbed owner accounts may have access to this particular request document). As previously specified, due to RBAC policy, only a testbed owner account can gain access to the experiment reviewing interface.

Reviewing an experiment resembles the review outcome of *i.e.*, scientific manuscripts, in that the reviewer might *accept*, *reject*, or *request revisions* to be made to the experiment, before it is approved. To review an experiment request, the Portal grants the testbed owner access to an incomplete version of the experiment's descriptor file, a comprehensive structure that is exchanged between the Portal and the Experiment Coordinator to create and queue a new experiment execution in one of the testbed infrastructures. Only upon submitting an acceptance decision will the descriptor be completed (and thus made valid as payload for the Experiment Coordinator), and the connection to the Experiment Coordinator is established, via an ***external coordinator*** Angular service, which issues requests to the Coordinator in accordance with the API documentation in D4.5 (to be updated in D2.5 *"5G-EPICENTRE experiment execution"* in M34). The descriptor JSON Object used as payload in the request body is summarised in Table 5.

An example request JSON to the Experiment Coordinator by the 5G-EPICENTRE Portal is shown in Table 6.

Table 6: Example request JSON for the Experiment Coordinator's "/experiment/run" API.

```
{
    "Application": null,
    "Automated": true,
```

```
    "ExclusiveExecution": true,
    "ExperimentType": "Standard",
    "Extra": {
        "Url": "http://X.X.X.X:YYYY/start",
        "ServerProbes": [{
            "origin": "UE",
            "userId": "00",
            "experiment_id": "experiment00",
            "publish": true,
            "request_body": {
                "agent_id": "Probe2A",
                "action": "Start",
                "parameters": {
                    "-s": "",
                    "-u": "",
                    "-p": 6004
                }
            }
        }],
        "ClientProbes": [{
            "origin": "UE",
            "userId": "00",
            "experiment_id": "experiment00",
            "publish": true,
            "request_body": {
                "agent_id": "Probe1B",
                "action": "Start",
                "parameters": {
                    "-c": "Probe2A",
                    "-p": 6004
                    "-u": "",
                    "-b": "150M",
                    "-t": 1200
                    "-i": 1
                }
            }
        }],
    },
    "NSs": [],
    "Parameters": {
        "Action": "deploy",
        "Filename": "mobitrust.zip",
        "Namespace": "malaga",
```

```
        "Network Application_id": "undefined",
        "HSPF": "yes",
        "HSPF_microservices": [
            "mt-monitor",
            "postgresql"
        ]
    }
    "Remote": null,
    "RemoteDescriptor": null,
    "ReservationTime": [1686050700000, 1686140700000],
    "Scenario": null,
    "Slice": null,
    "TestCases": ["Helm Agent"],
    "UEs": [],
    "Version": "2.1.0",
}
```


4.3 Presenting data generated at the testbeds

According to the experiment execution processes which are triggered after an experiment deployment, each 5G-EPICENTRE vertical application deployment (on any testbed) will be accompanied by generation of metrics on a variety of experimental condition indicators, KPIs and detected anomalies. These metrics enter the 5G-EPICENTRE upstream information flow by means of asynchronous messaging communication based on the MQTT protocol and a RabbitMQ message-oriented middleware. The Aggregator component, as stated in D1.4, is responsible for collecting these metrics, and hence generates and forwards messages to the RabbitMQ broker (server), which routes each message to a specific address (queue) identified via the topic exchange routing key pattern. The Portal backend implements an MQTT Client which connects to the broker and subscribes to the Aggregator component's topic exchange, thus making it able to consume messages routed to this queue. The orchestration of this flow is illustrated in Figure 7.

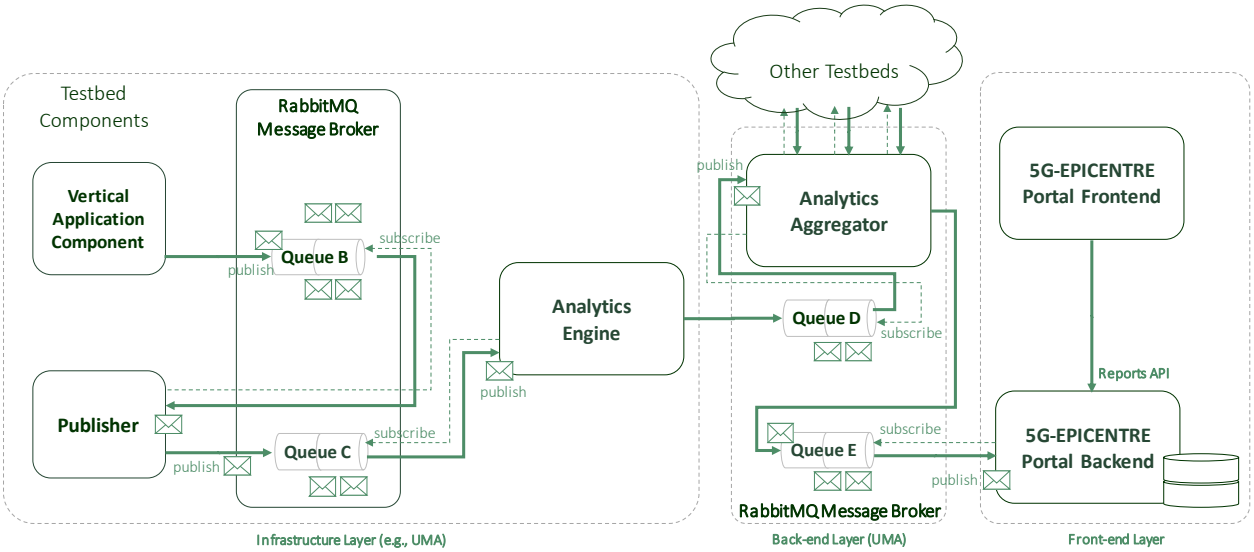

Figure 7: 5G-EPICENTRE Upstream Information Flow orchestration

It is worth noting that, in accordance with the specificities characterising each 5G PPDR scenario listed in D1.2, different metrics and KPIs are prioritized, and shown to experimenters in custom experiment report (functional requirement FR26, see Section 2.1). A different visualization dashboard environment is therefore generated for each experiment scenario, which can be obtained by cross-referencing the messages' *"testbed_id"* payload parameter with the experiment's *"testbed"* parameter. Additional visualization components (widgets) are triggered whenever experiments involve platform Network Application deployments (*e.g.*, the Network Intrusion Detection network application). Finally, through the Analytics services network application, experimenters can opt to include their own KPIs in the analytics pipeline, thus generating uniform visualization structures that pertain to their individualized needs and data.

4.3.1 5G-EPICENTRE backend experiment report generation

Experiment reports refer to Portal-readable documents stored on the database that are constructed at runtime, whenever new messages arrive at the MQTT client. They contain the entirety of insights generated during an experimental run, including experiment KPIs, alongside testbed-generated metrics for monitoring the experiment environment under test and related anomalies.

Each message received contains a JSON payload schema which identifies the experiment for which the metrics are generated (*"experiment_id"*), the testbed where the metrics originate (*"testbed_id"*) and the kind of metrics and calculations that the *"data"* contains by means of the *"type"* parameter field. The information enables the Portal to precisely "match" the message to a particular (accepted and scheduled) experiment execution request, thus creating automatically the **Experiment insights** page for that experiment. The means by which the Portal renders the messages readable at the frontend is by transforming the incoming information into an experiment "report" mongoose document model, which is stored in the *"experiment_reports"* collection in the database.

Whenever a new message is received, the backend queries the *"experiment_reports"* collection for an existing document for which the *"experiment_id"* field matches the one in the message JSON payload. In case a report is not found, a new document is created in the collection, holding distinct data fields for experiment KPIs, 5G network measured parameters (under the specified simulated traffic conditions), and detected anomalies. For every subsequent message, the existing report is updated by adding elements to each of the aforementioned array (depending on the payload *"type"*). In this manner, experiment reports can be accessed both in real-time (a report can be visualized as soon as data enters the *"experiment_reports"* collection), as well as after the experiment has been executed, persisting for as long as the experimenter needs it to. Using the Reports API (see Section 3.2.4), users can gain managerial control over the experiment reports generated, most importantly, being able to access the data and trigger the Portal frontend's data visualization components.

4.3.2 5G-EPICENTRE frontend visualization components

Below is a list of visualization components available to Portal end users. The majority of the below visualization components are developed using the ApexCharts open-source charting library, which (apart from interactive visualization structures) offers out-of-the-box solutions for downloading chart graphics into PNG or SVG image data, as well as exporting measurements in Comma Separated Values (CSV) text files for external processing.

4.3.2.1 Gauge charts

A gauge chart is a data visualization component that resembles a speedometer, and is ideal for pinpointing the value of a KPI at a given point in time. It is particularly useful for estimating the KPI's progress against ranges of pre-determined targets.

In the case of 5G-EPICENTRE KPI calculations, these pertain to the three types of performance defined in the experiment strategy and plan documentation (D1.6): ***upgradeable (U)***, **acceptable (A)**, and ***optimal*** **(O)**. According to D1.6, upgradeable results are defined as those that do not meet experimenters' expectations; whereas acceptable results are those that do. Finally, optimal results are defined as those that exceed expectations.

Using gauge charts, any single KPI value at any point in time can be isolated and visualized to be compared against set criteria for UAO classification, revealing performance insights in a user-friendly and easy-to-interpret manner. Aggregated measurements (*e.g.*, mean values of KPIs across all time points from statistical analysis data triggered at KPI calculation routines of the Analytics Engine) can also be visualized in this manner. Gauge charts are also useful in communicating data related to security measurements taken by the Network Intrusion Detection network application, depicting total and malicious traffic flow detections per second (see also D2.7).

An example of a gauge chart component inside the 5G-EPICENTRE Portal frontend is presented in Figure 8.

Figure 8: Example of gauge chart visualization components (performance within 'Acceptable' range, on the left; and 'Optimal' performance case, on the right)

4.3.2.2 Line charts (time-series graphs)

A line chart is a data visualization component that aims at visualizing KPI values over progressive time periods. It is ideal to depict change of KPI values over the entire course of an experiment.

Within 5G-EPICENTRE spline line charts (*i.e.*, lines are drawn to connect points in smooth curves) are used to depict quantitative variation of KPIs and metrics over pre-determined time intervals (wherein the Aggregator accumulates and publishes data coming from the testbeds). Line charts are automatically generated for every value measurement inside the *"data"* Aggregator message JSON payload parameter, for which a *"timestamp"* value is provided. The *"unit"* and *"param_ID"* parameters are used to combine multiple line charts together in a single visualization widget. Line charts are also useful in communicating data related to security measurements taken by the Network Intrusion Detection network application, e.g., for depicting flow rate of total and abnormal traffic over time (see also D2.7).

Users are able to zoom in and out, single-out a particular time-series, remove time-series from the graph, and view individual measurements for every time point in which the measurement was collected.

An example of a line chart component inside the 5G-EPICENTRE Portal frontend is presented in Figure 9.

4.3.2.3 Boxplots

A boxplot is a data visualization component that depicts distribution and variability, displaying multiple statistical data measures in one single comprehensive structure. They can hence be used to compare distribution of different data sets for one topic, and are an ideal tool for visualizing statistical analysis data.

Within 5G-EPICENTRE, such statistical analysis is carried out by the KPI calculation routines of the Analytics Engine, which propagates to the Portal a summary of the following statistical measures: *minimum*; *first quartile*

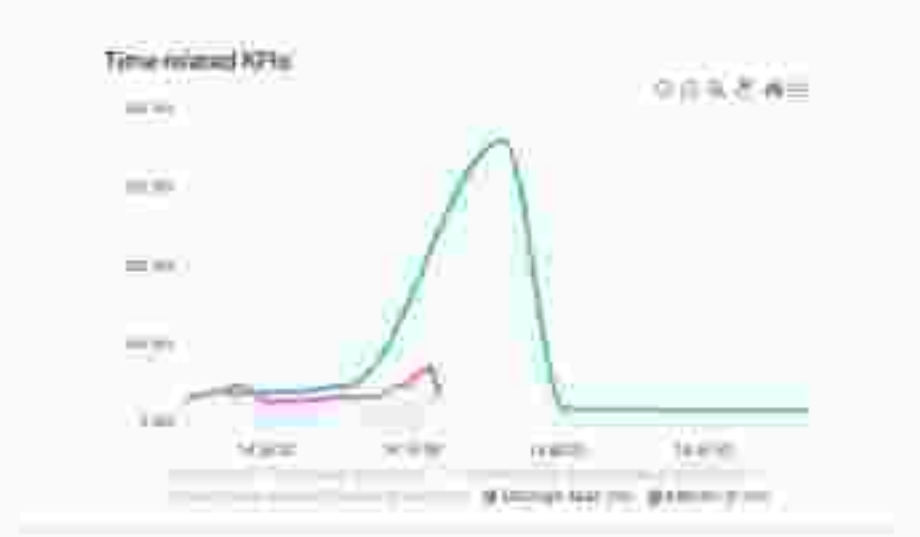

Figure 9: Example of a line chart visualization component

(p25); *median*; *third quartile (p75)*; *maximum*; *mean*; *standard deviation*; *5th percentile (p05)*; and *95th percentile (p95)* values. The first five of these can be used to structure a complete boxplot visualization *i.e.*, a box that represents the spread of data (interquartile range), with "whiskers" extending on both sides towards the minimum and maximum values of the data set, thus enabling experimenters to quickly examine the different data sets for which statistical analysis has been carried out.

An example of a boxplot component inside the 5G-EPICENTRE Portal frontend is presented in Figure 10.

Figure 10: Example of a boxplot visualization component

4.3.2.4 Heat map chart

A heat map chart is a data visualization component that utilizes colors to represent magnitude/intensity of data values within a dataset represented as a 2D grid cell. They are ideal for delivering an immediate visual summary of very large datasets.

A Heat map chart is an exclusive graphical component to the dashboard generated for the Network Intrusion Detection network application. It is used to distinguish among different number of flows (per unit of time), thus representing flows per minute, for which the colours indicate the length of the packets (see also D2.7). An example of heat map chart component inside the 5G-EPICENTRE Portal frontend is presented in Figure 11.

Figure 11: Heatmap visualization component

4.3.2.5 Map chart

The map chart is a visualization component that displays data values over a geographical area provided by the Leaflet library. It is ideal for pinpointing the geographical position of objects.

Within 5G-EPICENTRE, the map chart is used to pinpoint the location of User Equipment (UE), particularly in outdoor experimentation scenarios (*e.g.*, during drone flights). In several cases, KPI values may be accompanied by a latitude and longitude pair (also, altitude), which can be used to mark the location of a UE during the time when a particular KPI measurement was taken. It can be used as a visual reference point to the experiment execution, as well as for granting insight on the impact of distance of the UE from the data center to KPIs, such as latency and round-trip-times.

An example of a map component inside the 5G-EPICENTRE Portal frontend is presented in Figure 12.

Figure 12: Map visualization component

5 Usage

In this Section, usage of the Portal will be discussed as it pertains to its role within the four indicative functional scenarios of the 5G-EPICENTRE platform described in D1.4 (Section 3.7 Information view): 1) **Delegating vertical application components to the testbed operator**; 2) **Scheduling an experiment**; 3) **Experiment deployment**; and 4) **Experiment execution**. In addition, solid plans are provided for the deployment and execution of the project first-party experiments on top of the 5G-EPICENTRE infrastructure via the Portal.

5.1 Usage examples

5.1.1 Delegating vertical application components to the testbed operator

The following list presents a step-by-step walkthrough for an experimenter to delegate their vertical application Helm chart packages to the testbed owners via the platform GUI.

Step 1: The user (Experimenter) visits the Portal space and is redirected to the **Login Screen**, where they use the brackets (in light orange colour) to type in their email used for signing up, and password associated with their account and clicking on the bright orange “Sign in” button. The screen offers the user an option to retrieve a lost or forgotten password (“Forgot password?” button), command the browser to store a cookie for automatic sign-in (“Remember me” option), as well as allowing redirection to the new user registration page, where an account can be requested. The Login Screen can be seen in Figure 13.

Figure 13: 5G-EPICENTRE Portal – Login Screen

Step 2: Upon a successful login attempt, the Portal redirects the user to the main **Dashboard** page. The Dashboard page is different for the different user roles accommodated by the Portal. After logging in, the user gains access to several persistent UI elements, such as the following: i) the navigation pane shown on the left side of the browser window, which enables access to the different services available to each Actor role; ii) a Search function to rapidly find content without navigating to it; iii) a notifications icon for reviewing new items added

to the user's notifications list (*e.g.*, a new experiment request); and iv) the Account details, where the user can manage their account (*e.g.*, switch password). The Dashboard page, along with the described elements, can be seen in Figure 14.

Figure 14: Main Dashboard page (for an Experimenter)

Step 3: A user wishing to delegate a new vertical application artefact should click on "Resources" in the navigation pane. They are then redirected to the **Resources** page, where there are the following UI elements: i) a search bar, where the user can search for the experiments that have been executed; ii) a bright orange button titled "Delegate new" (on the right); and iii) a sorting filter (on the right) (Figure 15).

Figure 15: Resources page

Step 4: After the user clicks on the "Delegate new" button, they are redirected to the **Resource delegate** page, where they provide the necessary metadata (e.g., title, categories), visibility and Helm chart file (Figure 16).

Step 5: The final step requires the Experimenter to opt for an action. The Experimenter has the option to either save their work as a Draft, cancel the entire operation, or "Submit" their delegation request, thus triggering the ordering process (Figure 17).

If publishing the request is successful, the user is redirected to the **Resources** page. Otherwise, the browser will alert on errors. In contrast to Figure 15, this time the Resources page will display the basic information about the delegation request order (*i.e.*, the date the user has chosen for the experiment execution, the title of the experiment, the chosen testbed and the status of the order – see Figure 18).

Figure 16: Resource delegate page – prior to any user interaction

Figure 17: Resource delegate page – confirmation dialogue

Figure 18: Resources page, with new delegation order visible

5.1.2 Scheduling an experiment

The following list presents a step-by-step walkthrough for an experimenter scheduling an experiment.

Step 1: Same as Step 1 from Section 5.1.

Step 2: Same as Step 2 from Section 5.1.

Step 3: A user wishing to create a new experiment should click on "Experiments" in the navigation pane. They are redirected then to the **Experiments** page, where there are the following UI elements: i) a search bar, where the user can search for the experiments that have been executed; ii) a bright orange button titled "Create a new experiment" (on the right); and iii) a sorting filter (on the right) (Figure 19).

Figure 19: Experiments page

Step 4: The user should then click on the "Create a new experiment" button, and will be redirected to the **Scenario** page where they have to select a 5G PPDR scenario from the 4 available options that are provided by the four testbeds:

- Video & Throughput 5G scenario
- Drone Management scenario
- QoS & Slicing scenario
- Instantiation & Latency scenario

The user selects one of the available options, such as the "QoS & Slicing scenario", and clicks on the bright orange "Next step" button (Figure 20).

Step 5: After the user clicks on the "Next step" button, the Experiment Composer presents the various fields required for the definition of the request experiment descriptor in an online form document, through the **Experiment Information** page.

On this page the user has to fill in some basic information about the experiment, such as the title of the experiment, start and end date, whether the execution will be automated or supervised and a description which should be at least 15 characters. On the right side of the screen there is also a calendar that indicates the availability of the selected testbeds.

After the user completes all the required information, they can click on the "Next step" button. They also have the option to go back to the previous step, and save the **Experiment information** page if they wish to continue later or cancel the process (Figure 21).

Step 6: Upon pressing "Next Step", the user is redirected to the **Experiment Artefacts** page. On this page the user can add a Helm chart, pick an available 5G-EPICENTRE Network Application and set the desired traffic simulation parameters.

Step 7: Under "Add deployment artefacts" in the top yellow panel, the user clicks on the orange "Add Helm chart" button. The user selects a Helm chart from the list that appears (mobitrust.zip in Figure 22). Then they click on the bright orange "Add Helm chart" button.

Figure 20: Experiment Composer - Scenario page

Figure 21: Experiment Composer – Experiment Information landing page

Figure 22: Experiment Composer – adding a Helm chart to the experiment descriptor

Step 8: The user has to select one of the 5G-EPICENTRE Network Applications by clicking on the orange "Add service" button.

Step 9: A pop-up window appears on which the user selects one of the Network Applications from the list of options that appears, by clicking on the checkbox to the left of the Network Application name (Network Intrusion and Detection in Figure 23). The user then clicks on the bright orange "Add Portal Service" button (Figure 23).

Figure 23: Experiment Composer – adding a Network Application to the experiment descriptor

Step 10: The user is redirected back to the **Experiment Artefacts** page. There they click on the small orange "expand" icon (in the Network Intrusion and Detection card in Figure 24) to open configuration properties.

Figure 24: Experiment Composer – Opening configuration properties

Step 11: The user can uncheck the "Block the origin IP upon the detection of a malicious flow (Default)" option to demonstrate the capacity to set up a client to interact with a RabbitMQ message broker for handling traffic in custom service (Figure 25).

Figure 25: Experiment Composer – Unchecking "Block the origin IP upon detection of a malicious flow (Default)" option

Step 12: The user clicks on the bright orange "Sidecar management" button (Figure 25). In the dialog that appears the user selects the microservices they want to inject sidecars to. They can click on the checkboxes next to each

microservice name. Alternatively, they can “Select all” or “Deselect all”. They must click on the bright orange “Confirm” button for the selection to be stored (Figure 26).

Figure 26: Experiment Composer – Selecting microservices to monitor

Step 13: The last step on the **Experiment Artefacts** page is to select the desired traffic simulation conditions. Under “Traffic simulation” the user clicks on the “+Add traffic” button.

Step 14: In the dialog box the user can select one of the three available options (light, crowded or disaster) and then clicks on the bright orange “Select profile” button (Figure 27).

Figure 27: Experiment Composer – Selecting traffic simulation conditions

Step 15: After the user has provided all the necessary information in the experiment descriptor, they click on the "Next step" button to proceed (Figure 28).

Figure 28: Experiment Composer – Clicking on the "Next step" button

Step 16: The step requires the Experimenter to review the generated request experiment descriptor. The Experimenter has the option to either revert to a previous step, save their work as a Draft, or cancel the entire operation. On the other hand, they may opt to "Publish" their experiment descriptor, thus triggering the ordering process (Figure 29).

Figure 29: Experiment Composer – Final experiment review and confirmation

If publishing the experiment descriptor is successful, the user is redirected to the **Experiments** page (Figure 19). Otherwise, the browser will alert on errors. The Experiments page will display the basic information about the experiment order (*i.e.*, the date the user has chosen for the experiment execution, the title of the experiment, the chosen testbed and the status of the order).

Figure 30: Experiments page – Displaying key information about the ordered experiment execution

5.1.3 Experiment deployment

The below list presents a step-by-step walkthrough for a testbed owner approving deployment of an experiment.

Step 1: The user (testbed owner) visits the Portal space and is redirected to the **Login Screen**, where they use the brackets (in light orange colour) to type in their email used for signing up, and password associated with their account and clicking on the bright orange "Sign in" button.

Step 2: Upon a successful login attempt, the Portal redirects the user to the main **Dashboard** page, which should notify on the new experiment request (Figure 31).

Figure 31: Experiments page (testbed owner view) – Displaying key information about the ordered experiment execution

Step 3: The user should then navigate to the "Experiment requests" tab in the sidebar and observe the newly added experiment (see Figure 32).

Step 4: By clicking on the experiment Title (the green link inside the experiment card in Figure 32, which for the particular example reads 'MobiTrust'), the user will be redirected to the experiment's page (see Figure 33).

Step 5: The user can click on the descriptor.json button to download and view the descriptor in a text editor. Then, clicking on the bright orange "Review" button opens the reviewing dialogue.

Figure 32: Experiments page (testbed owner view) – Displaying key information about the ordered experiment execution

Figure 33: Experiments page (testbed owner view) – Displaying key information about the ordered experiment execution

Step 6: The user is now able to select a status (“Accept”, “Accept with Revisions”, “Reject”) from the dropdown menu. Comments can be input on the respective field. Clicking on the bright orange “Submit” button will finalize the experiment (Figure 34). After the final step, the Portal should send the experiment descriptor to the Experiment Coordinator. As soon as the Coordinator responds with an execution id (indicating that the experiment has been received and queued for execution), the Portal marks the experiment’s record as ‘executable’, which allows visiting the experiment’s Insights page.

5.1.4 Experiment execution

Usage of the Portal in this scenario implicates that the start date and time for the experiment to be executed have been reached, and that the Experiment Coordinator has successfully deployed all vertical and network application microservices on the testbed Kubernetes cluster. The user can navigate to the experiment’s Insights page, either through a direct button on the experiment’s record, or via the Experiment’s page itself.

Figure 34: Experiments page – Displaying key information about the ordered experiment execution

In accordance to the provisions of Section 4.3, the experiment report is constructed from the values stored inside the mongoose document modelled after the 'Experiment Report' schema (see Section 4.3.1). The user can interact with the various visualization components (see Section 4.3.2), export data in image or CSV file format (useful for generating print documentation of experiment results' insights), or delete a no longer valuable report (which removes the report also from the Mongoose database, according to the Reports API documentation – see Section 3.2.4). Figure 35 and Figure 36 feature indicative examples of an experiment's insights page, showcasing various visualization components together.

5.2 Deployment and execution of the project Use Cases

In this Section, we elaborate on the plans for the deployment and execution of the first-party experiments on top of the 5G-EPICENTRE infrastructure. It delivers on a concrete framework for potentially extending the functionality and capabilities of the platform through the UC partners' revised plans for exposing services to third-parties (whereby a UC might decouple one or more of its "building blocks" into re-usable Network, or Application Functions and Network Applications – the latter as specified in [1]). In such cases, the (sub-)service should be packed in its own Helm chart, exposed via the Network Service Repository, so that it can be chained (*i.e.*, within the UC-owner's own vertical system, or with a generic vertical system envisioned by a third-party). These are expected to become accessible through the 5G-EPICENTRE Portal as deployed services, that are available on the pre-specified testbeds where each UC is deployed. Each shall offer a "how-to" instruction set, that should be available to the 3rd party experimenter via the Portal.

These exposing features will be accommodated in the Portal via the "Platform Network Application" chaining approach described in Section 4.2.2 ("Experiment artefacts selection"), and demonstrated in Section 5.1.2 (via the "Add service" button workflow, see steps 8-12) for the Network Intrusion and Detection service on offer. Hence, it is important to identify UCs with such potential platform extensions, to anticipate and accommodate the UI extensions needed in the Portal (i.e., APIs being exposed to external parties, so that they can use the service alongside their own vertical system/application, and thus, prepare, within the context of integration in WP4, the built-in Portal UIs for each one).

Figure 35: 5G-EPICENTRE Experiments insights page 1/2 (QoS & Slicing specific dashboard)

Figure 36: 5G-EPICENTRE Experiments insights page 2/2 (D2.7 Network Application dashboard integration shown)

These initial plans were collected by all UC-owners and organized into the contents of Table 7. D4.2 shall report on the final catalogue of Network Applications, NFs and AFs that will be available for external experimentation.

Table 7: Plans for the deployment and execution of UC experiments on top of the 5G-EPICENTRE infrastructure

Use Case (UC) No	Status of the deployment and plans for supporting execution of third-party software
UC1	**Will the vertical system be packed into one or more Helm charts:** Yes.
	Has the application been deployed and tested: UC1 has been successfully deployed and tested on UMA and ALB platforms.
	Will the UC expose any services to third-parties, as a deployed component on the pre-specified testbeds: Select UC1 vertical system services will be made available for third parties experimentation on the UMA and ALB platforms**.**
UC2	**Will the vertical system be packed into one or more Helm charts:** Yes.
	Has the application been deployed and tested: UC2 has been deployed and tested with collaborating testbeds. All experimentation data has been shared with the Consortium, whereas a set of basic results were obtained during the first party experimentation phase.
	Will the UC expose any services to third-parties, as a deployed component on the pre-specified testbeds: UC2 plans for Open-Source Contributions, contribution of Application Functions (available for third-party experimentation, but with some constrains), and an Experimentation monitoring & troubleshooting framework. This will allow the visualisation of formatted data, enabling troubleshooting in the early stages of integration, and facilitating the proper tracing of experiments.
UC3	**Will the vertical system be packed into one or more Helm charts:** Yes.
	Has the application been deployed and tested: UC3 is successfully tested and deployed on the HHI testbed. Moreover, synergy has been established between UC3 and UC7, regarding deployment and testing of the UC3 vertical system on top of UC7's hardware components (see also UC7 information, below).
	Will the UC expose any services to third-parties, as a deployed component on the pre-specified testbeds: The UC3 vertical system can be decomposed, and shall offer the 5G-EPICENTRE platform repository one service usable and chainable by third-parties, which forwards the HHI Drone (UE available at HHI testbed) video feed securely. This will offer an out-of-the-box streaming solution to PPDR experimenter, who can then stack up their software on top of it (*e.g.*, a machine learning service, *etc*.).
UC4	**Will the vertical system be packed into one or more Helm charts:** Yes.
	Has the application been deployed and tested: UC4 has been deployed and tested with collaborating testbeds.
	Will the UC expose any services to third-parties, as a deployed component on the pre-specified testbeds: The UC4 vertical system cannot be decomposed. However, ONE will support direct integration with external services brought by third parties.
UC5	**Will the vertical system be packed into one or more Helm charts:** Yes

	Has the application been deployed and tested: The UC has been deployed, tested and validated in both the ALB and UMA testbeds.
	Will the UC expose any services to third-parties, as a deployed component on the pre-specified testbeds: RZ provides an all-in-one service, where both cloud service and vertical application are provisioned. No plans to expose services to third parties are foreseen.
UC6	**Will the vertical system be packed into one or more Helm charts:** Yes
	Has the application been deployed and tested: Deployment is has been tested and validated at the HHI Testbed.
	Will the UC expose any services to third-parties, as a deployed component on the pre-specified testbeds: OPTO can provide parts of its dockerized vertical system (*i.e.*, Analyzer and Proxy), which could be used with a zeroMQ network socket and a Protocol Buffers (protobuf)-formatted definition of variables.
UC7	**Will the vertical system be packed into one or more Helm charts:** Yes
	Has the application been deployed and tested: Deployment of the UC7 vertical system on both the UMA and HHI testbeds.
	Will the UC expose any services to third-parties, as a deployed component on the pre-specified testbeds: While no plans are in place for exposing UC7 software blocks to external parties, YBQ will support third-party experimentation by offering access to their hardware platform (custom smart glasses) via the collaborating testbeds. This UE could then host Android-based vertical applications developed by third-parties, that can chain to the platform components via service APIs (see D1.4 and [1]). The practice has already been adopted by other partners inside the Consortium (ADS, HHI). A comprehensive user and technical manual are being prepared for this.
UC8	**Will the vertical system be packed into one or more Helm charts:** This UC is demonstrative of the capacity of the platform and testbed architectures to host Virtual Machine (VM) based vertical systems Thereby, this UC will not be packaged in a Helm chart.
	Has the application been deployed and tested: The UC8 application has been successfully deployed on both the UMA and CTTC testbeds. Testing has been carried out in both testbeds. Metrics that demonstrate achievement of KPIs can be obtained only where a standalone Windows-based machine is used.
	Will the UC expose any services to third-parties, as a deployed component on the pre-specified testbeds: Because the vertical system in this UC follows a different architectural deployment, and is not planned to be packaged and accessed via the Helm chart driven Network Service Repository, the UC will not expose services to third-parties.

6 Conclusions

In the present document, the entirety of work undergone in the 30 months' duration of 5G-EPICENTRE WP3 was presented. The outputs of this WP deliver on the architectural components of the Front-end Layer, which encapsulates all user-facing applications and systems designed to facilitate a more natural and streamlined interaction between the experimenters and the testbed operators, regarding 5G PPDR experiment execution. All supported functionalities, interfaces and intercommunications with other 5G-EPICENTRE architectural components were elaborated, demystifying how, in the context of the project's third-party experimentation activities (WP5), the external to the project experimenters are meant to utilize the 5G-EPICENTRE platform for their own experiments.

A key aspect in the design of the presented tools and systems refers to their usability and user-friendliness, toward minimizing the risk of users disengaging during experiment on-boarding processes. Several simplifications were introduced since the original designs and mock-ups were presented (D3.3, referring to the original architectural model in D1.3), which are informed by the most recent shifts of the project toward embracing the new Network Application concept and delivery model [1], as well as the project's own amended experimental procedures, definitions and platform updates (D1.4). Therefore, in this document, we have presented both platform (*i.e.*, Portal) and Network Application components (*i.e.*, Configurator), which both intend to support the fundamental objective of Network Applications as *"a middleware layer to simplify the implementation and deployment of vertical systems on a large scale"* [1].

With delivery of this report, activities in WP3 are concluded, in that core functions and module processes will no longer be added to the software that accompanies this report. Any further development activity (*e.g.,* feature and API refinements, code optimization, decorative elements, etc.) shall henceforth be carried out in the context of Task 4.4 "End-to-end platform integration activities", toward lending support to third parties in their deployment of innovative PPDR based solutions for first responders, occurring at Task 5.2 *"PPDR third parties innovation"*, and reported in D5.3 *"Final evaluation report"*.

References

[1] Sayadi, B., Chang, C.-Y., Tranoris, C., Iordache, M., Katsaros, K., Vilalta, R., et al. (2022). *Network Applications: Opening up 5G and beyond networks* [White paper]. Zenodo. https://doi.org/10.5281/zenodo.7123919

[2] ETSI. (2022). System architecture for the 5G System (5GS). *ETSI Technical Specification 123 501 V17.5.0 (2022-7)*. Retrieved from https://www.etsi.org/deliver/etsi_ts/123500_123599/123501/17.05.00_60/ts_123501v170500p.pdf

[3] Provos, N., & Mazieres, D. (1999). A future-adaptable password scheme. In *USENIX Annual Technical Conference, FREENIX Track (Vol. 1999, pp. 81-91)*. Retrieved from https://www.usenix.org/conference/1999-usenix-annual-technical-conference/future-adaptable-password-scheme

Annex I: Portal API documentation

User APIs

Login user

This API is used to authenticate a user in the Portal, using a set of stored credentials.

Table I: API for signing in and authenticating a user

Login user	
Method	POST
Endpoint	/api/users/login
Request Headers	None
Request Body	**email** [String] REQUIRED The user's email address.
	password [String] REQUIRED The user's password.
Response	**200** The authentication was successful. The response will contain the User Model of the user that was authenticated, together with the newly signed token.
	400 The request was invalid. The response will contain an Errors JSON Object with the following error text message: *"Username or password is not valid!"*.

Register new user

This API is used to register a user in the Portal, using a set of provided credentials.

Table II: API for registering a new user

Register new user	
Method	POST
Endpoint	/api/users/register
Request Headers	None
Request Body	**name** [String] REQUIRED The user's name, to be displayed throughout the Portal to address the user.
	email [String] REQUIRED The user's email.

	password [String] REQUIRED The user's password.
	confirmPassword [String] REQUIRED The user's password confirmation. It should match the value of 'password'.
Response	**200** The registration was successful. The response will contain the User Model of the user that was authenticated, together with the newly signed token.
	400 The request was invalid. The response will contain an Errors JSON Object with the following error text message: *"A user with this email address already exists in our database!"*.

Experiments API

Get all

This API is used to retrieve all the experiments that are stored in the *"experiments"* collection inside the database, and which are associated with this user.

Table III: API for retrieving all experiments from the server

Get all	
Method	GET
Endpoint	/api/experiments
Request Headers	**access_token** [String] REQUIRED Signed JWT for the user making the request, containing the user's id, username, email, administrator status (*i.e.*, whether the user is a "testbed owner") and testbed identifier (for testbed owner accounts only). It is automatically injected to the request by the Portal frontend's authentication HTTP interceptor, which intercepts the request and adds this header.
Request Body	None
Response	**200** The query was successful. The response will contain a list of Experiment Model documents matching the query, or an empty list if no matches are found.

Get by search term

This API is used to retrieve all the experiments that are stored in the *"experiments"* collection inside the database, which are associated with this user, and which match a particular given search term.

Table IV: API for retrieving all the experiments from the server, that match a particular search term

Get by search term	
Method	GET

Endpoint	/api/experiments/search/:searchTerm
Request Headers	**access_token** [String] REQUIRED Signed JWT for the user making the request, containing the user's id, username, email, administrator status (*i.e.*, whether the user is a "testbed owner") and testbed identifier (for testbed owner accounts only). It is automatically injected to the request by the Portal frontend's authentication HTTP interceptor, which intercepts the request and adds this header.
Request Body	None
Response	**200** The query was successful. The response will contain a list of Experiment Model documents matching the query, or an empty list if no matches are found.

Get by ID

This API is used to retrieve a single experiment stored in the *"experiments"* collection inside the database, which matches a particular given string identifier.

Table V: API for retrieving an experiment by a string id

Get by ID	
Method	GET
Endpoint	/api/experiments/:experimentId
Request Headers	**access_token** [String] REQUIRED Signed JWT for the user making the request, containing the user's id, username, email, administrator status (*i.e.*, whether the user is a "testbed owner") and testbed identifier (for testbed owner accounts only). It is automatically injected to the request by the Portal frontend's authentication HTTP interceptor, which intercepts the request and adds this header.
Request Body	None
Response	**200** The query was successful. The response will contain the Experiment Model matching the query, or an *undefined* variable, if there is no match.

Submit an experiment request

This API is used to submit a new experiment request to the Portal.

Table VI: API for creating a new experiment in the database

Submit an experiment request	
Method	POST
Endpoint	/api/experiments/create

Request Headers	**access_token** [String] REQUIRED Signed JWT for the user making the request, containing the user's id, username, email, administrator status (*i.e.*, whether the user is a "testbed owner") and testbed identifier (for testbed owner accounts only). It is automatically injected to the request by the Portal frontend's authentication HTTP interceptor, which intercepts the request and adds this header.
Request Body	**id** [String] OPTIONAL Identifier of the experiment. This is generated automatically upon document creation in the database, and, therefore, is not used in this request (although the field is required in the interface that is used to construct the request body).
	title [String] REQUIRED Title of the experiment, as specified by the user during the experiment creation process (see Section 4.2).
	owner [String] REQUIRED The (user)name of the account that created the experiment.
	filename [String] REQUIRED The filename of the vertical application's Helm chart in the Network Service Repository, as selected by the user during the experiment creation process (see Section 4.2).
	Network Applications [Array<String>] OPTIONAL A list of names of the 5G-EPICENTRE platform Network Applications (see deliverables D1.4 and D4.2 for more details) to deploy alongside the vertical application's Helm chart, as specified by the user during the experiment creation process (see Section 4.2).
	traffic [String] OPTIONAL The name of the traffic simulation profile for the requested experiment, as specified by the user during the experiment creation process (see Section 4.2).
	start_date_ts [Number] REQUIRED Timestamp of the designated start point for trying experiment execution, as specified by the user during the experiment creation process (see Section 4.2). This value effectively tells the Experiment Coordinator when to start querying for resources to deploy the experiment.
	end_date_ts [Number] REQUIRED Timestamp of the designated end point for trying experiment execution, as specified by the user during the experiment creation process (see Section 4.2). This value effectively tells the Experiment Coordinator when to stop the querying for resources to deploy the experiment.

	automated [Boolean] REQUIRED A Boolean flag indicating whether the experiment will be automatically deployed by the Experiment Coordinator, as specified by the user during the experiment creation process (see Section 4.2).
	description [String] REQUIRED Textual description of the experiment, as specified by the user during the experiment creation process (see Section 4.2).
	testbed [String] REQUIRED Denotes which testbed is assigned the experiment based on the PPDR scenario that the user specified during the experiment creation process (see Section 4.2). Allowed values are “Aveiro”, “Berlin”, “Malaga”, and “Barcelona”.
	status [String] REQUIRED Denotes the status of the experiment. Allowed values are “Draft”, “Submitted”, “Accepted”, “Rejected”, “Cancelled”, and “Accepted with revisions”. It is created automatically by the Portal frontend during the experiment creation process (see Section 4.2).
	history [Array<Object>] OPTIONAL A list of JSON Objects describing the record of submissions for this experiment. Each record stores the date in which it was created, the last date in which it was updated, textual comments associated with the review of the testbed owner for the particular submission, and the status of the submission following the review. It is created automatically by the Portal frontend during the experiment creation process (see Section 4.2).
	descriptor [Object] REQUIRED A JSON Object matching the experiment descriptor, which is the payload sent to the Experiment Coordinator when finalising the deployment request. It is created automatically by the Portal frontend during the experiment creation process (see Section 4.2).
Response	**200** The experiment creation was successful. The response will contain the Experiment Model of the experiment that was created.
	400 The request was invalid. The response will contain an Errors JSON Object with the following error text message: ‘*Experiment exists!’*.

Update an experiment request

This API is used to submit a revision to an existing experiment request created using the Portal.

Table VII: API for updating an experiment in the database

Update an experiment request

Method	POST
Endpoint	/api/experiments/revise
Request Headers	**access_token** [String] REQUIRED Signed JWT for the user making the request, containing the user's id, username, email, administrator status (*i.e.*, whether the user is a "testbed owner") and testbed identifier (for testbed owner accounts only). It is automatically injected to the request by the Portal frontend's authentication HTTP interceptor, which intercepts the request and adds this header.
Request Body	**id** [String] REQUIRED Identifier of the experiment.
	filename [String] OPTIONAL The filename of the vertical application's Helm chart in the Network Service Repository, as selected by the user during the experiment creation process (see Section 4.2).
	Network Application [Array<String>] OPTIONAL A list of names of the 5G-EPICENTRE platform Network Applications (see deliverables D1.4 and D4.2 for more details) to deploy alongside the vertical application's Helm chart, as specified by the user during the experiment creation process (see Section 4.2).
	traffic [String] OPTIONAL The name of the traffic simulation profile for the requested experiment, as specified by the user during the experiment creation process (see Section 4.2).
	start_date [Number] OPTIONAL Timestamp of the designated start point for trying experiment execution, as specified by the user during the experiment creation process (see Section 4.2). This value effectively tells the Experiment Coordinator when to start querying for resources to deploy the experiment.
	end_date [Number] OPTIONAL Timestamp of the designated end point for trying experiment execution, as specified by the user during the experiment creation process (see Section 4.2). This value effectively tells the Experiment Coordinator when to stop the querying for resources to deploy the experiment.
	automated [Boolean] OPTIONAL A Boolean flag indicating whether the experiment will be automatically deployed by the Experiment Coordinator, as specified by the user during the experiment creation process (see Section 4.2).

	description **[String] OPTIONAL** **Textual description of the experiment, as specified by the user during the experiment creation process (see Section 4.2).**
	status [String] REQUIRED Denotes the status of the experiment. Allowed values are "Draft", "Submitted", "Accepted", "Rejected", "Cancelled", and "Accepted with revisions". It is created automatically by the Portal frontend during the experiment creation process (see Section 4.2).
	lastRecord [Object] REQUIRED A JSON Object describing the last record of submissions for this experiment. Each record stores textual comments associated with the review of the testbed owner for the particular submission, and the status of the submission following the review. Upon being stored in the database, it automatically creates parameters for the date in which it was created, and the last date in which it was updated. It is created automatically by the Portal frontend during the experiment creation process (see Section 4.2).
	pushRecord [Boolean] REQUIRED A Boolean value that indicates whether the *'lastRecord'* Object should be pushed to the array of submissions kept in the experiment's *'history'* parameter (if true), or whether to update the values of the existing last record in that parameter (if false).
	descriptor [Object] OPTIONAL A JSON Object matching the experiment descriptor, which is the payload sent to the Experiment Coordinator when finalizing the deployment request. It is created automatically by the Portal frontend during the experiment creation process (see Section 4.2).
Response	**200** The experiment update was successful. The response will contain a JSON Object summarising the results of the mongoose *updateOne*() function.
	400 The request was invalid. The response will contain an Errors JSON Object with the following error text message: '*Experiment does not exist!'*.

Remove an experiment request

This API is used to remove/delete an existing experiment request created using the Portal.

Table VIII: API for removing an experiment from the database

Remove an experiment request	
Method	POST
Endpoint	/api/experiments/remove
Request Headers	**access_token** [String] REQUIRED

	Signed JWT for the user making the request, containing the user's id, username, email, administrator status (*i.e.*, whether the user is a "testbed owner") and testbed identifier (for testbed owner accounts only). It is automatically injected to the request by the Portal frontend's authentication HTTP interceptor, which intercepts the request and adds this header.
Request Body	**id** [String] REQUIRED Identifier of the experiment.
Response	**200** The experiment deletion was successful. The response will contain the Experiment Model of the experiment that was deleted.
	400 The request was invalid. The response will contain an Errors JSON Object with the following error text message: '*Experiment does not exist!'*.

Resources API

Get all

This API is used to retrieve all the vertical application artefact delegation requests that are stored in the *"helm-chart-artefacts"* collection inside the database, and which are associated with this user.

Table IX: API for retrieving all resources from the server

Get all	
Method	GET
Endpoint	/api/resources
Request Headers	**access_token** [String] REQUIRED Signed JWT for the user making the request, containing the user's id, username, email, administrator status (*i.e.*, whether the user is a "testbed owner") and testbed identifier (for testbed owner accounts only). It is automatically injected to the request by the Portal frontend's authentication HTTP interceptor, which intercepts the request and adds this header.
Request Body	None
Response	**200** The query was successful. The response will contain a list of Resource Model documents matching the query, or an empty list if no matches are found.

Get by search term

This API is used to retrieve all the vertical application artefact delegation requests that are stored in the *"helm-chart-artefacts"* collection inside the database, which are associated with this user, and which match a particular given search term.

Table X: API for retrieving resources from the server, that match a particular search term

Get by search term	
Method	GET
Endpoint	/api/resources/search/:searchTerm
Request Headers	**access_token** [String] REQUIRED Signed JWT for the user making the request, containing the user's id, username, email, administrator status (*i.e.*, whether the user is a "testbed owner") and testbed identifier (for testbed owner accounts only). It is automatically injected to the request by the Portal frontend's authentication HTTP interceptor, which intercepts the request and adds this header.
Request Body	None
Response	**200** The query was successful. The response will contain a list of Resource Model documents matching the query, or an empty list if no matches are found.

Get by ID

This API is used to retrieve a single vertical application artefact delegation request stored in the *"helmchart-artefacts"* collection inside the database, which matches a particular given string identifier.

Table XI: API for retrieving a resource by a string id

Get by ID	
Method	GET
Endpoint	/api/resources/:resourceId
Request Headers	**access_token** [String] REQUIRED Signed JWT for the user making the request, containing the user's id, username, email, administrator status (*i.e.*, whether the user is a "testbed owner") and testbed identifier (for testbed owner accounts only). It is automatically injected to the request by the Portal frontend's authentication HTTP interceptor, which intercepts the request and adds this header.
Request Body	None
Response	**200** The query was successful. The response will contain the Resource Model matching the query, or an *undefined* variable, if there is no match.

Get by tag name

This API is used to retrieve all the resources that are stored in the *"helmchart-artefacts"* collection inside the database, which are associated with this user, and for which the user-defined *"category"* parameter includes elements that match a particular given tag name.

Table XII: API for retrieving all the resources from the server, that match a particular tag name

Get by tag name	
Method	GET
Endpoint	/api/resources/tag/:tagName
Request Headers	**access_token** [String] REQUIRED Signed JWT for the user making the request, containing the user's id, username, email, administrator status (*i.e.*, whether the user is a "testbed owner") and testbed identifier (for testbed owner accounts only). It is automatically injected to the request by the Portal frontend's authentication HTTP interceptor, which intercepts the request and adds this header.
Request Body	None
Response	**200** The query was successful. The response will contain a list of Resource Model documents matching the query, or an empty list if no matches are found.

Submit a vertical application artefact delegation request

This API is used to submit a new vertical application artefact delegation request to the Portal.

Table XIII: API for creating a new vertical application artefact delegation request in the database

Submit a vertical application artefact delegation request	
Method	POST
Endpoint	/api/resources/create
Request Headers	**access_token** [String] REQUIRED Signed JWT for the user making the request, containing the user's id, username, email, administrator status (*i.e.*, whether the user is a "testbed owner") and testbed identifier (for testbed owner accounts only). It is automatically injected to the request by the Portal frontend's authentication HTTP interceptor, which intercepts the request and adds this header.
Request Body	**id** [String] OPTIONAL Identifier of the vertical application artefact delegation request. This is generated automatically upon document creation in the database, and, therefore, is not used in this request (although the field is required in the interface that is used to construct the request body).
	title [String] REQUIRED Title of the vertical application artefact delegation request, as specified by the user during the resource creation process (see Section 4.2).

	owner [String] REQUIRED The (user)name of the account that created the vertical application artefact delegation request.
	filename [String] REQUIRED The filename of the vertical application's Helm chart, as uploaded by the user during the resource creation process (see Section 4.2).
	category [Array<String>] OPTIONAL A list of free keywords/tags that can be used to describe the vertical application artefact delegation request (for search optimization purposes), as specified by the user during the resource creation process (see Section 4.2).
	status [String] REQUIRED Denotes the status of the vertical application artefact delegation request. Allowed values are "Draft", "Submitted", "Accepted", "Rejected", "Cancelled", and "Accepted with revisions". It is created automatically by the Portal frontend during the resource creation process (see Section 4.2).
	history [Array<Object>] OPTIONAL A list of JSON Objects describing the record of submissions for this vertical application artefact delegation request. Each record stores the date in which it was created, the last date in which it was updated, textual comments associated with the review of the testbed owner for the particular submission, and the status of the submission following the review. It is created automatically by the Portal frontend during the resource creation process (see Section 4.2).
	public [Boolean] REQUIRED A Boolean indicator for whether the artefact should be made public (*i.e.*, freely and openly accessible to others), or private (*i.e.*, only accessible to the experimenter making the upload, and the testbed owners). It is explicitly specified by the user during the resource creation process (see Section 4.2).
Response	**200** The vertical application artefact delegation request creation was successful. The response will contain the HelmChart Model of the resource that was created.
	400 The vertical application artefact delegation request was invalid. The response will contain an Errors JSON Object with the following error text message: '*Resource exists!'*.

Update a resource request

This API is used to submit a revision to an existing vertical application artefact delegation request created using the Portal.

Table XIV: API for updating a vertical application artefact delegation request in the database

Update a vertical application artefact delegation request

Method	POST
Endpoint	/api/resources/revise
Request Headers	**access_token** [String] REQUIRED Signed JWT for the user making the request, containing the user's id, username, email, administrator status (*i.e.*, whether the user is a "testbed owner") and testbed identifier (for testbed owner accounts only). It is automatically injected to the request by the Portal frontend's authentication HTTP interceptor, which intercepts the request and adds this header.
Request Body	**id** [String] REQUIRED Identifier of the vertical application artefact delegation request.
	filename [String] OPTIONAL The filename of the vertical application's Helm chart, as uploaded by the user during the resource creation process (see Section 4.2).
	category [Array<String>] OPTIONAL A list of free keywords/tags that can be used to describe the vertical application artefact delegation request (for search optimization purposes), as specified by the user during the resource creation process (see Section 4.2).
	testbed [Array<String>] OPTIONAL When the response is created by a testbed owner, regarding a review of the vertical application artefact delegation request, this parameter denotes a list of testbeds that have validated the request, and for which the deployment of the artefact has been verified. Each testbed owner can only push their own testbed's value in the array, which should contain at the most 4 values (corresponding to the four testbeds).
	status [String] REQUIRED Denotes the status of the vertical application artefact delegation request. Allowed values are "Draft", "Submitted", "Accepted", "Rejected", "Cancelled", and "Accepted with revisions". It is created automatically by the Portal frontend during the resource creation process (see Section 4.2).
	lastRecord [Object] REQUIRED A JSON Object describing the last record of submissions for this vertical application artefact delegation request. Each record stores textual comments associated with the review of the testbed owner for the particular submission, and the status of the submission following the review. Upon being stored in the database, it automatically creates parameters for the date in which it was created, and the last date in which it was updated. It is created automatically by the Portal frontend during the vertical application artefact delegation process (see Section 4.2).

	pushRecord [Boolean] REQUIRED A Boolean value that indicates whether the *'lastRecord'* Object should be pushed to the array of submissions kept in the artefact's *'history'* parameter (if true), or whether to update the values of the existing last record in that parameter (if false).
Response	**200** The resource update was successful. The response will contain a JSON Object summarising the results of the mongoose *updateOne*() function.
	400 The request was invalid. The response will contain an Errors JSON Object with the following error text message: *'Resource does not exist!'*.

Remove a resource request

This API is used to remove/delete an existing vertical application artefact delegation request created using the Portal.

Table XV: API for removing a vertical application artefact delegation request from the database

Remove a vertical application artefact delegation request	
Method	POST
Endpoint	/api/resources/remove
Request Headers	**access_token** [String] REQUIRED Signed JWT for the user making the request, containing the user's id, username, email, administrator status (*i.e.*, whether the user is a "testbed owner") and testbed identifier (for testbed owner accounts only). It is automatically injected to the request by the Portal frontend's authentication HTTP interceptor, which intercepts the request and adds this header.
Request Body	**id** [String] REQUIRED Identifier of the vertical application artefact delegation request.
Response	**200** The resource deletion was successful. The response will contain a JSON Object summarising the results of the mongoose *deleteOne*() function.
	400 The request was invalid. The response will contain an Errors JSON Object with the following error text message: '*Resource does not exist!'*.

Upload temporary chart package

This API is used to upload a file to a temporary storage in the Portal backend *(/uploads* folder).

Table XVI: API for uploading a file to the backend temporary storage folder.

Upload temporary chart package	
Method	POST

Endpoint	/api/resources/upload
Request Headers	**access_token** [String] REQUIRED Signed JWT for the user making the request, containing the user's id, username, email, administrator status (*i.e.*, whether the user is a "testbed owner") and testbed identifier (for testbed owner accounts only). It is automatically injected to the request by the Portal frontend's authentication HTTP interceptor, which intercepts the request and adds this header.
Request Body	**payload** [FormData] REQUIRED The request payload is formatted in accordance to the XMLHttpRequest FormData interface, where a new custom key (*"thumbnail"*) has been appended holding the file as its value.
Response	**200** The query was successful. The response will contain the full path to the /uploads folder, wherein the file has been stored.
	422 The query content is unprocessable. The response will contain an Errors JSON Object with the following error text message: '*Uploaded file is not a valid Helm chart*'.
	500 Internal server error - the operation that sends the file from the temporary path to the path specified as its parameter has failed. The response will contain an Errors JSON Object with the related error text message.

Download temporary chart package

This API is used to download a file from the temporary storage in the Portal backend *(/uploads* folder).

The server retrieves the filename from the request parameters and re-creates the local path string where the file is located. It then proceeds to transfer the file at the given path as an attachment to the response.

Table XVII: API for downloading a file from the backend temporary storage folder.

Download temporary chart package	
Method	GET
Endpoint	/api/resources/download/:filename
Request Headers	**access_token** [String] REQUIRED Signed JWT for the user making the request, containing the user's id, username, email, administrator status (*i.e.*, whether the user is a "testbed owner") and testbed identifier (for testbed owner accounts only). It is automatically injected to the request by the Portal frontend's authentication HTTP interceptor, which intercepts the request and adds this header.
Request Body	None

Response	**200** The query was successful. The response will contain the full path to the /uploads folder, wherein the file has been stored.
	400 The request was invalid. The response will contain an Errors JSON Object with the following error text message: '*File does not exist!*'.

Delete temporary chart package

This API is used to delete a file from the temporary storage in the Portal backend *(/uploads* folder).

Table XVIII: API for deleting a file from the backend temporary storage folder.

Download temporary chart package	
Method	DELETE
Endpoint	/api/resources/:filename
Request Headers	**access_token** [String] REQUIRED Signed JWT for the user making the request, containing the user's id, username, email, administrator status (*i.e.*, whether the user is a "testbed owner") and testbed identifier (for testbed owner accounts only). It is automatically injected to the request by the Portal frontend's authentication HTTP interceptor, which intercepts the request and adds this header.
Request Body	None
Response	**200** The query was successful. The response will contain the full path to the /uploads folder, wherein the file has been stored.
	400 The request was invalid. The response will contain an Errors JSON Object with the following error text message: '*File does not exist!*'.
	500 Internal server error. The response will contain an Errors JSON Object with the following error text message: '*File could not be deleted!*'.

Reports API

Get by ID

This API is used to retrieve a single experiment report stored in the *"experiment-reports"* collection inside the database, which matches a potential set of given string identifiers.

Table XIX: API for retrieving an experiment report by a set of string ids

Get by ID	
Method	GET
Endpoint	/api/reports/:experimentId/:executionId

Request Headers	**access_token** [String] REQUIRED Signed JWT for the user making the request, containing the user's id, username, email, administrator status (*i.e.*, whether the user is a "testbed owner") and testbed identifier (for testbed owner accounts only). It is automatically injected to the request by the Portal frontend's authentication HTTP interceptor, which intercepts the request and adds this header.
Request Body	None
Response	**200** The query was successful. The response will contain the Resource Model matching the query, or an *undefined* variable, if there is no match.
	400 The request was invalid. The response will contain an Errors JSON Object with the following error text message: '*Requested experiment report does not exist!*'.

Delete report

This API is used to remove/delete an existing experiment report from the *"experiment-reports"* collection inside the database.

Table XX: API for removing an experiment report by supplying it a set of string ids

Delete reports	
Method	DELETE
Endpoint	/api/reports/:experimentId/:executionId
Request Headers	**access_token** [String] REQUIRED Signed JWT for the user making the request, containing the user's id, username, email, administrator status (*i.e.*, whether the user is a "testbed owner") and testbed identifier (for testbed owner accounts only). It is automatically injected to the request by the Portal frontend's authentication HTTP interceptor, which intercepts the request and adds this header.
Request Body	None
Response	**200** The query was successful. The response will contain the Resource Model matching the query, or an *undefined* variable, if there is no match.
	400 The request was invalid. The response will contain an Errors JSON Object with the following error text message: '*Requested experiment report does not exist!!*'.

Notifications API

Get all

This API is used to retrieve all the notifications addressed to this user, that are stored in the 'notifications' collection inside the database.

Table XXI: API for retrieving all notifications from the server

Get all	
Method	GET
Endpoint	/api/notifications
Request Headers	**access_token** [String] REQUIRED Signed JWT for the user making the request, containing the user's id, username, email, administrator status (*i.e.*, whether the user is a "testbed owner") and testbed identifier (for testbed owner accounts only). It is automatically injected to the request by the Portal frontend's authentication HTTP interceptor, which intercepts the request and adds this header.
Request Body	None
Response	**200** The query was successful. The response will contain a list of Notification Model documents matching the query, or an empty list if no matches are found.

Get all unread

This API is used to retrieve all the unread notifications addressed to this user, and which are stored in the 'notifications' collection inside the database.

Table XXII: API for retrieving all unread notifications from the server

Get by search term	
Method	GET
Endpoint	/api/notifications/unread
Request Headers	**access_token** [String] REQUIRED Signed JWT for the user making the request, containing the user's id, username, email, administrator status (*i.e.*, whether the user is a "testbed owner") and testbed identifier (for testbed owner accounts only). It is automatically injected to the request by the Portal frontend's authentication HTTP interceptor, which intercepts the request and adds this header.
Request Body	None
Response	**200** The query was successful. The response will contain a list of Notification Model documents matching the query, or an empty list if no matches are found.

Get by search term

This API is used to retrieve all the notifications addressed to this user, that are stored in the 'notifications' collection inside the database, and which match a particular given search term.

Table XXIII: API for retrieving all notifications from the server, that match a particular search term

Get by search term	
Method	GET
Endpoint	/api/notifications/search/:searchTerm
Request Headers	**access_token** [String] REQUIRED Signed JWT for the user making the request, containing the user's id, username, email, administrator status (*i.e.*, whether the user is a "testbed owner") and testbed identifier (for testbed owner accounts only). It is automatically injected to the request by the Portal frontend's authentication HTTP interceptor, which intercepts the request and adds this header.
Request Body	None
Response	**200** The query was successful. The response will contain a list of Notification Model documents matching the query, or an empty list if no matches are found.

Send notification

This API is used to create a new notification in the database, making it available to the correspondents that should be alerted to it.

Table XXIV: API for creating a new notification in the database.

Submit a notification to the server	
Method	POST
Endpoint	/api/notifications/create
Request Headers	**access_token** [String] REQUIRED Signed JWT for the user making the request, containing the user's id, username, email, administrator status (*i.e.*, whether the user is a "testbed owner") and testbed identifier (for testbed owner accounts only). It is automatically injected to the request by the Portal frontend's authentication HTTP interceptor, which intercepts the request and adds this header.
Request Body	**id** [String] OPTIONAL Identifier of the notification. This is generated automatically upon document creation in the database, and, therefore, is not used in this request (although the field is required in the interface that is used to construct the request body).
	text [String] REQUIRED Text of the notification. It is created automatically by the Portal frontend during the experiment creation process (see Section 4.2).

	testbed [String] REQUIRED Denotes which testbed is assigned the experiment based on the PPDR scenario that the user specified during the experiment creation process (see Section 4.2). Allowed values are “Aveiro”, “Berlin”, “Malaga”, “Barcelona” and “None”. It is used to associate the notification with every testbed owner account that manages a particular testbed (“None” notifications are assigned to all testbed owners).
	username [String] REQUIRED The username of the account owner, whose actions have triggered the creation of the notification. This field is automatically filled in by the Portal frontend during the experiment creation process (see Section 4.2).
	correspondent [String] OPTIONAL The username of the account owner, who should be notified to the creation of the notification. This field is automatically filled in by the Portal frontend during the experiment creation process (see Section 4.2).
	experimentId [String] REQUIRED The ID of the experiment or vertical application delegation request for which this notification is created. This field is automatically filled in by the Portal frontend during the experiment creation process (see Section 4.2).
Response	**200** The notification creation was successful. The response will contain the Notification Model of the notification that was created.

Update all notifications

This API is used to mark all notifications addressed to this user as “read”, from their point of view.

Table XXV: API for marking all notifications in this user’s inbox as “read”

Update all notifications	
Method	GET
Endpoint	/api/notifications/update/all
Request Headers	**access_token** [String] REQUIRED Signed JWT for the user making the request, containing the user’s id, username, email, administrator status (*i.e.*, whether the user is a “testbed owner”) and testbed identifier (for testbed owner accounts only). It is automatically injected to the request by the Portal frontend’s authentication HTTP interceptor, which intercepts the request and adds this header.
Request Body	None
Response	**200** The query was successful. The response will contain a JSON Object summarising the results of the mongoose *updateMany*() function.

	400 The request was invalid. The response will contain an Errors JSON Object with the following error text message: *"There are no notifications for this user!"*.

Update notification

This API is used to mark a particular notification accessed by this user as "read", from their point of view.

Table XXVI: API for marking a specified notification in this user's inbox as 'read'

Update a notification by id	
Method	GET
Endpoint	/api/notifications/update
Request Headers	**access_token** [String] REQUIRED Signed JWT for the user making the request, containing the user's id, username, email, administrator status (*i.e.*, whether the user is a "testbed owner") and testbed identifier (for testbed owner accounts only). It is automatically injected to the request by the Portal frontend's authentication HTTP interceptor, which intercepts the request and adds this header.
Request Body	**id** [String] OPTIONAL Identifier of the notification.
Response	**200** The query was successful. The response will contain a JSON Object summarising the results of the mongoose *updateOne*() function.
	400 The request was invalid. The response will contain an Errors JSON Object with the following error text message: '*Notification does not exist!*'.

Mark for removal/Remove a notification

This API is used to remove/delete a notification created using the Portal. The action will result in the omission of this notification from all GET operations that this user will request in the future. For a notification to be completely removed from the database, both correspondents (testbed owner and experimenter) must have marked the same notification for permanent removal.

Table XXVII: API for flagging a notification for removal from the database.

Remove a notification from the server	
Method	POST
Endpoint	/api/ notifications/remove
Request Headers	**access_token** [String] REQUIRED Signed JWT for the user making the request, containing the user's id, username, email, administrator status (*i.e.*, whether the user is a 'testbed owner') and testbed identifier (for testbed owner accounts only). It is automatically injected to the request by the

	Portal frontend's authentication HTTP interceptor, which intercepts the request and adds this header.
Request Body	**id** [String] REQUIRED Identifier of the notification.
Response	**200** The experiment deletion was successful. The response will contain the Experiment Model of the experiment that was deleted.
	400 The request was invalid. The response will contain an Errors JSON Object with the following error text message: *"Notification does not exist!"*.