

HORIZONS 2020 PROGRAMME

Research and Innovation Action – FIRE Initiative

Call Identifier:	H2020-ICT-2014-1
Project Number:	643943
Project Acronym:	FIESTA-IoT
Project Title:	Federated Interoperable Semantic IoT/cloud Testbeds and Applications

Infrastructure for Submitting and Managing IoT Experiments – V2

Document Id:	FIESTA-IoT-D4.8-20171130-Draft
File Name:	FIESTA-IoT-D4.8-20171130-Draft.pdf
Document reference:	Deliverable 4.8
Version:	Draft
Editor:	Rachit Agarwal/Nikolaos Georgantas/Valerie Issarny
Organisation:	Inria
Date:	30 / 11 / 2017
Document type:	R, DEM
Dissemination level:	PU

Copyright © 2017 National University of Ireland - NUIG / Coordinator (Ireland), University of Southampton IT Innovation - ITINNOV (United Kingdom), Institut National de Recherche en Informatique & Automatique - INRIA, (France), University of Surrey - UNIS (United Kingdom), Unparallel Innovation, Lda - Unparallel (Portugal), Easy Global Market - EGM (France), NEC Europe Ltd. NEC (United Kingdom), University of Cantabria UNICAN (Spain), Fraunhofer-FOKUS (Germany), Research and Education Laboratory in Information Technologies - Athens Information Technology - AIT (Greece), Sociedad para el desarrollo de Cantabria – SODERCAN (Spain), Ayuntamiento de Santander – SDR (Spain), Korea Electronics Technology Institute KETI, (Korea). The European Commission within HORIZON 2020 Program funds the FIESTA-IoT project.

PROPRIETARY RIGHTS STATEMENT

This document contains information, which is proprietary to the FIESTA-IoT Consortium.
Neither this document nor the information contained herein shall be used, duplicated or communicated by any means to any third party, in whole or in parts, except with prior written consent of the consortium.

DOCUMENT HISTORY

Rev.	Author(s)	Organisation(s)	Date	Comments
V01	Rachit Agarwal	Inria	2017/04/18 2017/09/04 2017/09/20 2017/09/20 2017/10/24	Initial version of the Document Updated TOC Section: Relation with the Functional Architecture, Starting Service with EEE, Experiment Deployment Services, Execute Experiment Section: EEE and Receiver Requirements, EEE Monitor APIs Specification, EEE Accounting APIs Specification, Experiment Data Receiver specification Section: Experiment Management Console Introduction
	Elias Tragos	NUIG	2017/10/26	NUIG Contributions V1
	Ronald Steinke	FOKUS	2017/10/27 2017/10/30	FOKUS contributions V1 FOKUS contributions V1 updates
	Tarek Elsaleh	UNIS	2017/10/27	UNIS contributions V1
	Ramnath Teja Chekka	KETI	2017/10/31	KETI contributions V1
V02	Rachit Agarwal	Inria	2017/11/01	V2 Generated
	Ramnath Teja Chekka	KETI	2017/11/03	KETI contributions V2
	Ronald Steinke	FOKUS	2017/11/07	FOKUS contributions V2 Updates
	Tarek Elsaleh	UNIS	2017/11/07	UNIS contribution V2 Updates
V03	Rachit Agarwal	Inria	2017/11/08	V3 Generated
	Ronald Steinke	FOKUS	2017/11/14	KETI contributions V3
	Ramnath Teja Chekka	KETI	2017/11/14	FOKUS contributions V3
V04	Rachit Agarwal	Inria	2017/11/14	V4 Generated, Conclusion
	Tarek Elsaleh	UNIS	2017/11/15	UNIS Contribution V4
	Elias Tragos	NUIG	2017/11/15	NUIG contributions V4
V05	Rachit Agarwal	Inria	2017/11/15	Final version ready for review
	Flavio Cirillo David Gomez Paul Grace	NEC UC ITInnov	2017/11/21 2017/11/21 2017/11/22	TR Review TR Review QR Review
V06	Rachit Agarwal, Elias Tragos, Tarek Elsaleh Ronald Steinke Ramnath Teja chekka	Inria	2017/11/25	Addressed comments
	Rachit Agarwal	Inria	2017/11/28	Deliverable ready for submission
Draft	Elias Tragos	NUIG	2017/11/30	Draft for submission

Overview of Updates/Enhancements over D4.7

Section	Description
2	Added new components and divided the section into two sections (section 2 and section 3)
4	Added FIESTA-IoT Technical Architecture
5	Modified API descriptions, Added new APIs
6	Added other tools section that describe specifications of the new components
7	Updated Portal description
8	Updated Implementation details

TABLE OF CONTENTS

1	Executive Summary/Introduction	7
2	Relation with the Functional Architecture	9
3	Requirements.....	10
3.1	Experiment Modelling.....	10
3.2	EEE and Receiver.....	11
3.3	Experiment Execution Result-set Data store	13
3.4	Experiment/Testbed Monitoring Tool.....	13
3.5	Portal.....	15
4	Fiesta-IoT Technical Architecture	16
4.1	Sequence Diagram	19
4.1.1	Starting Service with EEE.....	19
4.1.2	Sequence Diagram Experiment/Testbed Monitoring Tool	20
5	Other Tools.....	22
5.1	Experiment Result Store (ERS)	22
5.2	Experiment Data Receiver	22
5.3	UI Tools.....	23
5.3.1	Experiment Editor.....	23
5.3.2	Experiment Management Console (EMC).....	24
5.3.3	Reasoning tool (inference Tool)	26
5.3.4	FIESTA-IoT Acquisition Toolkit.....	45
5.3.5	Experiment/Testbed Monitoring Tool	47
6	Experimentation Services and API Specification.....	53
6.1	Experiment Deployment Services.....	53
6.1.1	Scheduling APIs	53
6.1.2	Subscription APIs.....	63
6.1.3	Polling APIs	65
6.2	Experiment Management Services	67
6.2.1	EEE Monitor APIs.....	67
6.2.2	EEE Accounting APIs.....	70
6.2.3	FIESTA-IoT Access Mechanism & Testbeds status APIs.....	71
6.3	Experiment ResultSet Storage APIs.....	76
6.4	Documentation of APIs	78
7	Prototype	79
7.1	PORTAL.....	79
7.1.1	Signing in.....	79
7.1.2	Menus	80
7.2	Usage	84
8	Implementation.....	93
8.1.1	Source Code Availability	93
8.1.2	Components	93
9	Conclusion	104
	References.....	105

LIST OF FIGURES

FIGURE 1: FIESTA-IoT FUNCTIONAL ARCHITECTURE COMPONENTS ADDRESSED IN THIS DELIVERABLE ARE MARKED IN GREEN.....	9
FIGURE 2: FIESTA-IoT TECHNICAL ARCHITECTURE (FULL VIEW)	17
FIGURE 3: FIESTA-IoT TECHNICAL ARCHITECTURE (EXPERIMENTER VIEW)	18
FIGURE 4: SEQUENCE DIAGRAM FOR STARTING AN EXPERIMENT	20
FIGURE 5: BOOTSTRAPPING OF THE TESTBED MONITORING.....	20
FIGURE 6: USER INTERACTION WITH THE TESTBED MONITORING	21
FIGURE 7: EXPERIMENTER INTERACTIONS WITH ERS.....	22
FIGURE 8: EXPERIMENT EDITOR INITIAL UI	23
FIGURE 9: FEMO XML PREVIEW	24
FIGURE 10: EXPERIMENT MANAGEMENT CONSOLE.....	25
FIGURE 11: CREATE RULE SCREEN	26
FIGURE 12: CREATE NEW RULE WHEN SEMANTIC EXPERT.....	27
FIGURE 13: CREATE NEW RULE WHEN SEMANTIC EXPERT –TEXT VIEW INPUT	28
FIGURE 14: CREATE NEW RULE - NON-SEMANTIC EXPERT.....	30
FIGURE 15: EXAMPLE OF RULE DETAILS.....	32
FIGURE 16: EDIT RULE INFORMATION	33
FIGURE 17: EDIT RULE CONTENT.....	34
FIGURE 18: RULE REGISTRATION HOME	35
FIGURE 19: REGISTER RULE- AVAILABLE RULES	36
FIGURE 20: REGISTER RULE - DETAIL RULE CONTENT	37
FIGURE 21: REGISTER RULE - SELECT SENSOR	38
FIGURE 22: REGISTER RULE – DETAILED INFORMATION	39
FIGURE 23: USER INTERFACE FOR EDITING A RULE REGISTRATION	40
FIGURE 24: RULE EXECUTION HOME PAGE	41
FIGURE 25: USER INTERFACE FOR CREATING A NEW RULE EXECUTION	42
FIGURE 26: EXECUTE RULE ON SENSOR BASE ON SPECIFIC TIME.....	43
FIGURE 27: RE-EXECUTE RULE.....	45
FIGURE 28: ANALYTICS TOOLKIT TABS.....	46
FIGURE 29: ANALYTICS INPUT TAB.....	46
FIGURE 30: ANALYTICS TOOLKIT RESULT	47
FIGURE 31: THE TESTBED MONITORING TOOL EMBEDDED IN THE FIESTA-IoT PORTAL	48
FIGURE 32: TESTBED MONITORING COMPONENT IN THE FIESTA-IoT PLATFORM.....	49
FIGURE 33: PORTAL WELCOME PAGE	79
FIGURE 34: PORTAL LOGIN PAGE	80
FIGURE 35: PORTAL STATISTICS PAGE	82
FIGURE 36: EXPERIMENT TEMPLATE FEMO.....	85
FIGURE 37: EXPERIMENT TEMPLATE FISMO	85
FIGURE 38: EXPERIMENT TEMPLATE QUERY	86
FIGURE 39: PORTAL EXPERIMENTER MENU.....	87
FIGURE 40: EXPERIMENT REGISTER CLIENT.....	88
FIGURE 41: EXPERIMENT REGISTER CLIENT - EXPERIMENT BROWSER	88
FIGURE 42: PART 1: EXPERIMENT DETAIL PANE	91
FIGURE 43: PART 2: ASSOCIATED FISMOS PANE	91
FIGURE 44: PART 3: SUBSCRIPTION PANE.....	91

LIST OF TABLES

TABLE 1: REQUIREMENTS ADDRESSED BY EXPERIMENT MODELLING TOOL (EXPERIMENT EDITOR)	10
TABLE 2: REQUIREMENTS ADDRESSED BY EEE AND RECEIVER	11
TABLE 3: REQUIREMENTS ADDRESSED BY ERS	13
TABLE 4: REQUIREMENTS ADDRESSED BY EXPERIMENT/TESTBED MONITORING TOOL	13
TABLE 5: REQUIREMENTS ADDRESSED BY PORTAL	15
TABLE 6: QUERY FOR CREATING A RULE	28
TABLE 7: SPARQL QUERY OR A RULE	31
TABLE 8: SPARQL QUERY	44
TABLE 9: SPARQL QUERY	50
TABLE 10: SPARQL QUERY	51
TABLE 11: ACCESS ROLES PER PORTAL MENU	83
TABLE 12: SYSTEM REQUIREMENTS FOR EXPERIMENT EDITOR	93
TABLE 13: DEPENDENCIES FOR EXPERIMENT EDITOR	94
TABLE 14: SYSTEM REQUIREMENTS FOR PORTAL	95
TABLE 15: DEPENDENCIES FOR PORTAL	95
TABLE 16: SYSTEM REQUIREMENTS FOR ERS	96
TABLE 17: DEPENDENCIES FOR ERS	97
TABLE 18: SYSTEM REQUIREMENTS FOR DATA RECEIVER	98
TABLE 19: DEPENDENCIES FOR DATA RECEIVER	99
TABLE 20: SYSTEM REQUIREMENTS FOR EXPERIMENT/TESTBED MONITORING TOOL	101
TABLE 21: DEPENDENCIES FOR EXPERIMENT/TESTBED MONITORING TOOL	101

TERMS AND ACRONYMS

Acronym	Meaning
API	Application Programming Interface
DOI	Domain of Interest
DSL	Domain Specific Language
EEE	Experiment Execution Engine
EMC	Experiment Management Console
ERM	Experiment Registry Management Component
ERS	Experiment Result Store
FAT	FIESTA-IoT Analytics Toolkit
FC	Functional Component
FEDSpec	FIESTA-IoT Experiment Description Specification
FEMO	FIESTA-IoT Experiment Model Object
FISMO	FIESTA-IoT Service Model Object
HTML	Hyper Text Markup Language
HTTP	HyperText Transfer Protocol
IOT	Internet of Things
IRI	Internationalized Resource Identifier
JSON	JavaScript Object Notation
OC	Open Call
UI	User Interface
URL	Universal Resource Locator
WP	Work Package

1 EXECUTIVE SUMMARY/INTRODUCTION

This deliverable is a second iteration [1]. It is noteworthy to assert that the current deliverable should not be considered as a standalone version but instead read along with deliverable [1]. This document provides an update to the reported tools and also provides technical details on newly developed tools that support experimentation over the FIESTA-IoT Meta Cloud (data store). Some key updates performed in the already reported tools, such as Experiment Execution Engine (EEE), Experiment Management Console (EMC) and Portal include:

- Updates to the API (Application Programming Interface) of Experiment Execution Engine,
- Updates to the User Interface (UI) of EMC to support:
 - Experimenters to get/know the required IDs for the getting data from Experiment Result Store (ERS)
 - Delete the scheduled FISMO (FIESTA-IoT Service Model Object) from EEE. Note that throughout this deliverable we will use FISMO and FEMO (FIESTA-IoT Experiment Model Object) but they should be considered as a Service (FISMO) within an Experiment (FEMO).
- Updates to the Portal include:
 - A newly designed interface to address issues raised previously, such as an enhanced menu that is based on the type of user (Experimenter, Testbed Admins and FIESTA-IoT Admins) and streamlined layout
 - Availability of new tools for different types of users of FIESTA-IoT, and
 - Support for the mobile version of portal.

Besides the above modifications and updates, we also report technical details on the newly developed and functional modules, such as Experiment editor, ERS and Experiment/Testbed Monitoring tool. Note that some technologies that we described in [1] towards building tools, such as Experiment Editor, are now not used to build the tool due to some limitations with regards to the handling of multiple users.

Within this deliverable, as reported in [1], we start by analysing the requirements collected in [2] for the developed tools, either new or existing, those coming from in-house experimenters [3], [4], Open Call (OC) participants and the validation done by in-house experimenters [5].

As this is the last technical deliverable in terms of tools provided under Work Packages 3 and 4 (WP3 and WP4), it is also essential that we report how the FIESTA-IoT platform technical architecture looks like and provide a brief overview of the interactions among those tools that facilitate experimentation over the FIESTA-IoT infrastructure. Note that, within this deliverable, we only focus on the part of FIESTA-IoT platform technical architecture that focuses on experimenters. Following the architecture, we provide updated sequence diagrams for “*starting the execution*” of the experiment using EEE. The update mainly reflects the integration of FIESTA-IoT Analytics Toolkit (FAT) and the possibility of scheduling the experiment on IoT-Registry or FAT.

For the monitoring tool, a technical description of it was provided in [6]. In this deliverable, however, we present the user side of the tool and focus on the UI. Additionally, we describe the new tools (those that were not reported before) such as ERS, Experiment Data receiver and UI interface for tools such as Experiment editor,

EMC and Reasoning tool to name a few. After this description, we then list the APIs that play an essential role in fulfilling the experimenters' needs. These APIs mainly include the scheduling, subscription, polling, monitoring, accounting and data download APIs provided by various tools such as EEE, monitoring and ERS tools.

To meet the users' expectations, performance analysis of the tools is an essential part once the tools are developed. As all the tools that support development, deployment and management of the experiment are either UI tools or delegate the requests to IoT-Registry, analysis of IoT-Registry with respect to the experimenters' need is essential. Since this assessment has been already carried out and been reported in [7], we do not report it here.

A simple mock-up walk-through of the FIESTA-IoT portal (that is now supported in various browsers either on Desktop or on Mobile) follows, with the aim to clearly explain to experimenters the workflow and steps to be performed in order to execute an experiment. Note that the steps reported in this deliverable only focus on one interaction path within the technical framework. However, we do not report workflow for experimenters that directly access IoT-Registry, rather we report the workflow that an experimenter need to follow when interacting with FIESTA-IoT tools such as Experiment Editor, EMC, EEE that facilitate the execution of the experiment.

A clear installation steps for the tools and how to use the above-mentioned components follow the section. The conclusion concludes the deliverable.

We also refer the audience to [8] and [1] in order to know more about WP4, its scope, related tasks and targeted audience.

2 RELATION WITH THE FUNCTIONAL ARCHITECTURE

The functional architecture of the FIESTA-IoT platform is described in [9]. An updated version of the same is available as in Figure 1. In relation to the platform, in this deliverable we are mainly focused on describing functionalities (other than those mentioned in [1]) that are now supported by both the FIESTA-IoT functional architecture and the FIESTA-IoT technical architecture. To provide a comprehensive view about the tools that are supported by the functional architecture and are reported in this deliverable, we refer readers to Figure 1. These tools are: EEE, Experiment Editor, Experiment Registry Module (ERM), EMC, ERS, and Experiment/Testbed Monitoring (performance monitoring). It should be noted that we have moved some of the tools already reported within WEB Browsing and Configuration Functional Component (FC) outside the FC. We mainly describe interactions between the components that enable experimentation over the FIESTA-IoT platform as part of the technical architecture (Section 4). As for the above-mentioned components, we present them in the next sections along with more requirements.

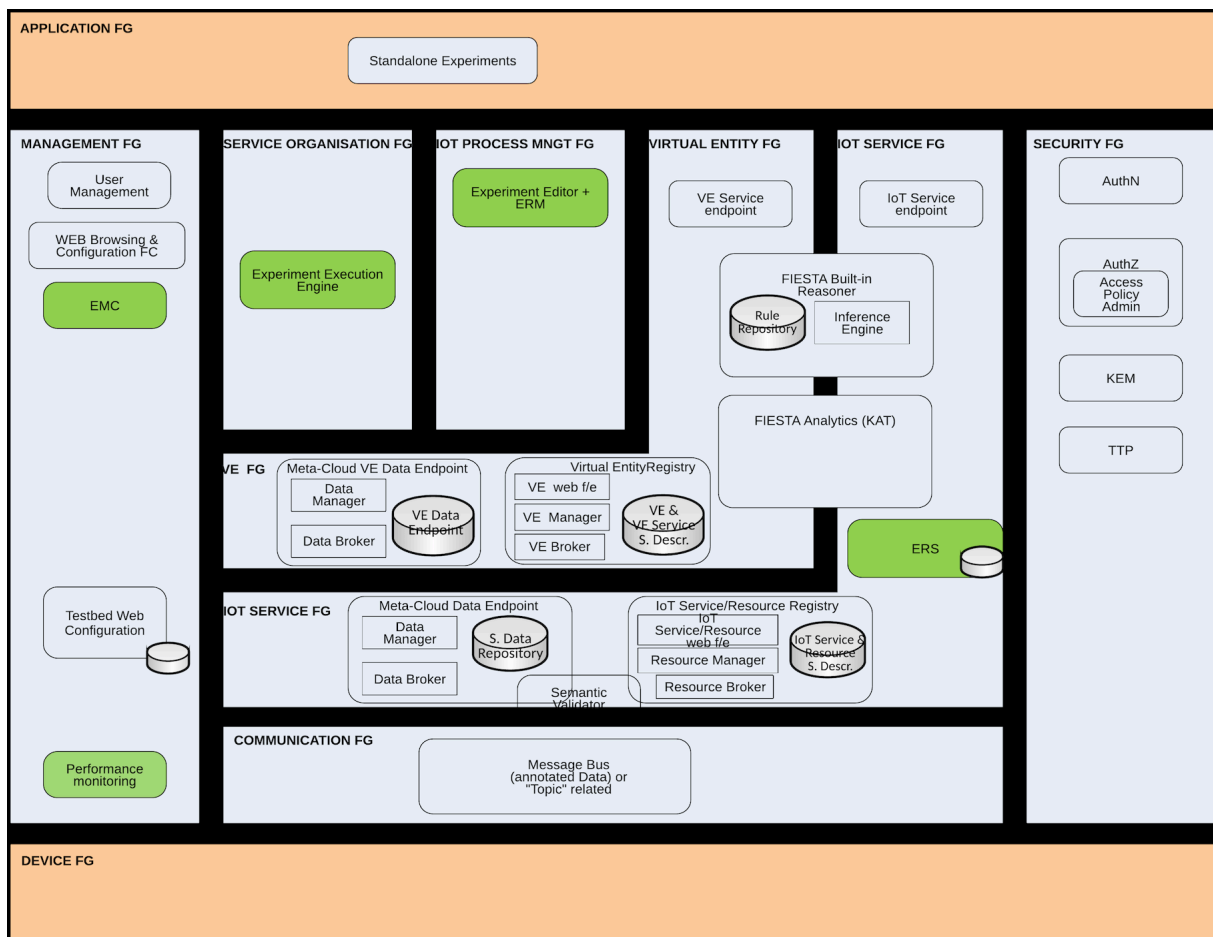


Figure 1: FIESTA-IoT Functional Architecture Components addressed in this deliverable are marked in green

3 REQUIREMENTS

In this section we provide how the requirements those proposed in [2] are fulfilled by the developed tools.

3.1 Experiment Modelling

Table 1 lists requirements addressed by Experiment Editor.

Table 1: Requirements addressed by Experiment Modelling tool (Experiment Editor)

Requirement ID	Description
24_NFR_ACC_Tools_planning_automato_tasks	FIESTA-IoT could provide tools to enable the planning of automated tasks
32_NFR_ACC_Provide_dev_deploy_manag_config_tools	FIESTA-IoT should provide development, deployment, management, and configuration tools
35_NFR_PLA_Manage_resources_in_query_or_experiment	FIESTA-IoT should visualize and manage the resources entailed in a specific query or experiment

For Experiment Modelling the Experiment Editor addresses the above three requirements. We next provide details so as to how these requirements were met.

- The experiment editor provides the experimenters a UI tool to build experiments (FEMO), services (FISMO) and Queries that would allow them to easily build task thereby fulfilling the requirements 24_NFR_ACC_Tools_planning_automato_tasks and 32_NFR_ACC_Provide_dev_deploy_manag_config_tools. The tool itself is explained later in the Section 5.3.1 while the workflow is provided in Section 7.2.1.2.1.

Note that the requirement

32_NFR_ACC_Provide_dev_deploy_manag_config_tools is also fulfilled by EEE as it deploys the configuration (in other words FEMO) that is created using Experiment Editor.

- With respect to the experiments, the tool also provides them the option to manage and configure the experiments (FEMOs) and services (FISMOs) thereby fulfilling the requirement 35_NFR_PLA_Manage_resources_in_query_or_experiment. Note that using the tool we allow the experimenters to manage their experiment. This is done using the capabilities that the UI provides and interactions the tool does with the ERM. As this tool is a UI based tool, the tool provides methods to visualize the needed attributes in the experiment.

3.2 EEE and Receiver

Beyond those described in the previous version of the deliverable [1], Table 2 lists requirements that are further satisfied by EEE and Receiver.

Table 2: Requirements addressed by EEE and Receiver

Requirement ID	Description
11_FR_ACC_Request_data_different_ways	It must be possible for the experimenter to request for data in different ways (e.g. event-based, periodic, and/or autonomous)
23_NRF_ACC_Page_in_subrequests	When a large set of information is requested, it should be possible to page it into different sub-requests
24_NFR_ACC_Tools_planning_automated_tasks	FIESTA-IoT could provide tools to enable the planning of automated tasks
30_NFR_ACC_FIESTA_well_documented	FIESTA-IoT must be well documented.
32_NFR_ACC_Provide_dev_deploy_manag_config_tools	FIESTA-IoT should provide development, deployment, management, and configuration tools
41_NFR_PLA_Minimize_processing_delay	Processing delay has to be minimized when requesting information
42_NFR_PLA_Data_generated_from_processing_info	Data generated from processing information could be provided to the experimenters
51_NFR_PLA_FIESTA_highly_reliable	FIESTA-IoT needs to be highly reliable
53_NFR_PLA_Execution_concurrent_services	The platform must support execution of concurrent Services including data generation and usage from the same resources

The EEE is a component that satisfies part of the non-functional requirement `32_NFR_ACC_Provide_dev_deploy_manag_config_tools` defined in [2]. Further, EEE is able to:

- Schedule at a defined rate a FISMO (a service Model that describes the experiment consisting of entities such as experiment control, details about the query, see [8]) as a Job on the Meta Cloud with minimum possible delay: this requires the EEE to read the `QuerySchedule` entity that is a part of FISMO, connect to the Meta Cloud and use the Meta Cloud API to execute the query defined in the `Query` attribute of the FISMO. EEE provides experimenters a functionality to subscribe to any services (FISMOS) on top of their own FISMOS, could request data in different ways for example, based on time period, and could poll for certain data (event based). This satisfies the `11_FR_ACC_Request_data_different_ways`. Different EEE APIs that support this requirement are reported in Section 6.1.

- In order to efficiently serve the experimenters with the data with minimum delays, EEE internally stores the service requests rather than contacting ERM component. Thus, EEE satisfies the 41_NFR_PLA_Minimize_processing_delay.
- Schedule multiple FISMOs on the Meta Cloud simultaneously. This satisfies 53_NFR_PLA_Execution_concurrent_services.
- Poll a service to get the data: the EEE enables the execution of the query defined in the Query entity of the FISMO once and on demand.
- After scheduling, EEE maintains a state variable of the scheduled job: this help experimenters to know the state of their experiment.
- EEE maintains a log of executed jobs: this enables an experimenter to know how many times a specific FISMO has been successfully executed.
- The Execution logs and state variables (processing information) is shown to the experimenter via EEE. This targets the 42_NFR_PLA_Data_generated_from_processing_info and is further complemented by the monitoring tool.
- EEE schedules FISMO queries on IoT-Registry or FAT in an automated manner where experimenters are needed to just start/stop the process on EEE. This feature enables 24_NFR_ACC_Tools_planning_auto_tasks.
- Provide a mechanism to the experimenters to subscribe/unsubscribe to a certain already discoverable FISMO: this enables experimenters to utilize already existing FISMOs in their experiments. In order to subscribe, the experimenter should provide the experimentOutput attribute in the FISMO so that the EEE could deliver the output accordingly.
- On top of subscription, if an owner deletes a FISMO, then the subscribers will not be notified about the deletion: this allows subscribers to keep execution of the subscribed FISMO ongoing until the FISMO is unsubscribed.
- Be able to delete any experimenter related executing job from the EEE along with its history.
- EEE is able to invoke related widgets like FIESTA-IoT Analytics toolkit besides just interacting with IoT-Registry.
- EEE is able to send data in different formats required by experimenters.
- For a large resultset paging of the result is provided: this enables the 23_NRF_ACC_Page_in_subrequests. This is achieved by the Sending module of the EEE that breaks the large datasets into multipart before sending it to the Receiver that concatenates these parts into one.
- The EEE is stable and satisfy the 51_NFR_PLA_FIESTA_highly_reliable.
- On top of above, all the APIs of the EEE are well documented (see Section 6.4) and made available to the experimenters so that they can understand the working of the EEE better. This ensures fulfilment of 30_NFR_ACC_FIESTA_well_documented with respect of EEE.

3.3 Experiment Execution Result-set Data store

Table 3: Requirements addressed by ERS

Requirement ID	Description
52_NFR_PLA_Elasticity_abundance_computational_assets	FIESTA-IoT should have elasticity and abundance in terms of computational assets (especially storage and computation)

The fit criterion with respect to the only requirement (listed in Table 3) satisfied by ERS states that “FIESTA-IoT is able to store data from experiments during the requested period and process any experiment that did not expire”. The ERS addresses this requirement by providing a storage facility for experimentation results (or data), which can be retrieved by the experimenter at a convenient time. Section 5.1 explains how this storage facility works.

3.4 Experiment/Testbed Monitoring Tool

Table 4: Requirements addressed by Experiment/Testbed Monitoring Tool

Requirement ID	Description
15_FR_ACC_Discover_resources_by_characteristics	It must be possible to get/discover resources based on characteristics
22_NFR_ACC_Distinguish_type_of_data	It must be clear to the experimenters what they are receiving, (e.g. measurements, metadata, resources, characteristics, etc.)
30_NFR_ACC_FIESTA_well_documented	FIESTA-IoT must be well documented.
35_NFR_PLA_Manage_resources_in_query_or_experiment	FIESTA-IoT should visualize and manage the resources entailed in a specific query or experiment
39_NFR_PLA_Info_testbed_agnostic_way	FIESTA-IoT must handle information in a Testbed agnostic way
40_NFR_PLA_Process_feedbacks	FIESTA-IoT should process the measurements and / or resources feedback to validate the functioning of resources
41_NFR_PLA_Minimise_processing_delay	Processing delay has to be minimised when requesting information
42_NFR_PLA_Data_generated_from_processing_info	Data generated from processing information could be provided to the experimenters
49_NFR_PLA_Reliable_time_sync	FIESTA-IoT should support testbeds in different time zones
64_NFR_RES_Resource_provide_characteristics	Every resource must be characterised

65_NFR_RES_Resource_identified_code	Every resource must be univocally identified by a code
--	--

The Experiment/Testbed Monitoring Tool satisfies parts of the non-functional requirement 35_NFR_PLA_Manage_resources_in_query_or_experiment. Additionally, the non-functional requirements described in 40_NFR_PLA_Process_feedbacks and 42_NFR_PLA_Data_generated_from_processing_info were addressed. Further, the Experiment/Testbed Monitoring tool is able to:

- Show the monitored testbeds in a total view and showing the number of active resources per testbed. Additionally, it shows every sensor per testbed with metadata and latest observations. This addresses partially 35_NFR_PLA_Manage_resources_in_query_or_experiment
- Process the gathered data of the testbeds used for monitoring in order to find not working resources. The result of the processed data shall be available via the API as well as the collected and transformed data. This addresses mainly the two requirements 40_NFR_PLA_Process_feedbacks and 42_NFR_PLA_Data_generated_from_processing_info
- Retrieving the data from the IoT-Registry and storing it in a transformed way into another database in order to prepare the data for the visualization and for providing it via the API. This assures 41_NFR_PLA_Minimise_processing_delay
- Providing the summarized overview and the detailed view per testbed in a way that it is clear for the experimenter which information he/she is retrieving. This fulfils 22_NFR_ACC_Distinguish_type_of_data, 49_NFR_PLA_Reliable_time_sync, 64_NFR_RES_Resource_provide_characteristics and 65_NFR_RES_Resource_identified_code.
- Using the additional IDs per stored resource but also linking to the original ID, which is used in the IoT-Registry. So not only the combined and transformed resources used in the Monitoring Tool can be addressed but also the original resources. This makes sure that 39_NFR_PLA_Info_testbed_agnostic_way is still fulfilled.
- The API is documented but also self-explanatory in its usage, as required by 30_NFR_ACC_FIESTA_well_documented.
- The API provides methods to mimic the same filtering methodology as it is used in the IoT-Registry, e.g., filtering resources by phenomena. This fulfils 15_FR_ACC_Discover_resources_by_characteristics.

3.5 Portal

Table 5: Requirements addressed by Portal

Requirement ID	Description
20_FR_SEC_Experimenter_single-sign-on	Single-sign-on mechanism has to be in place
24_NFR_ACC_Tools_planning_auto_tasks	FIESTA-IoT could provide tools to enable the planning of automated tasks
30_NFR_ACC_FIESTA_well_documented	FIESTA-IoT must be well documented.
35_NFR_PLA_Manage_resources_in_query_or_experiment	FIESTA-IoT should visualise and manage the resources entailed in a specific query or experiment

The FIESTA-IoT portal plays the role of the user interface for all types of users. It incorporates interfaces for managing and handling all functionalities provided by the FIESTA-IoT platform. For logging in, a single security mechanism is used and the users are only required to login once, they get a token, initiating a session and then they get access to all functionalities, without the need to login separately (addressing the 20_FR_SEC_Experimenter_single-sign-on requirement) [10].

The portal includes also modules for running automated tasks, especially for the registration of multiple resources (at once) or for scheduling the execution of experiments (addressing the 24_NFR_ACC_Tools_planning_auto_tasks requirement).

Additionally, the portal provides a simple visualization tool for the results of a query/experiment, so that experimenters can have a first look at the results.

Testbed providers and experimenters can also see real-time information about the testbeds and the registered resources, to see which are online and sending data and use it for debugging purposes (addressing the 35_NFR_PLA_Manage_resources_in_query_or_experiment requirement).

The FIESTA-IoT portal includes also a help section dedicated to the documentation of all tools, services and APIs for the experimenters and the testbed providers (addressing the 30_NFR_ACC_FIESTA_well_documented requirement).

4 FIESTA-IOT TECHNICAL ARCHITECTURE

A technical version of the FIESTA-IoT platform architecture is provided in Figure 2. Note that, within the scope of this deliverable we are only limited to describe those components that address the experimentation plane (upper part of the Figure 2). Figure 3 provides this view. We next provide a brief description of the functionality of the components:

- **IoT-registry:** This component is the cornerstone of the FIESTA-IoT platform. It is the module in charge of handling the semantic information that flows across the FIESTA-IoT platform. Basically, it undertakes the control of the triple-store and internally holds the overall semantic meta-repository. This component is already described in [7] thus it is not described in this deliverable.
- **Experiment Registry Management (ERM):** It is the registry where all the experiments are stored. The Experiment Execution Engine and the Experiment Management Console use the ERM APIs to read the information stored about the experiment and take actions accordingly.
- **Experiment Management Console (EMC):** It is the User Interface (UI) to the Experiment Execution Engine (EEE). Using this an experimenter can control the execution of the FISMOs beyond what is specified via FEDSpec (FIESTA-IoT Experiment Description Specification). Using EMC an experimenter can also know other related information about the experiment that he provided in the FEDSpec.
- **Experiment Execution Engine (EEE):** Engine that executes the experimenter's need on the IoT-Registry at a specified schedule. It defines a set of services/APIs that are essential for the execution of the experiment. The EMC uses EEE APIs to provide experimenters the execution related information.
- **Experiment Result Store (ERS):** ERS stores the results that are not been sent to the experimenter due to any reason like unavailability of receiver etc.
- **Experiment Data Receiver (Receiver):** This component is usually executed on the experimenter side and not on FIESTA-IoT side. This component opens a channel for receiving data from EEE after the execution of the query.
- **Experiment Editor (Editor):** This component enables experimenters to quickly create FEDSpecs and deploy them on the FIESTA-IoT platform. These FEDSpecs will then be read by EEE and executed accordingly.
- **FIESTA-IoT Analytics Toolkit (FAT):** This component enables experimenters to execute data analysis techniques on datasets retrieved from IoT-Registry.
- **FIESTA-IoT Monitoring:** This tool allows experimenters to view basic statistics of the data available within FIESTA-IoT ecosystem. It also allows experimenters to know which testbeds are pushing data and how many resources are active in the moment.
- **Reasoning:** this tool allows experimenters to define their own reasoning rules (or re-use rules defined by other experimenters) to run on top of the gathered data in order to extract some results. The rules are in the form of “if-then” and can be run on current or historical data streams.

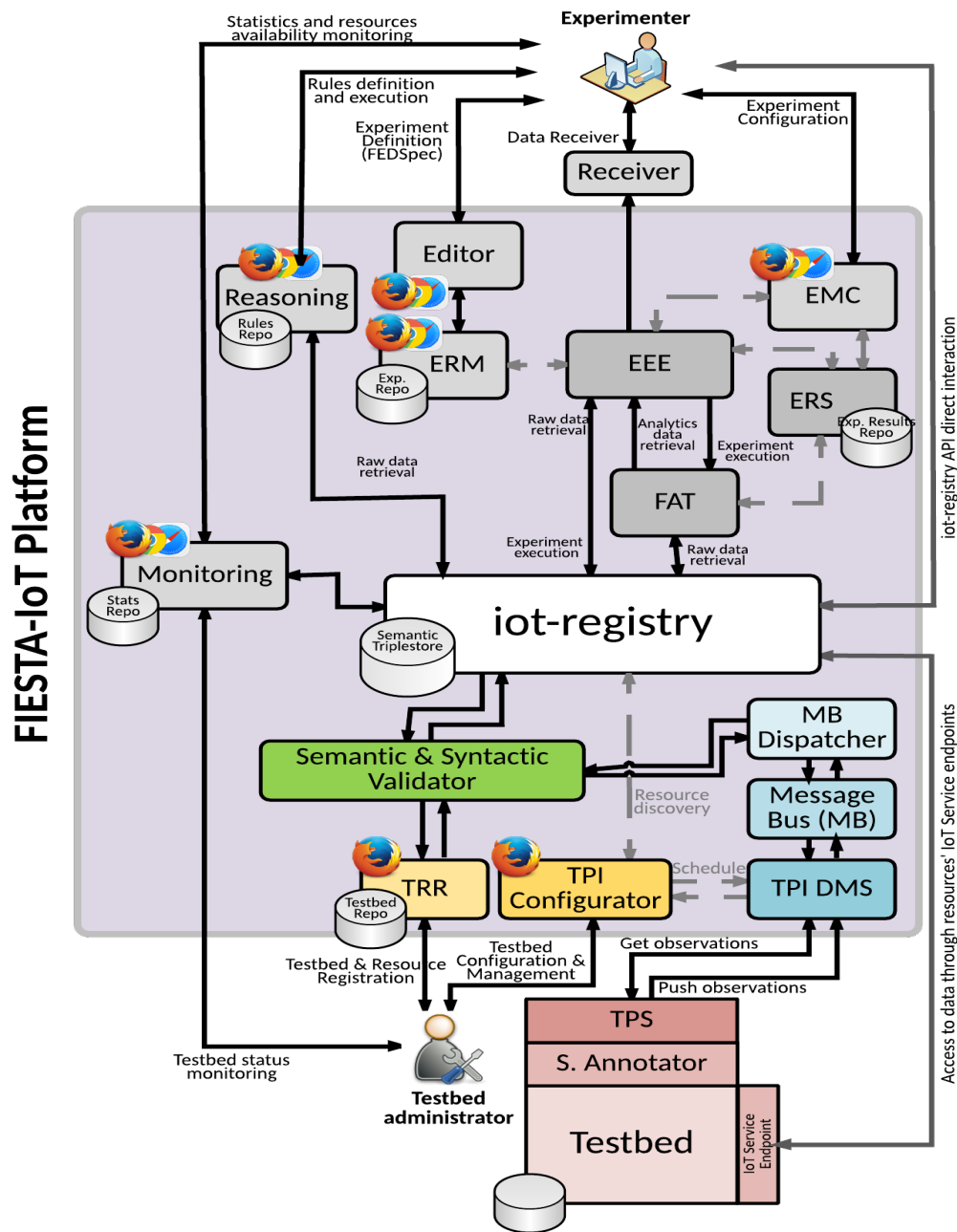


Figure 2: FIESTA-IoT Technical Architecture (Full View)

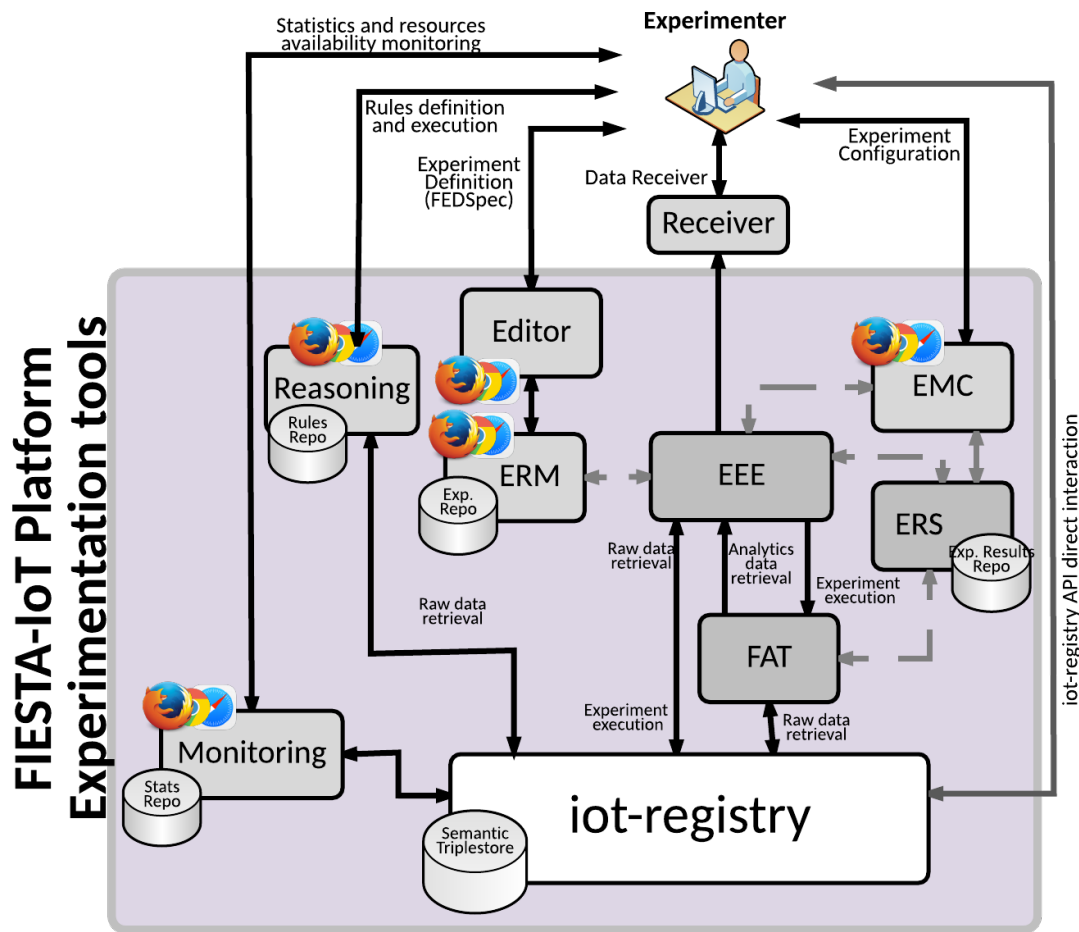


Figure 3: FIESTA-IoT Technical Architecture (Experimenter view)

An experiment is defined as “*Experiment is a test under controlled conditions that is made to demonstrate a known truth, examine the validity of a hypothesis, or determine the efficacy of something previously untried*” [11]. Nevertheless, as discussed previously we focus on data-oriented experimentation that can be performed on IoT data stored in the FIESTA-IoT platform. To support experimentation, tools that enable development, deployment and management of an experiment are developed and integrated to fulfil the execution of an experiment. To brief about the tools (see Figure 3), using a UI tool such as Experiment Editor an experimenter can create or develop the DSL for the experiment based on their needs. This DSL, also called as a FEDSpec, contains the specification for the EEE tool to execute the defined experiment. EEE essentially schedules or deploys the experiment on the FIESTA-IoT ecosystem based on the provided specifications. The Experiment Editor uses ERM to save a FEDSpec within the FIESTA-IoT ecosystem. EEE then reads the specifications to schedule the experiment on the FIESTA-IoT ecosystem. EEE is accompanied by an experiment controlling and management UI (Experiment Management Console or EMC) that enables experimenters to view execution summary and control the execution of their experiment. Once an experiment is executed by the EEE, the output is sent to experimenters, who have to enable a Receiver on their side to get and handle the results. In case these results are not delivered to the experimenter, they are stored in an ERS repository where experimenters can download the results at will. Nonetheless, these tools are also complemented by tool-specific dedicated public APIs using which

experimenters can also develop their own experiment workflow in case needed. In another case if an experimenter does not want to use such tools, they can create their own experiment execution like module and query directly the IoT-Registry using the public IoT-Registry APIs.

A description of some of the components such as EEE and EMC was provided in the V1 of the deliverable [1]. However, in this deliverable we focus on the components that were not described previously. In the next subsection, we present, modified sequence diagrams with respect to EEE functionality. We also show sequence diagrams for the new components. For components like FAT, deliverable [6] provides more technical details.

4.1 Sequence Diagram

The addition of the data analytics, result storage, and Experiment modelling has led to a modified sequence diagram for the EEE. Note that sequence diagrams presented in V1 of this deliverable are still valid except the starting of a service.

4.1.1 Starting Service with EEE

The updates to “starting of a service” sequence diagram is provided below in Figure 4. Here, we introduce FAT, Sender module and the ERS. If an experimenter defines widget parameter in the FISMO object of the FEDSpec, [12], towards the usage of FAT EEE calls the FAT APIs instead of calling IoT-Registry APIs directly. FAT then calls the IoT-Registry APIs, gets the results and stores them in the ERS. Experimenters are then required to call the ERS APIs to get the results. In Section 6.3 we define the ERS APIs. Instead, if the experimenter does not specify the widget, the EEE calls the IoT-Registry API to retrieve the results of the query specified in the FISMO object. EEE upon a successful response from IoT-Registry, sends the results to the experimenter to the URL endpoint specified by them. If the send fails due to any reason, the EEE stores the results in the ERS for experimenters to later get the results.

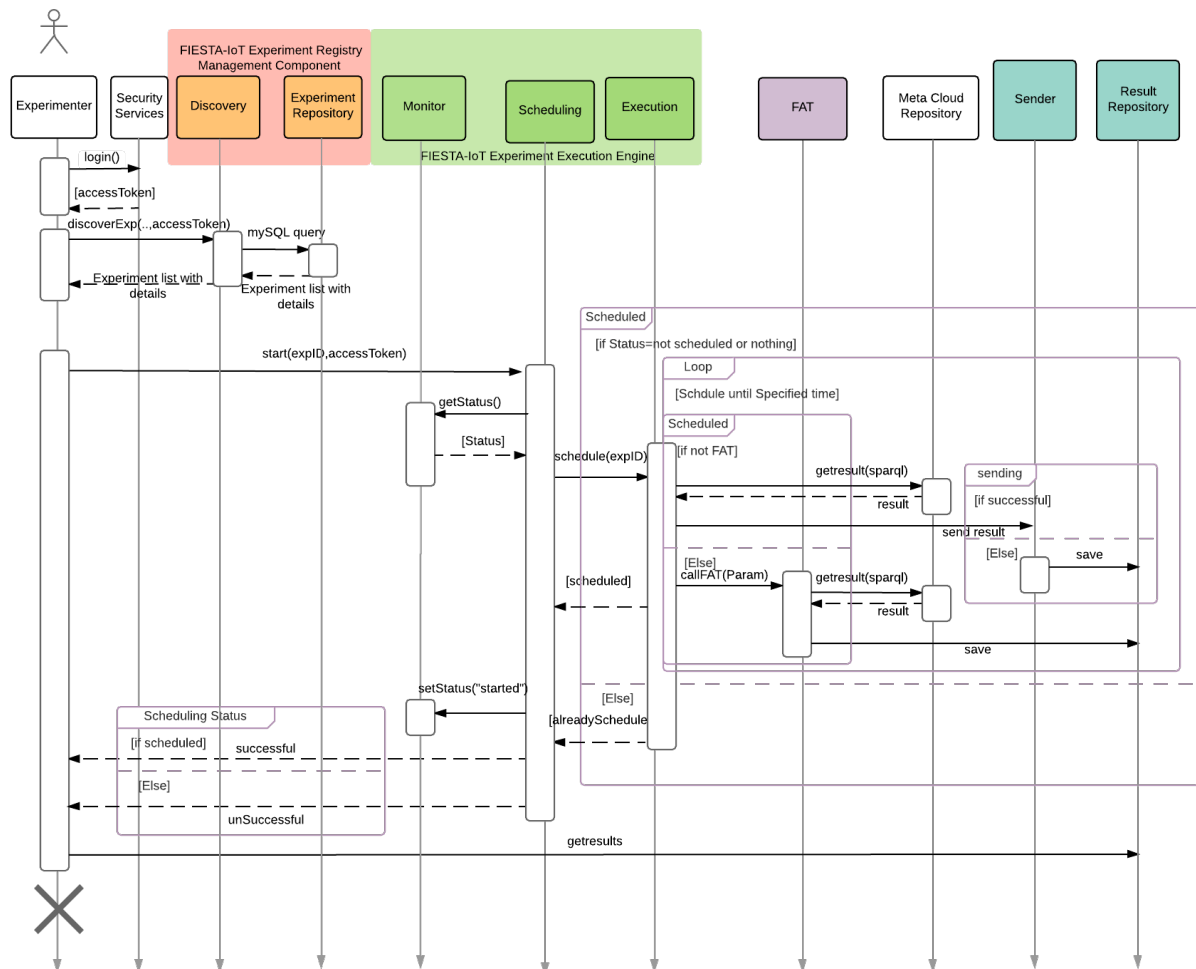


Figure 4: Sequence Diagram for starting an Experiment

4.1.2 Sequence Diagram Experiment/Testbed Monitoring Tool

This section provides sequence diagrams for the Experiment/Testbed Monitoring tool.

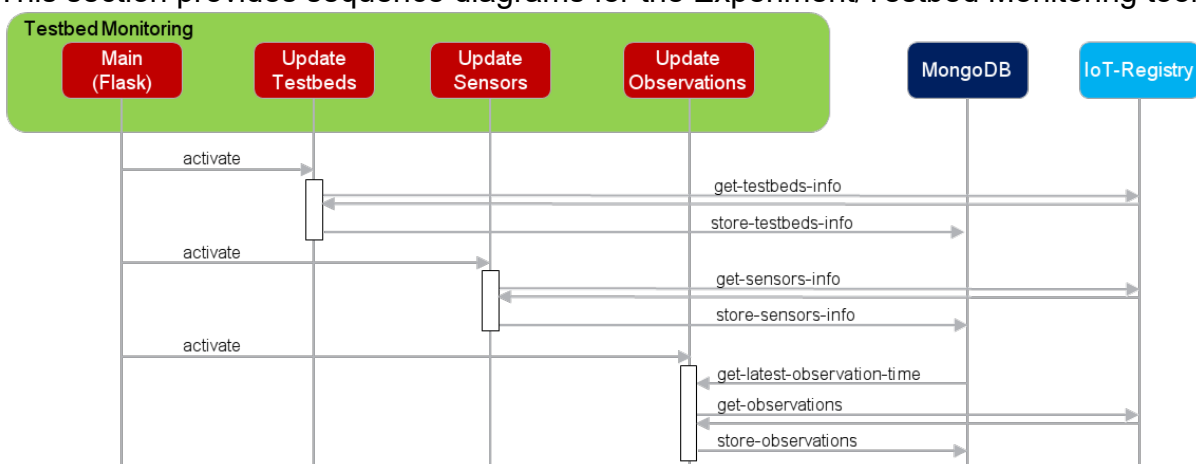


Figure 5: Bootstrapping of the Testbed monitoring

The bootstrapping of the Testbed Monitoring can be seen in Figure 5. First the tasks components “Update Testbeds”, “Update Sensors” and “Update Observations” for

retrieving the needed data from the IoT-Registry are started by the “main” component. The retrieved data is transformed and is stored into the mongoDB. This makes sure that enough data is available for the UI and the API to serve requests properly.

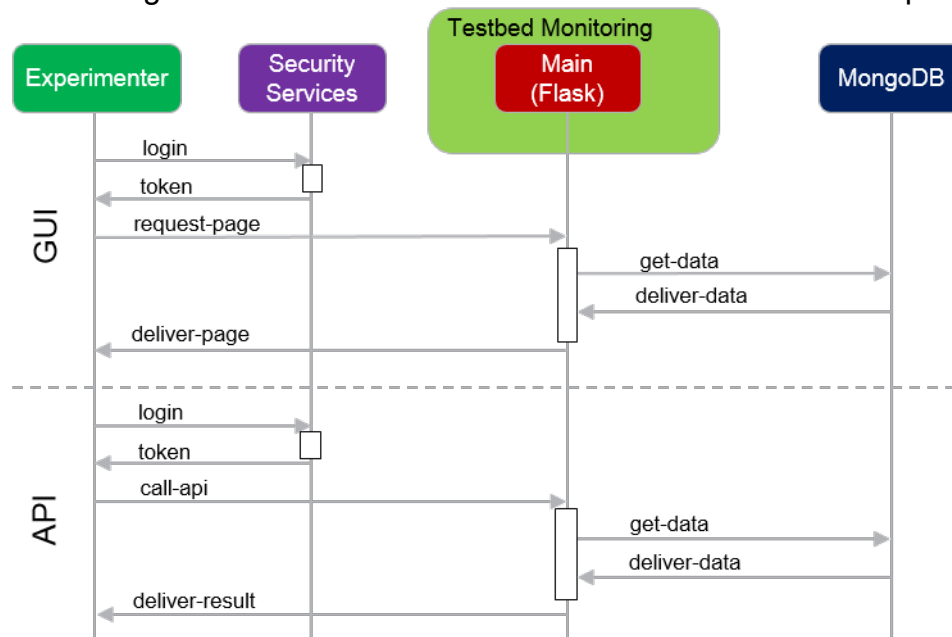


Figure 6: User Interaction with the Testbed Monitoring

In Figure 6 the two different options for retrieving either a page from the GUI or doing a request against the API are shown. The required login is for both operations necessary. Also for every operation the data will be retrieved from the mongoDB in order to serve the request in proper time.

5 OTHER TOOLS

5.1 Experiment Result Store (ERS)

The ERS is a component within the Experiment Execution subsystem that provides a temporary storage mechanism for experiments executed in an asynchronous manner. This component allows experimenters to retrieve the results of their experiments at their convenience. It should be noted that currently a result is removed from ERS once the experimenter retrieves it.

Figure 7 shows the interactions that an experiment undergoes for the data to be stored in the ERS. In the first step (as numbered in Figure 7), the experimenter invokes the EEE to process their experiment. In the second step, the EEE will in turn invoke an IoT service to retrieve a dataset. This invocation of IoT service consists of invoking components like FIESTA-IoT Analytics toolkit or IoT-Registry. In the third step, the EEE will typically handle the request and store the result in the ERS. In the case of the FAT service, FAT will forward the result dataset directly to the ERS. The Experimenters can then call the ERS API to retrieve for results of their experiments.

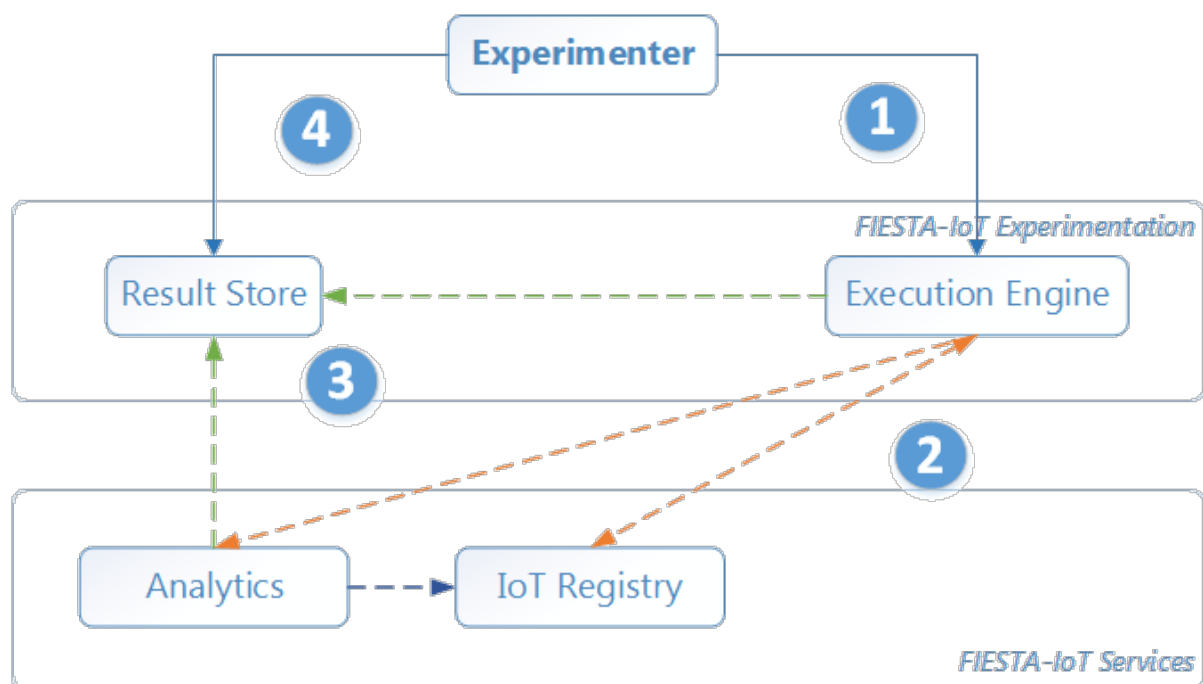


Figure 7: Experimenter interactions with ERS

5.2 Experiment Data Receiver

As a sample, FIESTA-IoT should provide an Experiment Data Receiver that should open a possibility for Experimenters to receive the data made available via EEE. Experimenters can use this tool on their dedicated servers to receive the data. The tool should be able to receive large data objects by the means of multipart file upload. Internally this tool should be able to then save the received data in particular location that is specified in the configuration of the tool.

5.3 UI Tools

5.3.1 Experiment Editor

The Experiment Editor is a UI tool that experimenters could use to model and edit an experiment. Once directed to the Experiment Editor, an experimenter would get a view as shown in Figure 8 with a rectangular block and a number (variable) of square blocks. The rectangle block contains:

- a number denoting the number of FEMOs created by the experimenter,
- a search icon that is used to find the FEMO when domain of interest (DOI) is provided and
- an “+” icon that represents “add a FEMO”, i.e., to create new FEMO.

Each square block represents a FEMO that the experimenter has previously created. The FEMO block consists of FEMO’s name, description, number of associated FISMOS and list of DOIs that are highlighted in different colour. Further, within each of these FEMOs’ specific “square block”, there would be three choices of operations that would represent (a) duplicate (b) edit, and (c) delete.

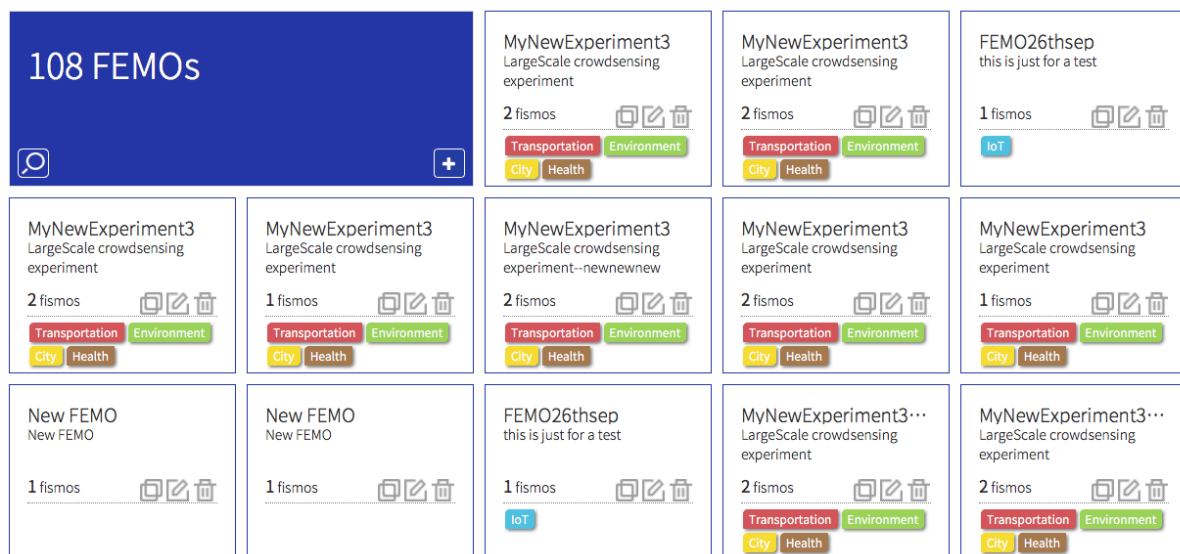


Figure 8: Experiment Editor initial UI

For the duplicate option, represented by the overlapping square boxes, by clicking on it would create a new FEMO with the same parameter settings as the original FEMO. The APIs that are used in the process are listed in the Scheduling API Specification Section 6.1.1.

Every FEMO can be edited. This can be done by clicking on the FEMO block or by clicking the “edit” icon. Once the parameters are changed, the experimenter can commit the changes by clicking on the save button as shown in Figure 36 (Experiment Template FEMO). The Edit feature is applied at the following three levels of experiment: FEMO, FISMO and Query Control. Once the changes are made at any given level, the Experiment Editor notifies the EEE and ERM about the state change

of the experiment. Before “saving” the changes, the experimenter could review the changes using the “Preview” (as shown in the Figure 9) option available beside “Save” option.

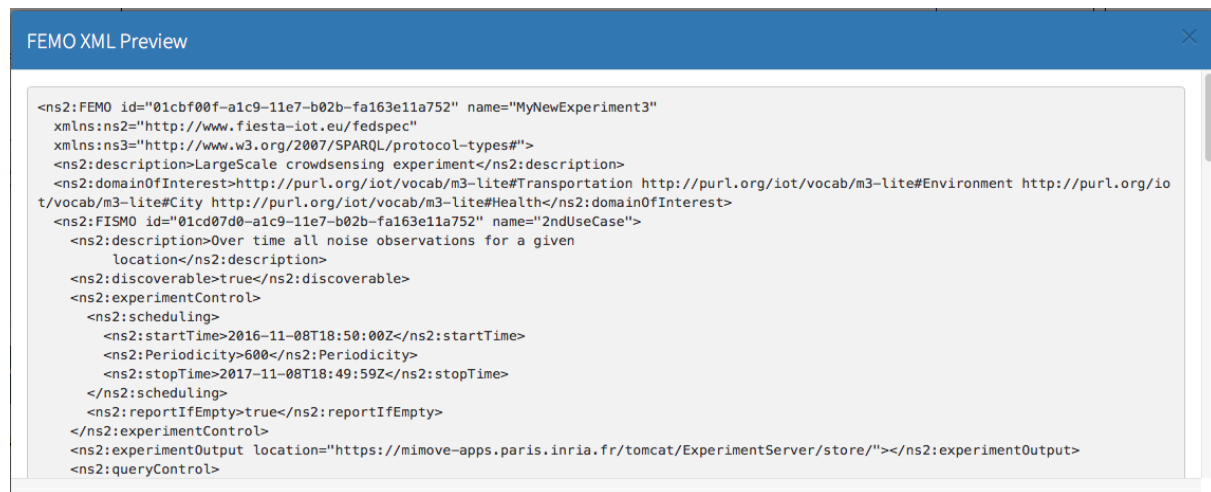


Figure 9: FEMO XML Preview

As stated before, an experimenter can delete an Experiment by clicking the “delete” icon on the FEMO block. This would trigger Experiment Editor to notify the EEE and ERM of any experiment termination.

5.3.2 Experiment Management Console (EMC)

The EMC is a UI where the experimenter could know about the status of their experiment(s). The EMC would list experiments associated to an experimenter. Upon selecting a specific FEMO, say “InriaExperiment” as in Figure 10, the details of the experiment should be presented to the experimenter. This includes FEMO details, associated FISMOS and other discoverable FISMOS. An experimenter should be able to see the experiment ID, name, description and domain of interest. On top of this, experimenters should have it handy the API through which they can download the experiment results that were not sent to them due to some errors. Towards this, a description or a footnote should be present that reflects this.

The “Associated FISMO” tab shows the “meta” information about the FISMO. This “meta” information includes:

- The jobID of the FISMO if it is scheduled, if it is not scheduled then “Not Yet Scheduled” information is displayed,
- The name and description of the FISMO,
- Experimenters can also start/stop a particular FISMO. By default, all the FISMOS would have status set to “Not Yet Scheduled”. The experimenter needs to explicitly start the FISMO to schedule it in the EEE. This would change the status to “Scheduled” in the UI,

- The “Start Now” and “Stop Now” only provide experimenters the information to either start the schedule or if the schedule already exists in the EEE then pause the schedule of the respective FISMO.
- The experimenters are able to view the logs of the “Past executions”. This includes information like date-time when the FISMO was successfully executed with the size of the data consumed by the FISMO from the Meta Cloud.
- An experimenter is able to delete any scheduled FISMO. In case a FISMO is not scheduled the experimenter is not able to delete the FISMO (i.e., they do not see the delete button). If deleted, the FISMO is deleted from the EEE along with all its references within EEE.
- For “Subscribed” FISMOS (as such FISMOS are not owned), the relevant information is shown including ownership status as “subscriber” and option to unsubscribe the subscription.
- Nevertheless, other than the above functionality experimenters should also poll for results.

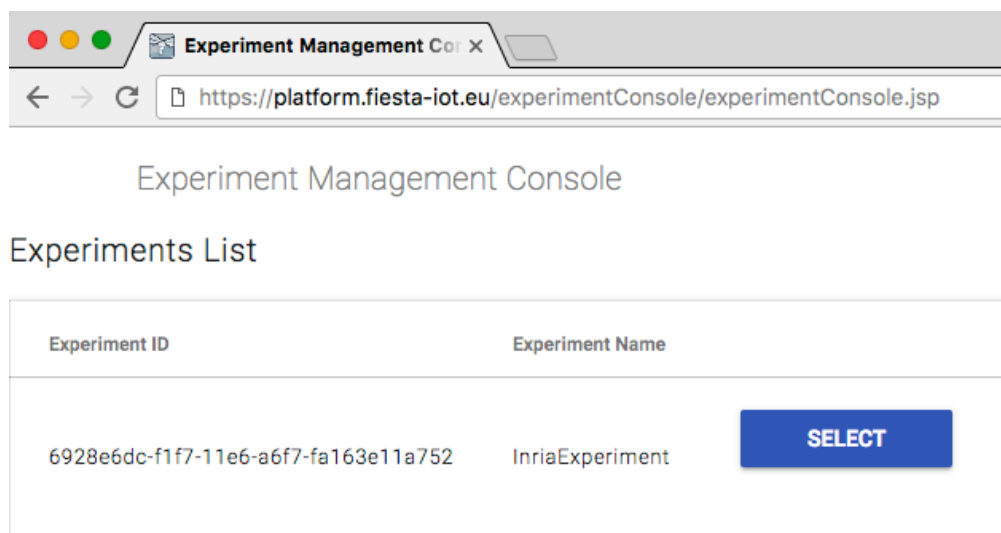


Figure 10: Experiment Management Console

The EMC should also provide an option for the experimenters to subscribe to already available FISMOS within the FIESTA-IoT ecosystem. As the FISMOS are already defined, the experimenter is able to:

- View the existing FISMO of choice,
- Provide URL location where the results of the execution of the subscribed instance of the FISMO should be sent and
- Subscribe the FISMO with the new URL location.

Nonetheless, despite subscribing using the URL location, the experimenter should not be able to change any other parameter of the FISMO they subscribe to.

5.3.3 Reasoning tool (inference Tool)

The FIESTA-IoT Reasoning component is an implementation of a semantic reasoner that works on top of the FIESTA-IoT platform. The reasoner engine was described in detail in Deliverable D3.6 [6], providing also details for the API for accessing the reasoning services. In this deliverable, we describe the UI developed within FIESTA-IoT and accessed through the portal, so that it can be used as a tool for experimenters regardless if they are experts in semantics or not. Along with this description, the readers are advised to read the respective Section 3.3 of D3.6 to become more familiar with the architecture of the reasoning engine. Briefly, with the reasoning tool the experimenters will be able to create inference rules in the form of expressions “if (condition) then (result)” for example:

- If (temperature) > (25degrees) then (notify_hot)
- If (speed) < (30km/h) then (notify_traffic)
- If (temperature) < (19degrees) and (humidity) > (60%) then (notify_unhealthy)

5.3.3.1 Rule Creation

An experimenter could create new rules in two ways: as a semantic expert or as a non-semantic expert. These actions could be performed on the FIESTA-IoT portal, where there is a menu called “Reasoning”, which has 3 sub menus: Create Rule, Register Rule, and Execute Rule. Note that the tool can also be used as standalone via dedicated APIs, which were mentioned in [6].

ID	Name	User Id	Description
8	Demo rule 1	testuser1	Demo rule 1
9	test1	etragos	
10	Myrule1	etragos	this is a description
11	Myrule2	etragos	
12	test22	etragos	
13	Demo register one	etragos	Demo register one
14	newPower	etragos	

Showing 1 - 7 of 7 items.

Figure 11: Create Rule Screen

5.3.3.1.1 Create new Rule – Semantic Expert

The FIESTA-IoT Reasoning module provides a simple UI (see Figure 11) for enabling experimenters to easily write the rule on a text-view. For assisting the experimenters in this process, the UI also provides sensor information base on the selected quantity kind, so that experimenters can easily see information for the sensors, so that they have a more detailed view when they create their own rule. Here, information like sensor ID, sensor quantity kind, sensor unit of measurement, sensor latitude, sensor longitude or current sensor data is presented as shown in Figure 12 and Figure 13.

Create or edit a Rule - For Semantic Expert

Name
Demo rule1

Description
Demo rule1

Select Quantity
Power

Select Sensor

- https://platform.fiesta-iot.eu/iot-registry/api/resources/VsnDY_ipleAhy2eCc5jxNRqGyBVslwso2bO-8KCr7GKnfKLgda8TdXltkjaADUHLt
- https://platform.fiesta-iot.eu/iot-registry/api/resources/5E2L0nnAsD75WlqCk2vqnHtsRwUk2331TjeHpQR7D_luqZKoFebr_8XTtixWgU
- https://platform.fiesta-iot.eu/iot-registry/api/resources/Ur7Q-GLgxiLsfK4ZhXffEryue052DxDQzb8jxqKMPyLJZUiTr-ZpAj1ZK_hi302o5g
- https://platform.fiesta-iot.eu/iot-registry/api/resources/DYRIY6yuZg7IzBNuQqbJf64IYV3IDcdGKVtYLCndHijL5P7H3wKu_47BcAU00q1
- <https://platform.fiesta-iot.eu/iot-registry/api/resources/tRRAK2IA6S5GEca2qPQD6hWzOn-klp82OXHnXltm16LbPISitapxvtgEcrrPmWu>
- <https://platform.fiesta-iot.eu/iot-registry/api/resources/x1AlxibeGRXJDPubYHcB9Wol22kDITewzJR1t445JQflPuv0YJivjsrb14DRkpj7mVv>
- <https://platform.fiesta-iot.eu/iot-registry/api/resources/KwobGd67MC71eEJb3xTNYjk1LNIOfVdJJ1DOsMHlmV0ByWW35bZV3NYM5Bh>
- https://platform.fiesta-iot.eu/iot-registry/api/resources/i6zudXsHdvXrTmJqDEM3Rr2QBF8x5823XGO_9AndMxLUGwQo4WkyGC8O6x
- <https://platform.fiesta-iot.eu/iot-registry/api/resources/RXjeO4KJxjS1R4MMYz3vDOrYlHyH2MAf6kSXmomkuVmQn-Hs5HDWVpmvgNd>
- https://platform.fiesta-iot.eu/iot-registry/api/resources/PMX3CfeJ3cuWxJhrXxpFzjqQXS9ARDLVmpOeL53uE-2tPXZJKsExRF_Z3P0Ei

Selected Sensor ID
https://platform.fiesta-iot.eu/iot-registry/api/resources/Ur7Q-GLgxiLsfK4ZhXffEryue052DxDQzb8jxqKMPyLJZUiTr-ZpAj1ZK_hi302o5g

Latitude
51.243343

Longitude
-0.5932438

Quantity Kind
<http://purl.org/iot/vocab/m3-lite#Power>

Unit of Measurement
<http://purl.org/iot/vocab/m3-lite#Watt>

Figure 12: Create new Rule when Semantic expert

Rule content

This field is required.

Cancel Save

Figure 13: Create new Rule when Semantic expert –Text view input

Experimenters can create a new simple rule with the “if then” logic within a query as shown in Table 6. In this example, we apply rule “if power_consumption>0.56 Watt then notify experimenter for high consumption”:

Table 6: Query for creating a Rule

```
@prefix iot-lite: <http://purl.oclc.org/NET/UNIS/fiware/iot-lite#> .
.
@prefix m3-lite: <http://purl.org/iot/vocab/m3-lite#> .
@prefix ssn: <http://purl.oclc.org/NET/ssnx/ssn#> .
@prefix geo: <http://www.w3.org/2003/01/geo/wgs84_pos#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix dul: <http://www.loa.istc.cnr.it/ontologies/DUL.owl#> .
@prefix time: <http://www.w3.org/2006/time#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix reasoning: <https://fiesta-iot.eu/reasoning#> .
(?observation rdf:type ssn:Observation),
(?observation ssn:observedProperty ?observedProperty),
(?observedProperty rdf:type m3-lite:Power),
(?observation ssn:observationResult ?sensorOutput),
(?sensorOutput ssn:hasValue ?obsValue),
(?obsValue dul:hasDataValue ?dataValue),
(?obsValue iot-lite:hasUnit ?unit),
(?unit rdf:type m3-lite:Watt),
greaterThan(?dataValue, "0.56"^^xsd:double) -> (?observation
reasoning:announce "high_notification"^^xsd:string).
```

After filling all the required information as in the UI (see Figure 12 and Figure 13), experimenters can click on the “save” button and store the rule in the FIESTA-IoT Reasoning database. Within FIESTA-IoT, by default all the created rules are public and available to all experimenters associated with FIESTA-IoT platform, hence all these rules can be re-used by other experimenters. When the rule is created successfully, the experimenter is redirected to the initial rule creation page, as shown in Figure 11.

5.3.3.1.2 Create new Rule when Non-Semantic expert

The FIESTA-IoT Reasoning tool also provides a simple UI for experimenters who are not familiar with semantics. To create a new rule, such experimenters would click on the “Create new rule – Non-Semantic Expert” button. This option is much easier when an experimenter does not have Semantic knowledge and wants to create new rules with the IF THEN logic (see Figure 14).

Create or edit a Rule - For Non-Semantic Expert

Name

Demo rule1

Description

Demo rule1

Select Quantity

Power

Select Sensor

-- Select Sensor --
https://platform.fiesta-iot.eu/iot-registry/api/resources/VsnDY_ipleAhy2eCc5jxNRqGyBVslwso2bO-8KCr7GKnfKLgda8TdXltkjaADUHLt
https://platform.fiesta-iot.eu/iot-registry/api/resources/5E2L0nnAsD75WlqCk2vqnHtsRwUk2331TjeHpQR7D_luqZKoFebr_8XTtixWgU
https://platform.fiesta-iot.eu/iot-registry/api/resources/Ur7Q-GLgxiLsfK4ZhXffEryue052DxDQzb8jxqkMPyLJZUITr-ZpAj1ZK_hi302o5gpi
https://platform.fiesta-iot.eu/iot-registry/api/resources/DYRIY6yuZg7izBNuQqbJf64IYV3IDcdGKVTLcndHjL5P7H3wKu_47BcAU00
<https://platform.fiesta-iot.eu/iot-registry/api/resources/tRRAK2IA6S5GEca2qPQD6hWzOn-kLp82OXHnXltm16LbPISitapxvtgEcrcPmWu>
<https://platform.fiesta-iot.eu/iot-registry/api/resources/x1AlxibeGRXJDPubYHcB9Wol22kDITEwzJR1t445JQfIPuv0YJivjsrb14DRkjp7mVv>
<https://platform.fiesta-iot.eu/iot-registry/api/resources/KwobGd67MC71etEJb3xTNYjk1LNIOFvdJJ1DOsMHlmV0ByWW35bZV3NYM5Bh>
https://platform.fiesta-iot.eu/iot-registry/api/resources/i6zudXsHdvXrTmJqDEM3Rr2QBF8x5823XGO_9AndMxLUGwQo4WkyGC8O6x
<https://platform.fiesta-iot.eu/iot-registry/api/resources/RXjeO4KJxS1R4MMYz3vDOrYLYH2MAf6kSXmomkuVmQn-Hs5HDWVpmvgNd>

Selected Sensor ID

https://platform.fiesta-iot.eu/iot-registry/api/resources/DYRIY6yuZg7izBNuQqbJf64IYV3IDcdGKVTLcndHjL5P7H3wKu_47BcAU00q1mkfNq_x8XzybC2wUo08-oE4XbkPWTXKOaJ6yyeah6OyePP3sB7fwlKYvO7gMbJHh

Latitude

51.243343

Longitude

-0.5932438

Quantity Kind

<http://purl.org/iot/vocab/m3-lite#Power>

Unit of Measurement

<http://purl.org/iot/vocab/m3-lite#Watt>

+ New Rule

IF

Power

>

1

Watt

THEN

high_notification

IF

Power

<

1

Watt

THEN

low_notification

Cancel

Save

Figure 14: Create new Rule - Non-Semantic Expert

An experimenter can click on the add-new-rule button “+ New Rule” to add a new rule or click on the remove icon “X” to remove it.

The FIESTA-IoT Reasoning tool will use the information added by the experimenter for the selected quantity kind, and the rule logic in order to generate a rule template by creating a SPARQL query as shown in Table 7:

Table 7: SPARQL query or a Rule

```
@prefix iot-lite: <http://purl.oclc.org/NET/UNIS/fiware/iot-lite#> .
@prefix m3-lite: <http://purl.org/iot/vocab/m3-lite#> .
@prefix ssn: <http://purl.oclc.org/NET/ssnx/ssn#> .
@prefix geo: <http://www.w3.org/2003/01/geo/wgs84_pos#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix dul: <http://www.loa.istc.cnr.it/ontologies/DUL.owl#> .
@prefix time: <http://www.w3.org/2006/time#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix reasoning: <https://fiesta-iot.eu/reasoning#> .
(?observation rdf:type ssn:Observation),
(?observation ssn:observedProperty ?observedProperty),
(?observedProperty rdf:type m3-lite:Power),
(?observation ssn:observationResult ?sensorOutput),
(?sensorOutput ssn:hasValue ?obsValue),
(?obsValue dul:hasDataValue ?dataValue),
(?obsValue iot-lite:hasUnit ?unit),
(?unit rdf:type m3-lite:Watt),
greaterThan(?dataValue, "1"^^xsd:double) -> (?observation reasoning:announce
"dangerous_notify"^^xsd:string).(?observation rdf:type ssn:Observation),
(?observation ssn:observedProperty ?observedProperty),
(?observedProperty rdf:type m3-lite:Power),
(?observation ssn:observationResult ?sensorOutput),
(?sensorOutput ssn:hasValue ?obsValue),
(?obsValue dul:hasDataValue ?dataValue),
(?obsValue iot-lite:hasUnit ?unit),
(?unit rdf:type m3-lite:Watt),
lessThan(?dataValue, "1"^^xsd:double) -> (?observation reasoning:announce
"lowpower_notify"^^xsd:string).
```

When an Experimenter clicks on the “Save” button, this rule will be stored in the FIESTA-IoT platform and then it will be public and re-usable by other experimenters.

5.3.3.1.3 Details of Rules

On the list of rules (see Figure 11) available on the FIESTA-IoT Reasoning, an experimenter can view (for example Rule 14 as shown in the Figure 15) the details of any rule by clicking on the “View” icon.

Rule 24

Name

rule 2

User Id

etragos

Content

```

@prefix iot-lite: <http://purl.oclc.org/NET/UNIS/fiware/iot-lite#> .
@prefix m3-lite: <http://purl.org/iot/vocab/m3-lite#> .
@prefix ssn: <http://purl.oclc.org/NET/ssnx/ssn#> .
@prefix geo: <http://www.w3.org/2003/01/geo/wgs84_pos#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix dul: <http://www.loa.istc.cnr.it/ontologies/DUL.owl#> .
@prefix time: <http://www.w3.org/2006/time#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix reasoning: <https://fiesta-iot.eu/reasoning#> .(?observation rdf:type ssn:Observation),
(?observation ssn:observedProperty ?observedProperty),
(?observedProperty rdf:type m3-lite:Power),
(?observation ssn:observationResult ?sensorOutput),
(?sensorOutput ssn:hasValue ?obsValue),
(?obsValue dul:hasDataValue ?dataValue),
(?obsValue iot-lite:hasUnit ?unit),
(?unit rdf:type m3-lite:Watt),
lessThan(?dataValue, "5"^^xsd:double) -> (?observation reasoning:announce "HIGH_power"^^xsd:string).

```

Sensor

https://platform.fiesta-iot.eu/iot-registry/api/resources/x1AxiBeGRXJDPUbYHcB9Voi22kDiTEwzJR1t445JQfIPuv0YJivjsrb14DRkpj7mVw5_Ax4eVEsDr1PMu0AJxoj0uQFEZh743kKon7QVRc-DmsGDO9E6fxBK6Oc9pd

Description

this is a description

Back

Edit

Figure 15: Example of Rule details

5.3.3.1.4 Edit a Rule

The function for editing a rule is available only to those experimenters that have created the particular rule. This means, an experimenter is not allowed to change a rule created by other experimenters for security purposes.

On the screen showing the list of rules (see Figure 11) or on the rule details screen (see Figure 15), when an experimenter clicks on the “Edit” button, the screen for editing rules will be shown as in Figure 16 and Figure 17 (Note that in the Figure 17 an experimenter can edit the rule in the provided textbox):

Create or edit a Rule - For Semantic Expert

ID

24

Name

rule 2

Description

this is a description

Select Quantity

Power

Select Sensor

https://platform.fiesta-iot.eu/iot-registry/api/resources/VsnDY_ipleAhy2eCc5jxNRqGyBVslwso2bO-8KCr7GKnfKLgda8TdXltkjaADUHLt
https://platform.fiesta-iot.eu/iot-registry/api/resources/5E2L0nnAsD75WlqCk2vqnHtsRxxUk2331TjeHpQR7D_luqZKoFebr_8XTtixWgU
https://platform.fiesta-iot.eu/iot-registry/api/resources/Ur7Q-GLgxiLsfK4ZhXffEryue052DxDQzb8jxqKMPyLJZUiTr-ZpAj1ZK_hi302o5gp
https://platform.fiesta-iot.eu/iot-registry/api/resources/DYRiY6yuZg7izBNuQqbjfq64IYV3IDcdGKVTyLCndHljl5P7H3wKu_47BcAUO0q1
<https://platform.fiesta-iot.eu/iot-registry/api/resources/tRRAK2IA6S5GEca2qPQD6hWzOn-kLp82OXHnXltm16LbPISitapxvtgEcrrPmWu>
<https://platform.fiesta-iot.eu/iot-registry/api/resources/x1AlxibeGRXJDPubYHcB9Wol22kDiEwzJR1t445JQflPuv0YJivjsrb14DRkpj7mVv>
<https://platform.fiesta-iot.eu/iot-registry/api/resources/KwobGd67MC71etEJb3xTNYjk1LNIOFvdJJ1DOsMHlmV0ByWW35bZV3NYM5Bt>
https://platform.fiesta-iot.eu/iot-registry/api/resources/i6zudXsHdvXrTmJqDEM3Rr2QBF8x5823XGO_9AndMxLUGwQo4WkyGC8O6x
<https://platform.fiesta-iot.eu/iot-registry/api/resources/RXjeO4KJxS1R4MMYz3vDOrYLYH2MAf6kSXmomkuVmQn-Hs5HDWVpvmvgNd>
https://platform.fiesta-iot.eu/iot-registry/api/resources/PMFY3CfeI3cuMxIbrYXnfZziaQYS0rAPDLV/MnQel53uE_2tPYZIkEYpE_73DnEi

Selected Sensor ID

https://platform.fiesta-iot.eu/iot-registry/api/resources/x1AlxibeGRXJDPubYHcB9Wol22kDiEwzJR1t445JQflPuv0YJivjsrb14DRkpj7mVv5_Ax4eVEsDr1PMu0AJxoj0uQFEZh743kKon7QVRc-DmsGDO9E6fxBK6Oc9pd

Latitude

51.243343

Longitude

-0.5932438

Quantity Kind

<http://purl.org/iot/vocab/m3-lite#Power>

Unit of Measurement

<http://purl.org/iot/vocab/m3-lite#Watt>

Figure 16: Edit Rule Information

```

    },
    "observationSamplingTime" : {
      "@id" : "http://purl.oclc.org/NET/ssnx/ssn#observationSamplingTime",
      "@type" : "@id"
    },
    "observedBy" : {
      "@id" : "http://purl.oclc.org/NET/ssnx/ssn#observedBy",
      "@type" : "@id"
    },
    "observedProperty" : {
      "@id" : "http://purl.oclc.org/NET/ssnx/ssn#observedProperty",
      "@type" : "@id"
    },
    "location" : {
      "@id" : "http://www.w3.org/2003/01/geo/wgs84_pos#location",
      "@type" : "@id"
    }
  }
}

```

Rule content

```

@prefix iot-lite: <http://purl.oclc.org/NET/UNIS/fiware/iot-lite#> .
@prefix m3-lite: <http://purl.org/iot/vocab/m3-lite#> .
@prefix ssn: <http://purl.oclc.org/NET/ssnx/ssn#> .
@prefix geo: <http://www.w3.org/2003/01/geo/wgs84_pos#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix dul: <http://www.loa.istc.cnr.it/ontologies/DUL.owl#> .
@prefix time: <http://www.w3.org/2006/time#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix reasoning: <https://fiesta-iot.eu/reasoning#> .(?observation rdf:type ssn:Observation),
(?observation ssn:observedProperty ?observedProperty),
(?observedProperty rdf:type m3-lite:Power),
(?observation ssn:observationResult ?sensorOutput),
(?sensorOutput ssn:hasValue ?obsValue),
(?obsValue dul:hasDataValue ?dataValue),
(?obsValue iot-lite:hasUnit ?unit),
(?unit rdf:type m3-lite:Watt),
lessThan(?dataValue, "5"^^xsd:double) -> (?observation reasoning:announce "HIGH_power"^^xsd:string).

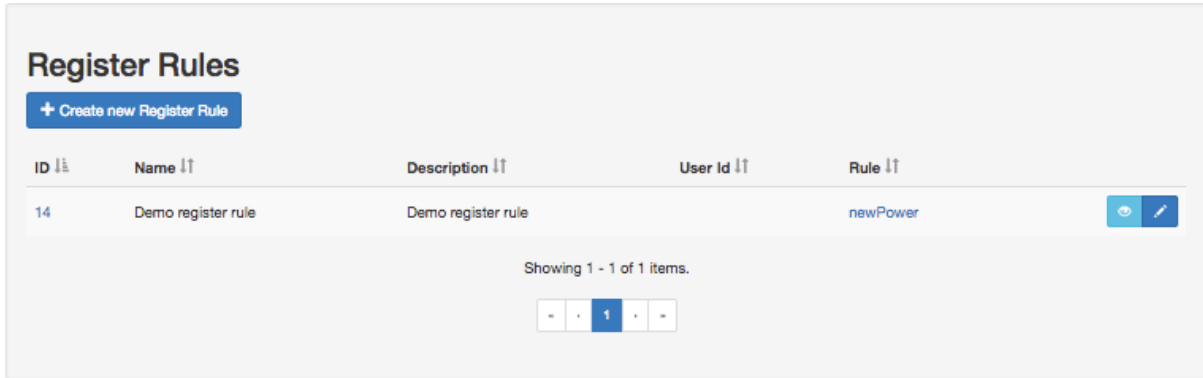
```

Cancel

Save

Figure 17: Edit Rule content**5.3.3.2 Rule Registration**

After creating the rule template, an experimenter needs to first register the rule on a selected sensor before executing it. This can be done through the “Reasoning” menu on the portal by selecting the “Register Rule: sub menu. The following Figure 18 is shown:



The screenshot shows a web interface titled "Register Rules". At the top left is a blue button labeled "+ Create new Register Rule". Below this is a table with the following columns: "ID", "Name", "Description", "User Id", and "Rule". The table contains one row with the values: "14", "Demo register rule", "Demo register rule", and "newPower". To the right of the "Rule" column is a small blue icon. Below the table, it says "Showing 1 - 1 of 1 items." and there is a pagination control showing "1" in a blue box.

ID	Name	Description	User Id	Rule
14	Demo register rule	Demo register rule		newPower

Showing 1 - 1 of 1 items.

Figure 18: Rule Registration home

For security/privacy reasons, each experimenter can only see his own registered rules and not those of other experimenters.

5.3.3.2.1 Register a rule

When an experimenter clicks on the “+ Create new Register Rule” button, the Figure 19 is shown, where the experimenter can add information, such as the description of the registered rule, the quantity kind and the sensor upon which the rule will be executed, and also select the rule template to be used for this registration:

Create or edit a Register Rule

Name

Demo register rule one

Description

Demo on register rule one

Rule

✓ -- Select Rule --

- Demo rule 1
- test1
- Myrule1
- Myrule2
- test22
- Demo register one
- newPower

This field is required.

Select Quantity

-- Select quantity kind --

Select Sensor

-- Select Sensor --

Cancel Save

Figure 19: Register Rule- Available Rules

As Figure 19 shows, an experimenter can select the rule template from the dropdown menu that shows all the created rules on the platform. By selecting one rule, its detailed information is shown in the “Rule content” field, as shown in Figure 20.

Create or edit a Register Rule

Name

Demo register rule one

Description

Demo on register rule one

Rule

newPower

Rule Content

```

@prefix iot-lite: <http://purl.oclc.org/NET/UNIS/fiware/iot-lite#> .
@prefix m3-lite: <http://purl.org/iot/vocab/m3-lite#> .
@prefix ssn: <http://purl.oclc.org/NET/ssnx/ssn#> .
@prefix geo: <http://www.w3.org/2003/01/geo/wgs84_pos#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix dul: <http://www.loa.istc.cnr.it/ontologies/DULowl#> .
@prefix time: <http://www.w3.org/2006/time#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix reasoning: <https://fiesta-iot.eu/reasoning#> .(?observation rdf:type ssn:Observation),
(?observation ssn:observedProperty ?observedProperty),
(?observedProperty rdf:type m3-lite:Power),
(?observation ssn:observationResult ?sensorOutput),
(?sensorOutput ssn:hasValue ?obsValue),
(?obsValue dul:hasDataValue ?dataValue),
(?obsValue iot-lite:hasUnit ?unit),
(?unit rdf:type m3-lite:Watt),
greaterThan(?dataValue, "0.1"^^xsd:double) -> (?observation reasoning:announce "notify_high"^^xsd:string).

```

Select Quantity

Power

Select Sensor

Figure 20: Register Rule - Detail Rule content

After selecting the rule template, the next step for the experimenter is to select the sensor ID to register (the quantity is pre-filled according to the rule information) as shown in Figure 21.

Create or edit a Register Rule

Name

New registration1

Description

New registration1

Rule

rule1

Rule Content

```

@prefix iot-lite: <http://purl.oclc.org/NET/UNIS/fiware/iot-lite#> .
@prefix m3-lite: <http://purl.org/iot/vocab/m3-lite#> .
@prefix ssn: <http://purl.oclc.org/NET/ssnx/ssn#> .
@prefix geo: <http://www.w3.org/2003/01/geo/wgs84_pos#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix dul: <http://www.loa.istc.cnr.it/ontologies/DUL.owl#> .
@prefix time: <http://www.w3.org/2006/time#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix reasoning: <https://fiesta-iot.eu/reasoning#> .(?observation rdf:type ssn:Observation),
(?observation ssn:observedProperty ?observedProperty),
(?observedProperty rdf:type m3-lite:Power),
(?observation ssn:observationResult ?sensorOutput),
(?sensorOutput ssn:hasValue ?obsValue),
(?obsValue dul:hasDataValue ?dataValue),
(?obsValue iot-lite:hasUnit ?unit),
(?unit rdf:type m3-lite:Watt),
lessThan(?dataValue, "5"^^xsd:double) -> (?observation reasoning:announce "HIGH"^^xsd:string).

```

Select Quantity

Power

Select Sensor

-- Select Sensor --

https://platform.fiesta-iot.eu/iot-registry/api/resources/VsnDY_ipleAhy2eCc5jxNRqGyBVslwso2bo-8KCr7GKnfKLgda8TdXitkjaADUHLt
https://platform.fiesta-iot.eu/iot-registry/api/resources/5E2L0nnAsD75WlqCk2vqnHtsRxwUk2331TjeHpQR7D_luqZKoFebr_8XTtixWgU
https://platform.fiesta-iot.eu/iot-registry/api/resources/Ur7Q-GLgxiLsfK4ZhXffEryue052DxDQzb8jxqKMPyLJZUitr-ZpAj1ZK_hi302o5gpi
https://platform.fiesta-iot.eu/iot-registry/api/resources/DYRIY6yuZg7izBNuQqbJfq64IYV3IDcdGKVtyLCndHljL5P7H3wKu_47BcAUO0q1
<https://platform.fiesta-iot.eu/iot-registry/api/resources/tRRAK2IA6S5GEca2qPQD6hWzOn-kLp82OXHnXltm16LbPISitapxvtgEcrrPmWu>
<https://platform.fiesta-iot.eu/iot-registry/api/resources/x1AlxibeGRXJDPubYHcB9Wol22kDiEwzjR1t445JQfIPuv0YJivjsrb14DRkpj7mVv>
<https://platform.fiesta-iot.eu/iot-registry/api/resources/KwobGd67MC71etEJb3xTNYjk1LNIOFvdJJ1DOsMHlmV0ByWW35bZV3NYM5Bt>
https://platform.fiesta-iot.eu/iot-registry/api/resources/i6zudXsHdvXrTmJqDEM3Rr2QBF8x5823XGO_9AndMxLUGwQo4WkyGC8O6x
<https://platform.fiesta-iot.eu/iot-registry/api/resources/RXjeO4KJxS1R4MMMyZ3vDOrYlHyH2MAf6kSXmomkuVmQn-Hs5HDWVpmvgNd>

Cancel

Save

Figure 21: Register Rule - Select Sensor

After filling the required information on the form and clicking the “Save” button, the rule registration functionality is finished and the new rule is registered and available for execution.

5.3.3.2.2 Detail Rule registration

Another functionality on the initial screen that lists the existing rule registrations (as shown in Figure 11) is to see the details of a registered rule, by clicking on the “detail” icon (as shown in Figure 22).

Register Rule 20

Name

reg1

Description

Rule Content

```

@prefix iot-lite: <http://purl.oclc.org/NET/UNIS/fiware/iot-lite#> .
@prefix m3-lite: <http://purl.org/iot/vocab/m3-lite#> .
@prefix ssn: <http://purl.oclc.org/NET/ssnx/ssn#> .
@prefix geo: <http://www.w3.org/2003/01/geo/wgs84_pos#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix dul: <http://www.loa.istc.cnr.it/ontologies/DUL.owl#> .
@prefix time: <http://www.w3.org/2006/time#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix reasoning: <https://fiesta-iot.eu/reasoning#> .(?observation rdf:type ssn:Observation),
(?observation ssn:observedProperty ?observedProperty),
(?observedProperty rdf:type m3-lite:Power),
(?observation ssn:observationResult ?sensorOutput),
(?sensorOutput ssn:hasValue ?obsValue),
(?obsValue dul:hasDataValue ?dataValue),
(?obsValue iot-lite:hasUnit ?unit),
(?unit rdf:type m3-lite:Watt),
lessThan(?dataValue, "5"^^xsd:double) -> (?observation reasoning:announce "HIGH"^^xsd:string).

```

Sensor

https://platform.fiesta-iot.eu/iot-registry/api/resources
/KwobGd67MC71etEJb3xTNYjk1LNIOFVdJJ1DOsMHimV0ByWW35bZV3NYM5Bks5UIWu7m-
KY1HkfohcaSmDFMQhlezplalhuueGuysFIFShPafeda76yOdCQTvgIsQzuL

User Id

etragos

Reasoning

Back

Edit

Figure 22: Register Rule – detailed information

5.3.3.2.3 Edit a Rule registration

When experimenters want to edit a rule registration, they can click on the “Edit” button on the detail rule registration page or on the “edit” icon on the list of rule registrations screen. Then, the following is shown (see Figure 23):

Create or edit a Register Rule

ID

20

Name

reg1

Description

Rule

rule1

Rule Content

```
@prefix iot-lite: <http://purl.oclc.org/NET/UNIS/fiware/iot-lite#> .
@prefix m3-lite: <http://purl.org/iot/vocab/m3-lite#> .
@prefix ssn: <http://purl.oclc.org/NET/ssnx/ssn#> .
```

Select Quantity

Power

Select Sensor

-- Select Sensor --

https://platform.fiesta-iot.eu/iot-registry/api/resources/VsnDY_ipleAhy2eCc5jxNRqGyBVslwso2bO-8KCr7GKnfKlgda8TdXltkjaADUHLt
https://platform.fiesta-iot.eu/iot-registry/api/resources/5E2L0nnAsD75WlqCk2vqnHtsRxxUk2331TjeHpQR7D_luqZKoFebr_8XTtixWgU
https://platform.fiesta-iot.eu/iot-registry/api/resources/Ur7Q-GLgxiLsfK4ZhXffEryue052DxDQzb8jxqKMPyLJZUiTr-ZpAj1ZK_hi302o5gp1
https://platform.fiesta-iot.eu/iot-registry/api/resources/DYRIY6yuZg7izBNuQqbJf64IYV3lDcdGKVTyLCndHijL5P7H3wKu_47BcAUO0q1
<https://platform.fiesta-iot.eu/iot-registry/api/resources/tRRAK2IA6S5GEca2qPQD6hWzOn-kLp82OXHnXltm16LbPISitapxvtgEcrrPmV>
<https://platform.fiesta-iot.eu/iot-registry/api/resources/x1AlxibeGRXJDPubYHcB9Wol22kDitEwzJR1t445JQfIPuv0YJivjsrb14DRkpj7mVv>
<https://platform.fiesta-iot.eu/iot-registry/api/resources/KwobGd67MC71etEJb3xTNYjk1LNIOFvdJJ1DOsMHImV0ByWW35bZV3NYM5Bt>
https://platform.fiesta-iot.eu/iot-registry/api/resources/i6zudXsHdvXrTmJqDEM3Rr2QBFb8x5823XGO_9AndMxLUGwQo4WkyGC8O6x
<https://platform.fiesta-iot.eu/iot-registry/api/resources/RXjeO4KJxS1R4MMYz3vDOrYLYH2MAf6kSXmomkuVmQn-Hs5HDWVpmvgNd>

Selected Sensor ID

https://platform.fiesta-iot.eu/iot-registry/api/resources/tRRAK2IA6S5GEca2qPQD6hWzOn-kLp82OXHnXltm16LbPISitapxvtgEcrrPmVuDg-vqcW8xUTwryJ13_jt-l01DzPKZA6v1VYA_UVRj7ihfGV9LONi8Tm0Ccv3rzBXR

```
{
  "@graph" : [ {
    "@id" : "https://platform.fiesta-iot.eu/iot-registry/api/observations/2dVJWfexlgnTnPLdhpRxxLr_tmqlhFRKNeUaQJw9tFVng1T
X2HjdKw3_tq4hxrUaldSkqNX07M0YtU8REq-w_7H0xR_9N7jRzdtqBCu-npFcpQh2oJzzUF73xkEIQT",
    "@type" : "http://www.w3.org/2006/time#Instant",
    "inXSDDateTime" : "2017-11-24T15:17:00Z"
```

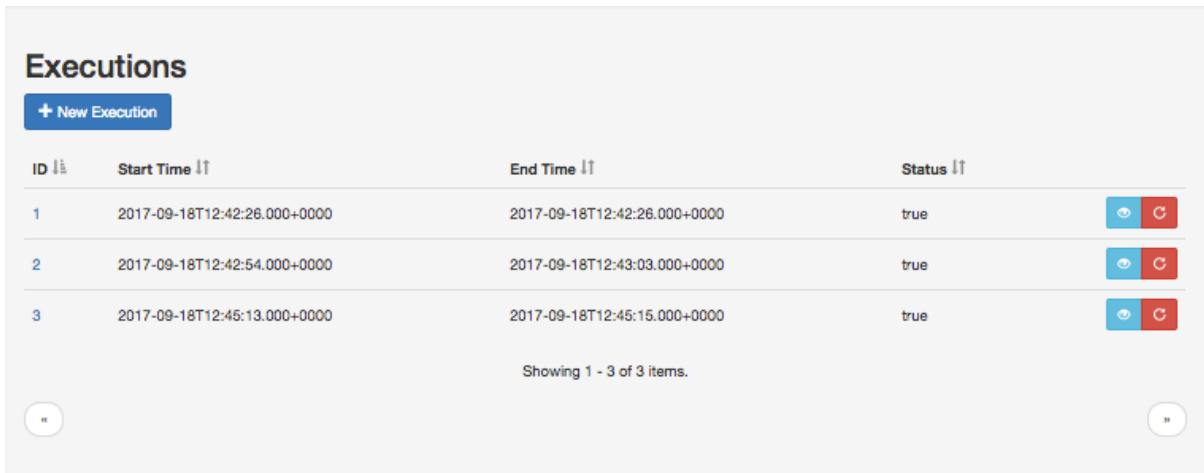
Figure 23: User Interface for editing a rule registration

The Experimenter can then edit the details of the registered rule, i.e. name, description, select new rule, select another sensor and then click “Save” to update all information on the FIESTA-IoT platform.

5.3.3.3 Rule Execution

The final step after creating and registering a rule is to execute it. The FIESTA-IoT platform provides three main functions for creating a “New execution”, performing a “Re-execution” and viewing the details of an execution.

A Rule execution is the function where the registered rule is executed upon the input sensor data, in order to create some inference data. The home screen of rule execution is shown in Figure 24



Executions

[+ New Execution](#)

ID	Start Time	End Time	Status	
1	2017-09-18T12:42:26.000+0000	2017-09-18T12:42:26.000+0000	true	View Refresh
2	2017-09-18T12:42:54.000+0000	2017-09-18T12:43:03.000+0000	true	View Refresh
3	2017-09-18T12:45:13.000+0000	2017-09-18T12:45:15.000+0000	true	View Refresh

Showing 1 - 3 of 3 items.

Figure 24: Rule Execution Home page

5.3.3.3.1 Create a New Execution

When an experimenter clicks on the “+ New Execution” button, the following form is shown (see Figure 25):

New Execution

Register Rule

reg1

Sensor

https://platform.fiesta-iot.eu/iot-registry/api/resources
/KwobGd67MC71etEJb3xTNYjk1LNIOFVdJJ1DOsMHImV0ByWW35bZV3NYM5BkS5UIWu7m-
KY1HkfohcaSmDFMQhlezplaihuueGuysFIFShPafeda76yOdCQTVglsQzuL

Rule Content

```
@prefix iot-lite: <http://purl.oclc.org/NET/UNIS/fiware/iot-lite#> .
@prefix m3-lite: <http://purl.org/iot/vocab/m3-lite#> .
@prefix ssn: <http://purl.oclc.org/NET/ssnx/ssn#> .
@prefix geo: <http://www.w3.org/2003/01/geo/wgs84_pos#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix dul: <http://www.loa.istc.cnr.it/ontologies/DUL.owl#> .
@prefix time: <http://www.w3.org/2006/time#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix reasoning: <https://fiesta-iot.eu/reasoning#> .(?observation rdf:type ssn:Observation),
(?observation ssn:observedProperty ?observedProperty),
(?observedProperty rdf:type m3-lite:Power),
(?observation ssn:observationResult ?sensorOutput),
(?sensorOutput ssn:hasValue ?obsValue),
(?obsValue dul:hasDataValue ?dataValue),
(?obsValue iot-lite:hasUnit ?unit),
(?unit rdf:type m3-lite:Watt),
lessThan(?dataValue, "5"^^xsd:double) -> (?observation reasoning:announce "HIGH"^^xsd:string).
```

Select Time For Execution

-- Select Execution Type --

-- Select Execution Type --

Current

Range

Figure 25: User Interface for creating a new Rule execution

In this form, the Experimenter can create a new execution, by selecting a registered rule and setting the “Time for execution”, which can be either in the current measurement or in the measurements within a time range.

5.3.3.3.2 New execution with current time

This rule execution happens when the experimenter selects the “Current” option and clicks on the “Save” button. Then, the FIESTA-IoT Reasoning module will execute this registered rule (sensor, rule), giving the result of the execution, which can be either “true” (success) or “false”, together with other details, such as the start, end time, sensor id, rule content, original data, inference data, and full data.

5.3.3.3.3 New execution with period or range of time

When an experimenter selects the “Range” execute option, he will be able to select the starting and ending date of the measurements to be considered in this rule, as shown in Figure 26.

New Execution

Register Rule

reg1

Sensor

https://platform.fiesta-iot.eu/iot-registry/api/resources
/KwobGd67MC71etEJb3xTNYjk1LNIOFVdJJ1DOsMHImV0ByWW35bZV3NYM5Bks5UIWu7m-
KY1HkfohcaSmDFMQhlezplalhuueGuysFIFShPafeda76yOdCQTvgIsQzuL

Rule Content

```
@prefix iot-lite: <http://purl.oclc.org/NET/UNIS/fiware/iot-lite#> .
@prefix m3-lite: <http://purl.org/iot/vocab/m3-lite#> .
@prefix ssn: <http://purl.oclc.org/NET/ssnx/ssn#> .
@prefix geo: <http://www.w3.org/2003/01/geo/wgs84_pos#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix dul: <http://www.loa.istc.cnr.it/ontologies/DUL.owl#> .
@prefix time: <http://www.w3.org/2006/time#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix reasoning: <https://fiesta-iot.eu/reasoning#> .
(?observation rdfs:type ssn:Observation),
(?observation ssn:observedProperty ?observedProperty),
(?observedProperty rdfs:type m3-lite:Power),
(?observation ssn:observationResult ?sensorOutput),
(?sensorOutput ssn:hasValue ?obsValue),
(?obsValue dul:hasDataValue ?dataValue),
(?obsValue iot-lite:hasUnit ?unit),
(?unit rdfs:type m3-lite:Watt),
lessThan(?dataValue, "5"^^xsd:double) -> (?observation reasoning:announce "HIGH"^^xsd:string).
```

Select Time For Execution

Range

Start

2017-11-13 15:18

End

2017-11-14 00:00

Cancel Save

Figure 26: Execute Rule on sensor base on specific time

The FIESTA-IoT Reasoning will execute a SPARQL query to retrieve sensor data as shown in Table 8:

Table 8: SPARQL Query

```

PREFIX iot-lite: <http://purl.oclc.org/NET/UNIS/fiware/iot-lite#>
PREFIX m3-lite: <http://purl.org/iot/vocab/m3-lite#>
PREFIX ssn: <http://purl.oclc.org/NET/ssnx/ssn#>
PREFIX geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX dul: <http://www.loa.istc.cnr.it/ontologies/DUL.owl#>
PREFIX time: <http://www.w3.org/2006/time#>

SELECT      ?sensingDevice  ?dataValue    ?dateTime    ?observation
?sensorOutput ?obsValue ?instant
WHERE {
  ?observation ssn:observedBy ?sensingDevice .
  VALUES ?sensingDevice {
    <https://platform.fiesta-iot.eu/iot-registry/api/resources/Ur7Q-
    GLgxiLsfK4ZhXffEryue052DxDQzb8jxqKMPyLJZUiTr-
    ZpAj1ZK_hi302o5gp8V6Fe1a2jEzg_STnJkUCQHp8f7qAg1DiohqUnfc1l3289Lvfcu
    RmXiDPfZR0l>} .
  ?observation ssn:observationResult ?sensorOutput .
  ?sensorOutput ssn:hasValue ?obsValue .
  ?obsValue dul:hasDataValue ?dataValue .
  ?observation ssn:observationSamplingTime ?instant .
  ?instant time:inXSDDateTime ?dateTime .
    FILTER (
      (xsd:dateTime(?dateTime)      >      xsd:dateTime("2017-09-
16T23:00:00Z"))
      &&      (xsd:dateTime(?dateTime)      <      xsd:dateTime("2017-09-
17T23:00:00Z"))
    ) .
}ORDER BY ?sensingDevice ASC(?dateTime)

```

5.3.3.3.4 Re-Execution

When an experimenter wants to repeat an execution of the rule, he can just click on the “Re-execute” button on the list of executions. Then, a similar form as with the rule execution will be shown (see Figure 27) and the user will be allowed to select if he wants to re-execute the rule on the current measurement or on a range of measurements.

New Execution

ID

28

Register Rule

-- Select Rule Registered --

This field is required.

Sensor

https://platform.fiesta-iot.eu/iot-registry/api/resources
/KwobGd67MC71etEJb3xTNYjk1LNIOFVdJJ1DOsMHImV0ByWW35bZV3NYM5BkS5UIWu7m-
KY1HkfohcaSmDFMQhlezplahuueGuysFIFShPafeda76yOdCQTvglsQzuL

Rule Content

```
@prefix iot-lite: <http://purl.oclc.org/NET/UNIS/fiware/iot-lite#> .
@prefix m3-lite: <http://purl.org/iot/vocab/m3-lite#> .
@prefix ssn: <http://purl.oclc.org/NET/ssnx/ssn#> .
@prefix geo: <http://www.w3.org/2003/01/geo/wgs84_pos#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix dul: <http://www.loa.istc.cnr.it/ontologies/DUL.owl#> .
@prefix time: <http://www.w3.org/2006/time#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix reasoning: <https://fiesta-iot.eu/reasoning#> .(?observation rdf:type ssn:Observation),
(?observation ssn:observedProperty ?observedProperty),
(?observedProperty rdf:type m3-lite:Power),
(?observation ssn:observationResult ?sensorOutput),
(?sensorOutput ssn:hasValue ?obsValue),
(?obsValue dul:hasDataValue ?dataValue),
(?obsValue iot-lite:hasUnit ?unit),
(?unit rdf:type m3-lite:Watt),
lessThan(?dataValue, "5"^^xsd:double) -> (?observation reasoning:announce "HIGH"^^xsd:string).
```

Select Time For Execution

-- Select Execution Type --

-- Select Execution Type --

Current

Range

Figure 27: Re-Execute Rule

5.3.4 FIESTA-IoT Acquisition Toolkit

Other than being a web service, the FAT can also be accessed through a web UI. This interface allows an experimenter to interact with the FAT toolkit visually for single experiments. The page mainly consists of three tabs (see Figure 28). The first being the “Input”. This allows the user to provide the SPARQL query for the dataset and the methods/parameters to apply on it (see Figure 29). Once this is submitted the result is displayed in the “Result” tab (see Figure 30). A plot for certain methods will be provided in the third tab, which is under development.

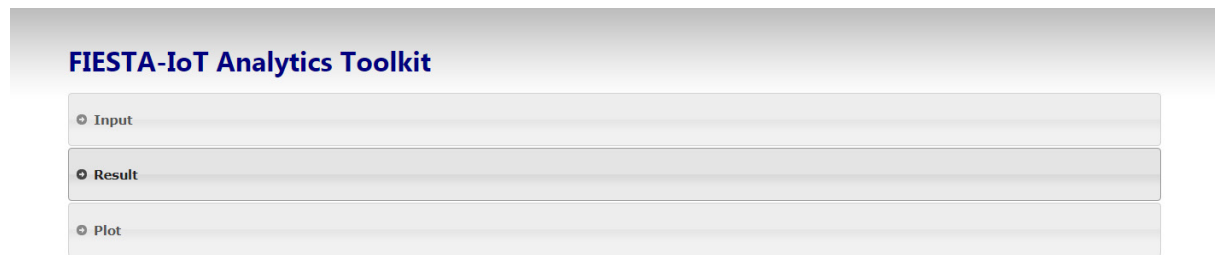


Figure 28: Analytics Toolkit Tabs

Welcome

- 1) Input the SPARQL query
- 2) Input the required methods and parameter
- 3) Result will be presented in table below
- 4) A plot will be available based on final method applied

SPARQL Query

```
PREFIX m3-lite: <http://purl.org/iot/vocab/m3-lite#>
PREFIX ssn: <http://purl.oclc.org/NET/ssnx/ssn#>
PREFIX geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX dul: <http://www.loa.istc.cnr.it/ontologies/DUL.owl#>
PREFIX time: <http://www.w3.org/2006/time#>

SELECT ?sensingDevice ?dataValue ?dateTime
WHERE {
  ?sensingDevice a m3-lite:EnergyMeter .
  ?sensingDevice iot-lite:hasQuantityKind ?qk .
  ?qk a m3-lite:Power .
  ?sensingDevice iot-lite:hasUnit ?unit .
  ?unit a m3-lite:Watt .
  ?sensingDevice iot-lite:isSubSystemOf ?device .
  ?device a ssn:Device .
  ?device ssn:onPlatform ?platform .
  ?platform geo:location ?point .
  ?point geo:lat ?lat .
  ?point geo:long ?long .
  ?observation ssn:observedBy ?sensingDevice .
  ?observation ssn:observationResult ?sensorOutput .
  ?sensorOutput ssn:hasValue ?obsValue .
  ?obsValue dul:hasDataValue ?dataValue .
```

Methods and Parameters

Method: Threshold

Method:

Figure 29: Analytics Input Tab

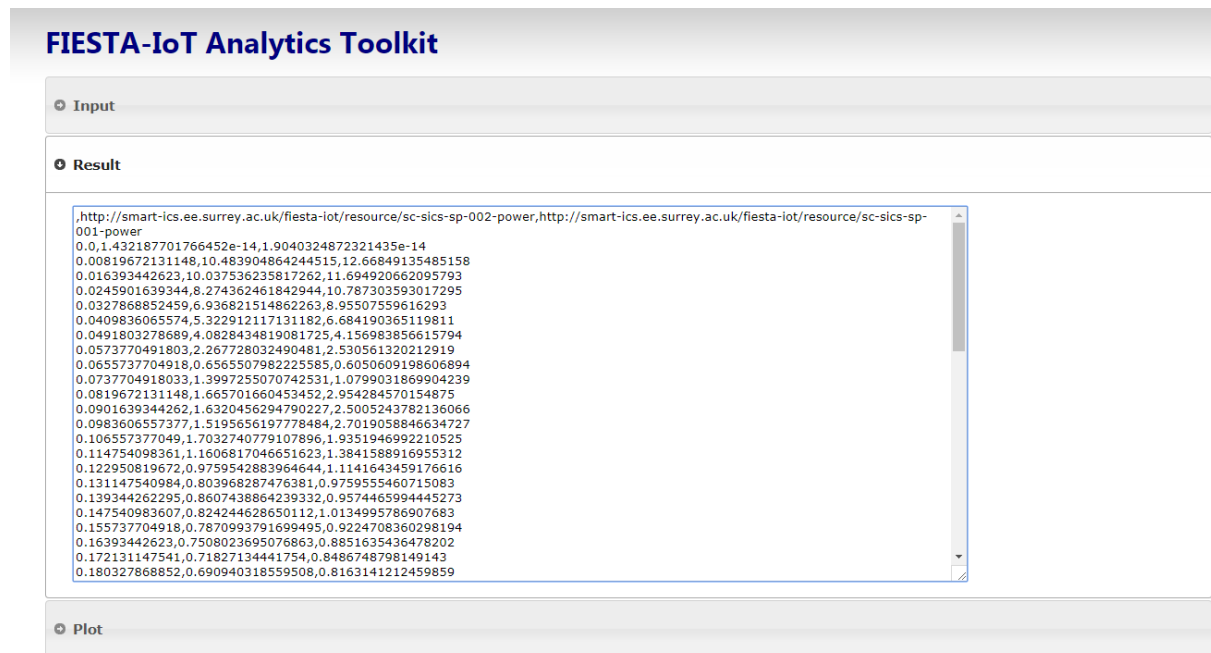


Figure 30: Analytics Toolkit Result

5.3.5 Experiment/Testbed Monitoring Tool

The Testbed Monitoring Tool is intended to provide FIESTA-IoT users with information about the data that is sent by testbeds and can be used by experimenters via the FIESTA-IoT portal.

5.3.5.1 UI Specification

The Testbed Monitoring is embedded in the portal as an iframe and can be used as every other component of the portal.

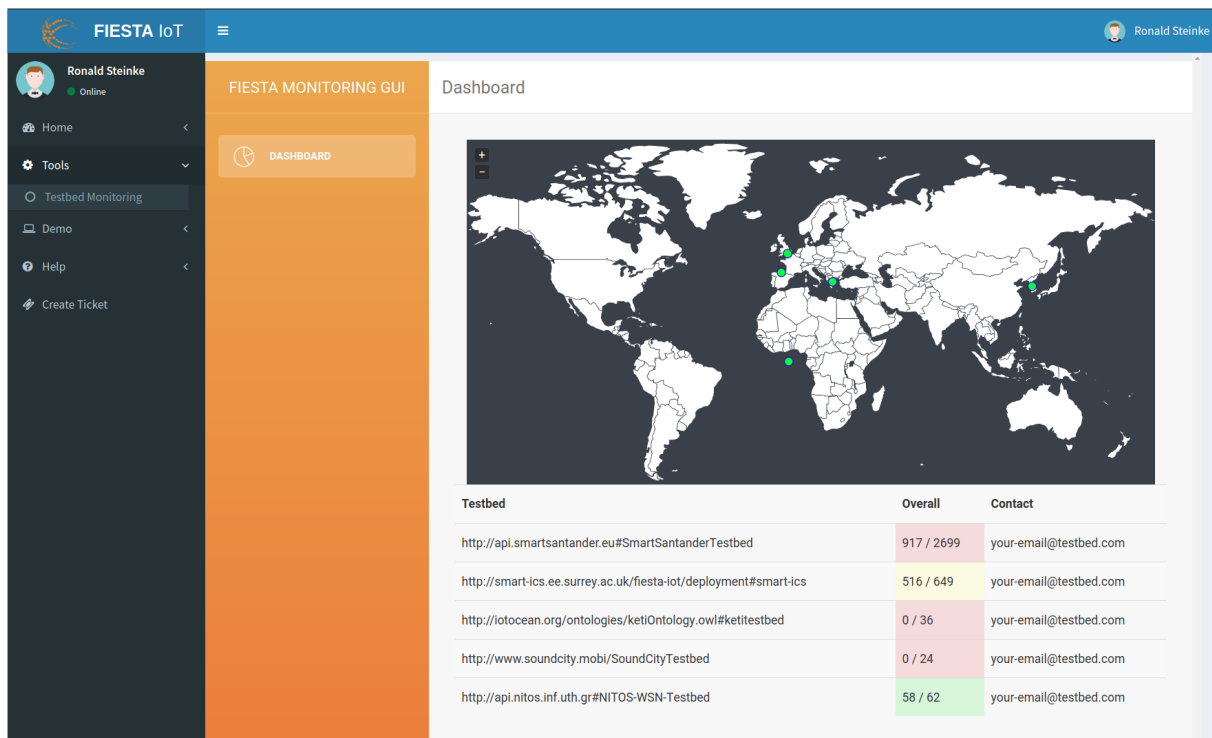


Figure 31: The Testbed Monitoring Tool embedded in the FIESTA-IoT portal

As seen in Figure 31 the Testbed Monitoring can be found under the Tools section as Testbed monitoring.

Its start page is the overview of all monitored testbeds. Here, the locations of the testbeds can be seen in a map and a table, which lists their name, the number of active sensors and total number of sensors, and the contact for each testbed. Here the total number of sensors means all registered sensors in the FIESTA-IoT platform belonging to this testbed and active sensors are resources that have an observation in the last 24 hours.

By clicking on one of a testbed, a detailed view lists all its underlying sensors and their locations in the map. For every sensor, the internal ID of the sensor used by the Monitoring Tool, the quantity kind, the last observation, the unit and the location are listed. The quantity kind and unit are using the m3-lite Taxonomy [13]. If a sensor is clicked, a modal view pops up, and shows a graph of the latest observations of this sensor and the sensor ID that is used by interacting with the IoT-Registry.

FIESTA-IoT admins can open the settings view where testbeds can be enabled for showing in the UI or disabled again. The Monitoring Tool provides a notification system. This can be used to receive a notification mail, when a testbed reaches a predefined state, e.g., the number of active sensors reaches a threshold. Experimenters can use this if they find a testbed with problems and want to get informed when it is ready to be used again. By the time writing this deliverable, this function was not yet implemented.

More detailed information about the usage of the Monitoring Tool can be found in Deliverable 3.6 [6].

5.3.5.2 Implementation

The system is integrated into the platform as an additional component.

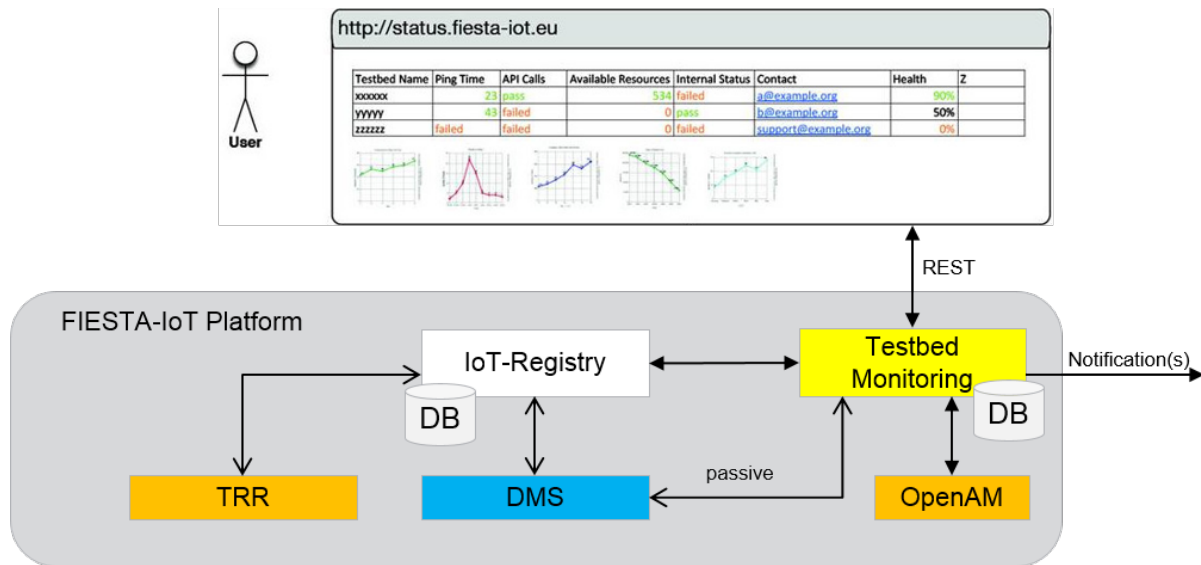


Figure 32: Testbed Monitoring component in the FIESTA-IoT Platform

As seen in Figure 32, the monitoring component connects to the Mongo DB and uses it for storing its data. It is also connected to the IoT-Registry in order to querying data of the platform. In addition, the monitoring tool asks the OpenAM service to use the information of every logged-in user to get the role of it and adjust the view.

5.3.5.2.1 General Procedure

The general procedure of the Monitoring tool is the following. In the beginning the configuration is read and the components like tasks, the database connection and the webserver are properly configured.

In the initialisation phase the background tasks will be started. These tasks are mainly to query the IoT-Registry for the relevant information like testbeds, resources and observations. In the future, tasks regarding analysis will be started in the beginning. Afterwards the Flask server is started for serving the GUI and the API.

5.3.5.2.2 Bootstrapping

The bootstrapping is done by the tasks that are related to the IoT-Registry. In this phase, the database will be cleaned when it is configured to do so. If not, the database will be searched for the latest observation time in order to properly set up the query for following observations. The tasks for updating testbeds and resources information will be activated. They will retrieve all relevant data from the IoT-Registry and store it into the database. The tasks are configured to be run on an interval base. When this is done, the task for querying observations is done. This task will use either the latest stored observation or a pre-configured time span in order to start the querying for observations. The retrieval of observations is done in smaller steps until it reaches the actual time and starts normal interval based updates. After this initial bootstrap, the

database is filled with the initial data and tasks like analysis and the webserver will be started.

5.3.5.2.3 Querying the IoT-Registry

As the Monitoring tool is deployed on the same machine as the IoT-Registry, for retrieving the testbeds information, the IoT-Registry API are used to directly retrieve the testbed names and IRIs (Internationalized Resource Identifier).

The gathering of all resources and observations is done via executing SPARQL queries. An example query to retrieve all sensors is provided in Table 9

Table 9: SPARQL Query

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX ssn: <http://purl.oclc.org/NET/ssnx/ssn#>
PREFIX iot-lite: <http://purl.oclc.org/NET/UNIS/fiware/iot-lite#>
PREFIX geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>
SELECT ?s ?qkt ?st ?ut ?lat ?long ?depl WHERE {
    ?s iot-lite:hasQuantityKind ?qk .
    ?qk a ?qkt .
    ?s a ?st .
    ?s iot-lite:hasUnit ?u .
    ?u a ?ut .
    OPTIONAL {
        ?s ssn:hasDeployment ?depl .
        ?s ssn:onPlatform ?plat .
        ?plat geo:location ?p .
        ?p geo:long ?long;
        geo:lat ?lat
    } .
    OPTIONAL {
        {
            ?s iot-lite:isSubSystemOf ?dev .
            ?dev ssn:hasDeployment ?depl .
            ?dev ssn:onPlatform ?plat .
            ?plat geo:location ?p .
            ?p geo:long ?long;
            geo:lat ?lat
        }
        OPTIONAL {
            ?dev ssn:hasSubSystem ?s .
            ?dev ssn:hasDeployment ?depl .
            ?dev ssn:onPlatform ?plat .
            ?plat geo:location ?p .
            ?p geo:long ?long;
            geo:lat ?lat
        }
    } .
}
```

```

    FILTER(bound(?plat)) .
}

```

Using the query, a search is performed for sensors that have a unit and a quantity kind. The sensor also has to have a testbed deployment and has to be on a platform that has a location. The deployment is used to determine afterwards to which testbed the sensor belongs. For every sensor the type, the unit, the quantity kind, the testbed and the location are stored into the database. Table 10 lists a sample query for retrieving observations:

Table 10: SPARQL Query

```

Prefix ssn: <http://purl.oclc.org/NET/ssnx/ssn#>
Prefix time: <http://www.w3.org/2006/time#>
Prefix xsd: <http://www.w3.org/2001/XMLSchema#>
Prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
Prefix dul: <http://www.loa.istc.cnr.it/ontologies/DUL.owl#>
select ?s ?ti ?dv where {
    ?o a ssn:Observation .
    ?o ssn:observedBy ?s .
    ?o ssn:observationResult ?or .
    ?or ssn:hasValue ?ov .
    ?ov dul:hasDataValue ?dv .
    ?o ssn:observationSamplingTime ?ot .
    ?ot time:inXSDDateTime ?ti .
}

```

Using the query, all observations are collected. For each observation: the related sensor, the time and the value is also gathered. The observations will be stored for every sensor in an array. For each observation, the value and the timestamp are only stored, other meta information like unit is retrieved from the sensor itself.

To limit the time to a specific interval, the IoT-Registry API supports setting the time boundaries per URL query parameters in the following way:

POST <OBSERVATIONS_QUERY_URL>?from=<FROM>&to=<TO>

Where the query url is /iot-registry/queries/execute/observations and <FROM> and <TO> are timestamps in the form 'YYYYMMDDHHmm'. See [7] for more details.

5.3.5.2.4 Database operations

The mongoDB is accessed via the pymongo module that maps the basic operations provided by the database to python. The tasks that will query the IoT-Registry are using the database to store all information that will be later consumed by other components of the tool.

The webserver that provides the GUI and the API is using the database to get the information, transform it in the required form and serve it.

5.3.5.2.5 UI tasks

The webserver fulfils two kinds of operations. The first is to provide the web sites in order to see the overview of all operations and also to present the detailed view of any testbed. The other is to provide an API for the data that is stored into the database.

5.3.5.2.6 Requesting OpenAM

In order to generate a specific view per user role in the UI, the monitoring tool uses the security component of the FIESTA-IoT platform. The UI is embedded in the portal UI that is protected by the security component. After a user is logged-in, a header is set for every further call. The monitoring tool uses this header in order to query the role of this specific user. After this, the required UI is compiled and delivered.

6 EXPERIMENTATION SERVICES AND API SPECIFICATION

6.1 Experiment Deployment Services

Below we list the experiment deployment related services provided by EEE. These services are services that ensures and target scheduling aspects, subscription and polling.

6.1.1 Scheduling APIs

The `/startFISMOExecution` starts the schedule as specified in the FISMO object. This API upon successful starting returns `{"response": "Job Scheduled", "jobID": <JobID>}`. The jobID and the status are stored in a database. The API reads the FISMO object associated with the FISMOID and its `QuerySchedule` attribute that contains scheduling information. The following scheduler services can be invoked using a path `https://<HOST>:<PORT>/schedulerServices/scheduler/<API>`

API	<code>/startFISMOExecution</code>
Description	This API is used to start execution of the experiment service (FISMO). This API provides a jobID to the FISMO upon the successful scheduling on the Meta Cloud. The API uses <code>timeSchedulePayload</code> to define the <code>startTime</code> , <code>stopTime</code> and <code>periodicity</code> of the job to be executed.
Method	POST
Input	<p>HeaderParam: String <code>fismoID</code> HeaderParam: String <code>femoID</code> HeaderParam: String <code>iPlanetDirectoryPro</code> HeaderParam: String <code>timeSchedulePayload</code></p> <p>The <code>timeSchedulePayload</code> is a JSON string that should contain <code>startTime</code>, <code>stopTime</code> and <code>periodicity</code>. A sample of such JSON is <code>{"startTime":"2016-09-15T13:57:00.0Z", "stopTime":"2016-09-15T16:30:00.0Z","periodicity":60}</code>. Here <code>startTime</code> and <code>stopTime</code> are in Date format (YYYY-MM-DD'T'HH:mm:ss.SSS'Z') and the <code>periodicity</code> is in seconds. The default value is set to <code>""</code>. The empty string is interpreted as 0.</p>
Output	<p><code>{"response": "Job Scheduled", "jobID": <JobID>}</code> is returned as a Response if successful. <code>{"response": <ERROR>}</code> is returned as a Response if unsuccessful.</p> <p>If the status is scheduled then <code>jobID</code> is returned.</p>
Produces	<code>application/json</code>

Errors	<ul style="list-style-type: none"> • “NoSuchServiceModelObjectID”: FISMOID is incorrect or does not exist. • “InvalidTimeScheduleStructure”: timeSchedulePayload JSON structure is incorrect or does not exist. • “UnParsableDate”: either startTime or stopTime is not in the correct format and thus cannot be parsed in the required format. • “SchedulerException”: is a generic error returned by the Quartz scheduler. • “ImplementationException”: is a generic error however with respect to this API it would mean “Failed to execute”
--------	--

To retrieve the jobIDs for a particular already scheduled FISMO, /getJobIDsfromFISMOID is used.

API	/getJobIDsfromFISMOID
Description	This API is used to get the jobID of a particular already scheduled FISMO. Note that this JobID is the ID given by the Scheduler to the FISMO execution.
Method	GET
Input	HeaderParam: String fismoID HeaderParam: String iPlanetDirectoryPro
Output	{“jobIDs”: [<JobID1>, <JobID2>..]} is returned as a Response if successful. Here the JobIDs is a list of job IDs associated to the FISMOID. A list is returned because there might be subscribers who might have subscribed to a particular FISMOID. Each subscription to a FISMOID, provides a new jobID to the subscription. This is because we consider each subscription to be different. {“response”: “No Jobs”} is also returned if there is no Jobs found for a particular FISMO. {“response”: <ERROR>} is returned as a Response if unsuccessful.
Produces	application/json
Errors	<ul style="list-style-type: none"> • “NoSuchServiceModelObjectID”: FISMOID is incorrect or does not exist • “SchedulerException”: is a generic error returned by the Quartz scheduler. • “ImplementationException”: is a generic error however with respect to this API it would mean “Failed to execute the API”

To retrieve jobID from a given fismoID, userID and femoID, /getJobIDfromFISMOIDUserIDandFEMOID is used.

API	/getJobIDfromFISMOIDUserIDandFEMOID
Description	This API is used to get the jobID associated to a particular fismoID, userID and femoID triple.
Method	GET
Input	HeaderParam: String fismoID HeaderParam: String femoID HeaderParam: String iPlanetDirectoryPro QueryParam: Boolean owner (default value true)
Output	{“jobID”: <JobID>} is returned as a Response if successful. {“response”: “No Job ID”} is also returned if there is no JobID was found for the input pair. {“response”: <ERROR>} is returned as a Response if unsuccessful. For the possible list of error please see the Errors row below.
Produces	application/json
Errors	<ul style="list-style-type: none"> • “NoSuchServiceModelObjectID”: FISMOID is incorrect or does not exist • “NoSuchExperimentID”: FEMOID is incorrect or does not exist • “PersistenceException”: is a generic error returned by the Quartz scheduler. • “ImplementationException”: is a generic error however with respect to this API it would mean “Failed to execute the API”

To retrieve the details about a jobID, /getJobIDDetails is used.

API	/getJobIDDetails
Description	This API is used to get the details associated to a particular jobID.
Method	GET
Input	HeaderParam: String jobID

	HeaderParam: String iPlanetDirectoryPro
Output	{“JobID”: <JobID>, “Group”: <GroupID>, “timeSchedule”: {“startTIme”: <startTime>, “stopTime”:<stopTime>, “periodicity”:<periodicity>}, “status”:<status>} is returned as a Response if successful. Here the groupID is the FISMOID and status is a job status from the list [BLOCKED, COMPLETE, ERROR, NONE, NORMAL, PAUSED]. {“response”: “No Job information found”} is also returned if there is no Jobs data. {“response”: <ERROR>} is returned as a Response if unsuccessful.
Produces	application/json
Errors	<ul style="list-style-type: none"> • “NoSuchJobID”: JobID is incorrect or does not exist • “SchedulerException”: is a generic error returned by the Quartz scheduler. • “ImplementationException”: is a generic error however with respect to this API it would mean “Failed to execute the API”

To retrieve the details about all jobIDs, /getAllJobIDDetails is used. This API is similar to the previous one.

API	/getAllJobIDDetails
Description	This API is used to get the details of all the jobIDs.
Method	GET
Input	HeaderParam: String iPlanetDirectoryPro
Output	{“JobsScheduled”: [{“jobID”: <JobID1>, “Group”: <GroupID>, “startTIme”: <startTime>, “stopTime”: <stopTime>, “periodicity”: <periodicity>, “status”: <status>}..]} is returned as a Response if successful. Here the groupID is the FISMOID and the status is a job status from the list [BLOCKED, COMPLETE, ERROR, NONE, NORMAL, PAUSED]. {“response”: “No Jobs Scheduled”} is also returned if there is no Jobs data. {“response”: <ERROR>} is returned as a Response if unsuccessful.
Produces	application/json
Errors	<ul style="list-style-type: none"> • “SchedulerException”: is a generic error returned by the Quartz scheduler.

	<ul style="list-style-type: none"> • “ImplementationException”: is a generic error however with respect to this API it would mean “Failed to execute the API”
--	--

Further, to get all the jobIDs for all the scheduled FISMOs use /getJobID

API	/getJobIDs
Description	To API is used to get all the existing jobIDs.
Method	GET
Input	HeaderParam: String iPlanetDirectoryPro
Output	{“jobIDs”: [{“jobID”: <JobID1>, “FISMOID”: <FISMOID>}..]} is returned as a Response if successful. Here the JobID is the job ID of the scheduled FISMOID. {“response”: “No Jobs Scheduled”} is also returned if there is no Jobs data. {“response”: <ERROR>} is returned as a Response if unsuccessful.
Produces	application/json
Errors	<ul style="list-style-type: none"> • “SchedulerException”: is a generic error returned by the Quartz scheduler. • “ImplementationException”: is a generic error however with respect to this API it would mean “Failed to execute the API”

The /stopJobExecution stops the job that was already started using the previous defined start APIs. This API takes as an input the JobID and stops the job by deleting it from the scheduler.

API	/stopJobExecution
Description	The API is used to pause the execution of a particular job
Method	POST
Input	HeaderParam: String jobID HeaderParam: String iPlanetDirectoryPro
Output	{“response”: “Job paused successfully”} is returned as a Response if successful. {“response”: <ERROR>} is returned as a Response if unsuccessful.
Produces	application/json

Errors	<ul style="list-style-type: none"> • “NoSuchJobID”: JobID is incorrect or does not exist • “SchedulerException”: is a generic error returned by the Quartz scheduler • “ImplementationException”: is a generic error however with respect to this API it would mean “Failed to execute the API”
--------	--

If a job is paused, it can also be resumed. To resume a job `/resumeJobExecution` is used.

API	<code>/resumeJobExecution</code>
Description	To API is used to resume the execution of a particular job
Method	POST
Input	HeaderParam: String jobID HeaderParam: String iPlanetDirectoryPro
Output	{“response”: “Job resumed successfully”} is returned as a Response if successful. {“response”: <ERROR>} is returned as a Response if unsuccessful.
Produces	application/json
Errors	<ul style="list-style-type: none"> • “NoSuchJobID”: JobID is incorrect or does not exist • “SchedulerException”: is a generic error returned by the Quartz scheduler • “ImplementationException”: is a generic error however with respect to this API it would mean “Failed to execute the API”

The EEE also provide APIs to reschedule, delete jobs and identify what are the currently executing jobs. This is achieved using `/rescheduleJob`, `/deleteScheduledJob`, `/deleteAllScheduledJobs` and `/getCurrentlyExecutingJobs`.

API	<code>/rescheduleJob</code>
Description	This API is used to change the schedule of an already scheduled Job.
Method	POST

Input	HeaderParam: String jobID HeaderParam: String iPlanetDirectoryPro HeaderParam: String timeSchedulePayload <p>The timeSchedulePayload is a JSON string that should contain startTime, stopTime and periodicity. A sample of such JSON is {"startTime": "2016-09-15T13:57:00.0Z", "stopTime": "2016-09-15T16:30:00.0Z", "periodicity": 60}. Here startTime and stopTime are in Date format (YYYY-MM-DD'T'HH:mm:ss.SSS'Z') and the periodicity is in seconds.</p>
Output	{“response”: “Job rescheduled successfully”} is returned as a Response if successful. {“response”: <ERROR>} is returned as a Response if unsuccessful.
Produces	application/json
Errors	<ul style="list-style-type: none"> • “NoSuchJobID”: JobID is incorrect or does not exist • “SchedulerException”: is a generic error returned by the Quartz scheduler • “ImplementationException”: is a generic error however with respect to this API it would mean “Failed to execute the API”

API	/deleteScheduledJob
Description	This API is used to remove a particular scheduled job from the Scheduler
Method	POST
Input	HeaderParam: String jobID HeaderParam: String iPlanetDirectoryPro
Output	{“response”: “Job deleted successfully”} is returned as a Response if successful. {“response”: “No Job found”} could also be returned. {“response”: <ERROR>} is returned as a Response if unsuccessful.
Produces	application/json
Errors	<ul style="list-style-type: none"> • “NoSuchJobID”: JobID is incorrect or does not exist • “SchedulerException”: is a generic error returned by the Quartz scheduler • “PersistenceException”: is a generic error returned by the Quartz scheduler.

	<ul style="list-style-type: none"> • “ImplementationException”: is a generic error however with respect to this API it would mean “Failed to execute the API”
--	--

API	/deleteAllScheduledJob
Description	This API is used to remove all scheduled job from the Scheduler. This API will be protected and will be only available to the FIESTA-IoT administrators.
Method	POST
Input	HeaderParam: String iPlanetDirectoryPro
Output	{“response”: “All Job deleted successfully”} is returned as a Response if successful. {“response”: “No Jobs found”} could also be returned. {“response”: <ERROR>} is returned as a Response if unsuccessful.
Produces	application/json
Errors	<ul style="list-style-type: none"> • “SchedulerException”: is a generic error returned by the Quartz scheduler • “PersistenceException”: is a generic error returned by the Quartz scheduler. • “ImplementationException”: is a generic error however with respect to this API it would mean “Failed to execute the API”

API	/getCurrentlyExecutingJobs
Description	This API is used to get all the jobs that are currently being processed. Note that this is different from listing all jobs that are available in the persistence store of the scheduler.
Method	GET
Input	HeaderParam: String iPlanetDirectoryPro
Output	{“response”: “Currently Executing Jobs.”, “Jobs”: [<jobs>..]} is returned if successful. {“response”: <ERROR>} is returned as a Response if unsuccessful.
Produces	application/json
Errors	<ul style="list-style-type: none"> • “SchedulerException”: is a generic error returned by the Quartz scheduler

	<ul style="list-style-type: none"> • “ImplementationException”: is a generic error however with respect to this API it would mean “Failed to execute the API”
--	--

To support ERM and provide single point of delete and update, EEE provides a set of triggers that should be used by the ERM to notify EEE whenever an experimenter deletes, reschedules or update a FISMO. Within this scenario, the APIs are `/fismoUpdateTrigger`, `/deleteFismoJobTrigger`, `/deletefismoJobTriggerlist` and `/deleteScheduledJobsOfFISMO`

API	<code>/fismoUpdateTrigger</code>
Description	This API is used to update a particular FISMO if it is already scheduled on the EEE.
Method	POST
Input	Body: FISMO fismo
Output	{“response”: “Job rescheduled successfully”} is returned as a Response if successful. {“response”: <ERROR>} is returned as a Response if unsuccessful.
Produces	application/json
Errors	<ul style="list-style-type: none"> • The FISMO object is null • “SchedulerException”: is a generic error returned by the Quartz scheduler • “ImplementationException”: is a generic error however with respect to this API it would mean “Failed to execute the API”

API	<code>/deleteFismoJobTrigger</code>
Description	This API is used to delete a particular FISMO if it is already scheduled on the EEE.
Method	POST
Input	HeaderParam: String fismoID
Output	{“response”: “Job deleted successfully”} is returned as a Response if successful. {“response”: <ERROR>} is returned as a Response if unsuccessful.

Produces	application/json
Errors	<ul style="list-style-type: none"> • “SchedulerException”: is a generic error returned by the Quartz scheduler • “PersistenceException”: is a generic error returned by the Quartz scheduler. • “ImplementationException”: is a generic error however with respect to this API it would mean “Failed to execute the API”

API	/deletefismoJobTriggerlist
Description	This API is used to delete list of FISMOs if it is already scheduled on the EEE.
Method	POST
Input	Body: String fismoIDs In JSONArray format
Output	{“response”: “FISMOs deleted successfully”} is returned as a Response if successful. {“response”: <ERROR>} is returned as a Response if unsuccessful.
Produces	application/json
Errors	<ul style="list-style-type: none"> • No FISMOs Specified. • “SchedulerException”: is a generic error returned by the Quartz scheduler • “PersistenceException”: is a generic error returned by the Quartz scheduler. • “ImplementationException”: is a generic error however with respect to this API it would mean “Failed to execute the API”

API	/deleteScheduledJobsOfFISMO
Description	This API is used to delete jobs associated to a particular FISMO.
Method	POST
Input	HeaderParam: String fismoIDs HeaderParam: String iPlanetDirectoryPro

Output	{“response”: “All jobs associated to Fismo are deleted”} is returned as a Response if successful. {“response”: <ERROR>} is returned as a Response if unsuccessful.
Produces	application/json
Errors	<ul style="list-style-type: none"> • “SchedulerException”: is a generic error returned by the Quartz scheduler • “PersistenceException”: is a generic error returned by the Quartz scheduler. • “ImplementationException”: is a generic error however with respect to this API it would mean “Failed to execute the API”

6.1.2 Subscription APIs

The subscription services (/subscribeToFISMOReport and /unsubscribeToFISMOReport to the discoverable FISMOs) are used so that an experimenter can subscribe to existing discoverable FISMOs or unsubscribe from already subscribed FISMO. The following subscription based services can be invoked using a path `https://<HOST>:<PORT>/schedulerServices/subscription/<API>`

API	/subscribeToFISMOReport
Description	This API is used to subscribe to a particular FISMO’s report
Method	POST
Input	HeaderParam: String fismoID HeaderParam: String userID HeaderParam: String femoID HeaderParam: String iPlanetDirectoryPro HeaderParam: String experimentOutput <p>Here the experimentOutput is the ExperimentOutput attribute of the FISMO in the JSON ({“url”: <url>}). A sample of currently valid experimentOutput is {“url”: “http://myExperiment.com”}. Further, the userID is the ID of the experimenter, and the femoID is the ID of the experiment to which subscription is to be associated to.</p>
Output	{“response”: “subscribed”, “FISMOID”: <FISMOID>, “JobID”: <JobID>} is returned as a Response if successful. {“response”: <ERROR>} is returned as a Response if unsuccessful.

Produces	application/json
Errors	<ul style="list-style-type: none"> • “NoSuchServiceModelObjectID”: FISMOID is incorrect or does not exist • “NoSuchUserID”: userID is incorrect or does not exist • “NoSuchExperimentID”: FEMOID is incorrect or does not exist • “AlreadySubscribed”: FISMOID is already subscribed and associated to the userID. • “InvalidURL”: invalid url • “InvalidExperimentOutputJson”: invalid Experiment Output Json • “ImplementationException”: is a generic error however with respect to this API it would mean “Failed to execute the API or subscription failed”

API	/unsubscribeToFISMOReport
Description	This API is used to unsubscribe from a particular FISMO’s report
Method	POST
Input	Header Param: String fismoID HeaderParam: String iPlanetDirectoryPro Header Param: String femoID femoID is ID of the experiment to which subscription is to be associated to.
Output	{“response”: “Unsubscribed”} is returned as a Response if successful. {“response”: <ERROR>} is returned as a Response if unsuccessful.
Produces	application/json
Errors	<ul style="list-style-type: none"> • “NoSuchServiceModelObjectID”: FISMOID is incorrect or does not exist • “NoSuchUserID”: userID is incorrect or does not exist • “NoSuchExperimentID”: FEMOID is incorrect or does not exist • “SubscriptionNotFound”: FISMOID is not associated to the userID. • “ImplementationException”: is a generic error however with respect to this API it would mean “Failed to execute the API or un-subscription failed”

6.1.3 Polling APIs

A polling service is a service using which an experimenter can run the FISMO once without actually scheduling it. The following polling based services can be invoked using a path `https://<HOST>:<PORT>/schedulerServices/polling/<API>`

API	/pollForReport
Description	This API is used to invoke a previously defined FISMO. A call to this API will only produce one Resultset that will be sent to the URL specified in the ExperimentOutput parameter.
Method	POST
Input	HeaderParam: String fismoID HeaderParam: String femoID HeaderParam: String iPlanetDirectoryPro QueryParam: Boolean owner (default value true). This parameter basically tells the EEE if it has to look into subscriber realm or the owner realm
Output	{“response”: “Polled Successfully”: “jobID”: <JOBID>} is returned as a Response if successful. Here JobID is the jobID of the generated for the particular poll. Experimenters are advised to keep this jobID in their record. {“response”: <ERROR>} is returned as a Response if unsuccessful.
Produces	application/json
Errors	<ul style="list-style-type: none"> • Something went wrong • FIESTA-IoT Analytics tool was not invoked correctly. Thus polling failed. • “NoSuchServiceModelObjectID”: FISMOID is incorrect or does not exist • “NoSuchExperimentID”: FEMOID is incorrect or does not exist • “InvalidURL”: invalid url • “InvalidExperimentOutputJson”: invalid Experiment Output Json • “ImplementationException”: is a generic error however with respect to this API it would mean “Failed to execute the API”

API	/dynamicPollForReport
Description	This API is used to invoke a previously defined FISMO. A call to this API will only produce one Resultset that will be sent to the URL specified in

	the ExperimentOutput parameter. However, this API is different from previous API with respect to the possibility of providing parameter values. This is useful in the case of mobile applications.
Method	POST
Input	<p>HeaderParam: String fismoID HeaderParam: String femoID HeaderParam: String iPlanetDirectoryPro QueryParam: Boolean owner (default value true) QueryParam: String geoLatitude (default value “0”) QueryParam: String geoLongitude (default value “0”) QueryParam: int intervalNowToPast (default value 0) QueryParam: Long fromTime (default value “0L”) QueryParam: Long toTime (default value “0L”) Body: String Others</p> <p>This is a JSON object represented as a string. The default value is “{}”. However, experimenters need to set the key value pair depending on the query. A JSON object the experimenters need to set is</p> <pre>{ "KATInput": {"Method": [""], "Parameters": [""]}, "otherParameters": {<key>:<value>} }</pre> <p>Here, KATInput essentially reflects the input needed for the FIESTA-IoT Analytics Toolkit, while otherParameters reflect the dynamic attributes</p>
Output	<p>{“response”: “Dynamically Polled Successfully”: “jobID”: <JOBID>} is returned as a Response if successful. Here JobID is the jobID of the generated for the particular poll. Experimenters are advised to keep this jobID in their record. {“response”: <ERROR>} is returned as a Response if unsuccessful.</p>
Produces	application/json
Errors	<ul style="list-style-type: none"> • Something went wrong • FIESTA-IoT Analytics tool was not invoked correctly. Thus, polling failed. • “NoSuchServiceModelObjectID”: FISMOID is incorrect or does not exist • “NoSuchExperimentID”: FEMOID is incorrect or does not exist • “JSONException”: invalid JSON • “QueryException”: invalid query and Parameters

	<ul style="list-style-type: none"> • “InvalidExperimentOutputJson”: invalid Experiment Output Json • “ImplementationException”: is a generic error however with respect to this API it would mean “Failed to execute the API”
--	---

6.2 Experiment Management Services

In this section, we list the experiment management APIs that are provided by the EEE and the testbed status Monitoring services.

6.2.1 EEE Monitor APIs

Here we list all the APIs that provide “meta” information about an experiment and the associated services (FISMOs). The following monitoring based services can be invoked using a path `https://<HOST>:<PORT>/schedulerServices/monitoring/<API>`

API	/getJobIDStatus
Description	This API is used to get the status of a particular jobID, i.e., one from the list [BLOCKED, COMPLETE, ERROR, NONE, NORMAL, PAUSED]
Method	GET
Input	QueryParam: String jobID HeaderParam: String iPlanetDirectoryPro
Output	{“JobID”: <JobID>, “status”: <STATUS>} is returned as a Response if successful. Here STATUS is one from the list as described above. Other messages that are returned are {“response”: “Job not Scheduled”} {“response”: <ERROR>} is returned as a Response if unsuccessful.
Produces	application/json
Errors	<ul style="list-style-type: none"> • “NoSuchJobID”: JobID is incorrect or does not exist • “SchedulerException”: is a generic error returned by the Quartz scheduler • “ImplementationException”: is a generic error however with respect to this API it would mean “Failed to execute the API”

API	/getAllSubscribersOfFISMOID
Description	This API is used to get a list of subscribers (or the experimenters) that are using a particular FISMO.

Method	GET
Input	QueryParam: String fismoID HeaderParam: String iPlanetDirectoryPro
Output	{“UserIDs”: [<UserID1>, <UserID2>,..]} is returned as a Response if successful. Here, the “UserIDs” is a list of userIDs that have subscribed to the particular FISMO. It is also possible to get an empty JSON object if there is no user that has subscribed to the given FISMOID. {“response”: <ERROR>} is returned as a Response if unsuccessful.
Produces	application/json
Errors	<ul style="list-style-type: none"> • “NoSuchServiceModelObjectID”: FISMOID is incorrect or does not exist • “ImplementationException”: is a generic error however with respect to this API it would mean “Failed to execute the API”

API	/getAllSubscriptionsOfExperimenter
Description	This API is used to get a list of user subscriptions irrespective of the experiment
Method	GET
Input	HeaderParam: String iPlanetDirectoryPro
Output	{“FISMOIDs”: [<FISMOID1>, < FISMOID2>,..]} is returned as a Response if successful. Here, the “FISMOIDs” is a list of FISMOIDs that the user has subscribed. It is also possible to get an empty JSON object if there are no FISMOIDs that a user has subscribed. {“response”: <ERROR>} is returned as a Response if unsuccessful.
Produces	application/json
Errors	<ul style="list-style-type: none"> • “ImplementationException”: is a generic error however with respect to this API it would mean “Failed to execute the API”

API	/getMySubscriptionsforExperiment
Description	This API is used to get a list of user subscriptions with respect to a particular experiment

Method	GET
Input	HeaderParam: String iPlanetDirectoryPro QueryParam: String femoID
Output	{“Subscriptions”: [{“jobID”: <jobID>, “fismoID”: <FISMOID1>}, ..]} is returned as a Response if successful. Here, the “Subscriptions” is a list of jobIDs and FISMOIDs that the user has subscribed. It is also possible to get an empty JSON object if there are no subscriptions for a particular experiment by the user. {“response”: <ERROR>} is returned as a Response if unsuccessful.
Produces	application/json
Errors	<ul style="list-style-type: none"> • “NoSuchUserID”: userID is incorrect or does not exist • “NoSuchExperimentID”: FEMOID is incorrect or does not exist • “ImplementationException”: is a generic error however with respect to this API it would mean “Failed to execute the API or Subscription Failed”

API	/getJobExecutionLog
Description	This API is used to get the ExecutionLog of a Job. The return is a JSON array with “executionTime” and “dataConsumed” information. Here executionTime is the time it took to successfully execute the Job.
Method	GET
Input	QueryParam: String jobID HeaderParam: String iPlanetDirectoryPro
Output	{“ExecutionLog”: [{“executionTime”: <time1>, “dataConsumed”: <dataConsumed1>}, {“executionTime”: <time2>, “dataConsumed”: <dataConsumed2>}, ..]} is returned as a Response if successful. Here, the “ExecutionLog” is a log of successful executions of jobID. It is also possible to get an empty JSON object if there is no ExecutionLog for the jobID. {“response”: <ERROR>} is returned as a Response if unsuccessful.
Produces	application/json
Errors	<ul style="list-style-type: none"> • “NoSuchJobID”: JobID is incorrect or does not exist • “ImplementationException”: is a generic error however with respect to this API it would mean “Failed to execute the API”

6.2.2 EEE Accounting APIs

Here we list all the APIs that provide counting of the number of times experiments associated to an experimenter have been executed and the number of times a particular experiment service (FISMOs) has been executed. The following accounting based services can be invoked using a path `https://<HOST>:<PORT>/schedulerServices/accounting/<API>`

API	/getUserExecutionCount
Description	This API is used to get the number of times a particular user has executed experiments
Method	GET
Input	HeaderParam: String iPlanetDirectoryPro QueryParam: String fromTime QueryParam: String toTime (default "") The fromTime is a string that should be in the format YYYY-MM-DD'T'HH:mm:ss.SSS'Z'. A sample fromTime is "2016-09-15T13:57:00.0Z". In case toTime is not provided, UTC now will be used.
Output	{"count": <count>} is returned as a Response if successful. Here, the "count" is the number of times a user has executed experiments. Note that the count can also be 0. {"response": <ERROR>} is returned as a Response if unsuccessful.
Produces	application/json
Errors	<ul style="list-style-type: none"> • "UnParsableDate": fromTime is not in the correct format and thus cannot be parsed in the required format. • "PersistenceException": is a generic error returned by the Quartz scheduler. • "ImplementationException": is a generic error however with respect to this API it would mean "Failed to execute the API"

API	/getJobExecutionCount
Description	This API is used to get the number of times a particular job was executed.
Method	GET

Input	QueryParam: String jobID HeaderParam: String iPlanetDirectoryPro
Output	{“count”: <count>} is returned as a Response if successful. Here, the “count” is the number of times the job is executed. Note that the count can also be 0. {“response”: <ERROR>} is returned as a Response if unsuccessful.
Produces	application/json
Errors	<ul style="list-style-type: none"> • “NoSuchJobID”: JobID is incorrect or does not exist • “PersistenceException”: is a generic error returned by the Quartz scheduler. • “ImplementationException”: is a generic error however with respect to this API it would mean “Failed to execute the API”

6.2.3 FIESTA-IoT Access Mechanism & Testbeds status APIs

Here we list all the APIs that provide experimenters info on the testbed status, monitor a sensor etc. The following APIs can be invoked using a path <https://<HOST>:<PORT>/testbed-monitoring/api/<API>>.

API	/testbeds
Description	This API is used to get all testbeds that are known by the monitoring tool.
Method	GET
Input	None
Output	Returns the list of testbeds in the format: {“<TESTBED_IRI>”: “<TESTBED_NAME>”, ..} Here <TESTBED_IRI> is the identifier which is used in the lot-Registry.
Produces	application/json
Errors	<ul style="list-style-type: none"> • None

API	/testbeds/activated
Description	This API is used to get all testbeds that are activated.

Method	GET
Input	None
Output	Returns the list of all activated testbeds in the same format as in /testbeds.
Produces	application/json
Errors	<ul style="list-style-type: none"> • None

API	/testbeds/<string:testbed_name>
Description	This API is used to get all known information about specific testbed.
Method	GET
Input	URLParam: String : The TESTBED_IRI or the INTERNAL_ID of the wanted testbed
Output	<p>Returns the testbed information in the following format:</p> <pre> { "activated": <activated>, "location": { "latitude": <latitude>, "longitude": <longitude> }, "testbed_name": <TESTBED_NAME>, "sensors": { "active": <active_sensors>, "relative": <relative_sensors>, "total": <total_sensors> }, "_id": <INTERNAL_ID>, "testbed_iri": <TESTBED_IRI> }</pre>
Produces	application/json
Errors	<ul style="list-style-type: none"> • {"error_msg": "No testbed found for <TESTBED_IRI>", "error": true}

API	/testbeds/<string:testbed_name>/sensors
Description	This API is used to get all known sensors from a specific testbed.
Method	GET
Input	<p>URLParam: String - The TESTBED_IRI or the INTERNAL_ID of the wanted testbed</p> <p>QueryParam: String <i>sensor-type</i> – The list can be filter by the type of sensor (e.g.: m3-lite:HumiditySensor)</p> <p>QueryParam: String <i>unit</i> – The list can be filter by the measured unit of sensor (e.g.: m3-lite:Percent)</p> <p>QueryParam: String <i>quantity-kind</i> – The list can be filter by the measured quantity kind of sensor (e.g.: m3-lite:RelativeHumidity)</p>
Output	<p>Returns the list of all sensors in the following format:</p> <pre>[{ "sensor_name": <SENSOR_NAME>, "longitude": <longitude>, "latitude": <latitude>, "sensor_type": <sensor_type>, "unit": <unit>, "deployment": <deployment>, # The ID of the testbed this sensor is deployed on "quantity_kind": <quantity_kind>, "newest_value": { "color": <color>, "value": <value> }, "newest_date": { "color": <color>, "value": <value> }, "_id": <INTERNAL_ID> }, {...}, ...]</pre>
Produces	application/json
Errors	<ul style="list-style-type: none"> • {"error_msg": "No testbed found for <TESTBED_IRI>", "error": true}

API	/sensors
Description	This API is used to get all known sensors from all testbeds.
Method	GET
Input	<p>QueryParam: String <i>sensor-type</i> – The list can be filter by the type of sensor (e.g.: m3-lite:HumiditySensor)</p> <p>QueryParam: String <i>unit</i> – The list can be filter by the measured unit of sensor (e.g.: m3-lite:Percent)</p> <p>QueryParam: String <i>quantity-kind</i> – The list can be filter by the measured quantity kind of sensor (e.g.: m3-lite:RelativeHumidity)</p>
Output	Returns the list of all sensors in the same format as in /testbeds/<string:testbed_name>/sensors
Produces	application/json
Errors	None

API	/sensors/<string:sensor_name>
Description	This API is used to get all information about one specific sensor.
Method	GET
Input	URLParam: String : The SENSOR_NAME or the INTERNAL_ID of the wanted sensor
Output	<p>Returns the sensor information in the following format:</p> <pre>{ "sensor_name": <SENSOR_NAME>, "longitude": <longitude>, "latitude": <latitude>, "sensor_type": <sensor_type>, "unit": <unit>, "deployment": <deployment>, # The INTERNAL_ID of the testbed this sensor is deployed on "quantity_kind": <quantity_kind>, "newest_value": { "color": <color>, "value": <value> }, "newest_date": { "color": <color>, "value": <value> } }</pre>

	<pre> }, "_id": <INTERNAL_ID> } </pre>
Produces	application/json
Errors	<ul style="list-style-type: none"> • {"error_msg": "No sensor found for <SENSOR_NAME>", "error": true}

API	/sensors/<string:sensor_name>/observations
Description	This API is used to get all observations for one specific sensor.
Method	GET
Input	URLParam: String : The SENSOR_NAME or the INTERNAL_ID of the wanted sensor
Output	<p>Returns the list of observations for this sensor in the following format:</p> <pre> [{ "time_value": <ISO_TIME>, "data_value": <VALUE> }, {...}, ...] </pre>
Produces	application/json
Errors	<ul style="list-style-type: none"> • {"error_msg": "No sensor found for <SENSOR_NAME>", "error": true}

API	/testbeds/<string:testbed_iri>/activate
Description	<p>This API is used to activate a testbed in the monitoring tool. All testbeds will be monitored but only activated testbeds will be shown in the GUI.</p> <p>Only FIESTA-IoT admins are permitted to activate and deactivate testbeds.</p>
Method	GET
Input	URLParam: String : The TESTBED_IRI or the INTERNAL_ID of the wanted testbed

Output	Returns the activated testbed in the same format as in /testbeds/<string:testbed_iri>
Produces	application/json
Errors	<ul style="list-style-type: none"> • {"error_msg": "Error while activating testbed <TESTBED_IRI>. Testbed could not be found.", "error": true}

API	/testbeds/<string:testbed_iri>/deactivate
Description	This API is used to deactivate a testbed in the monitoring tool.
Method	GET
Input	URLParam: String: The TESTBED_IRI or the INTERNAL_ID of the wanted testbed
Output	Returns the deactivated testbed in the same format as in /testbeds/<string:testbed_iri>
Produces	application/json
Errors	<ul style="list-style-type: none"> • {"error_msg": "Error while deactivating testbed <TESTBED_IRI>. Testbed could not be found.", "error": true}

6.3 Experiment ResultSet Storage APIs

Here we list all the APIs that provide ERS functionalities. The following APIs can be invoked using a path `https://<HOST>:<PORT>/experiment-result-store`

API	/experiment-result-store
Description	This interface allows experiment results to be stored in persistence until it is retrieved by the experimenter. Results must be stringified and encapsulated in a JSON object.
Method	POST
Input	HeaderParam: String userID, Username of the client HeaderParam: String femoID HeaderParam: String jobID: optional, jobID of the FISMO in the EEE

Output	204 OK
Produces	application/json
Errors	400 Bad Request

API	/experiment-result-store
Description	<p>All result sets that cannot be sent to the experimenters are stored in the Experiment Result Storage (ERS). ERS stores result set as is and returns them when service is invoked. Upon a success, the particular result set is deleted from the store.</p> <p>Experimenters need to use an ERS API to download the needed data. This API has a signature</p>
Method	GET
Input	<p>HeaderParam: String iPlanetDirectoryPro</p> <p>HeaderParam: String femoID</p> <p>HeaderParam: String jobID: optional, JobID of the FISMO in the EEE</p> <p>If both FEMOID and JobID are provided, then the corresponding FISMO results are returned.</p> <p>If only the FEMOID is provided, then all FISMO execution results under that particular FEMO along with its corresponding job IDs are returned.</p>
Output	<p>On successful response following provided template is returned</p> <pre>{ "femoResults": [{ "jobid": "<JOBID>", "results": [{ "time": "<TIMESTAMP>", "result": "<RESULTSET>" }] },.....]}</pre> <p>{“response”: <ERROR>} is returned as a Response if unsuccessful</p>
Produces	application/json
Errors	<p>400 Bad Request</p> <p>401 Unauthorized</p>

6.4 Documentation of APIs

The EEE API documentation was built using Swagger and is available to the Experimenters for testing and understanding. The EEE APIs are divided into 2 categories: one for the experimenters and another for the FIESTA-IoT Admins. For security purposed we just release the link¹ for the APIs that are made public to the experimenters. The public version of the APIs for other components like FAT² and ERS³ is available in Markdown. Please note that for some tools like the monitoring tool, the API documentation is still under implementation phase. It will soon be added to the portal.

¹ <https://platform.fiesta-iot.eu/EEEpdocs/>

² FAT API - <https://gist.github.com/UniSurreyIoT/521a1927681ff0727ab1d2a1d89e1b0c>

³ ERS API - <https://gist.github.com/UniSurreyIoT/c5fb321fd2c0b519cd3ef5f6793d7ffd>

7 PROTOTYPE

7.1 PORTAL

The FIESTA-IoT project has developed a portal to be used by all the users of the platform as a one-stop shop for all activities. This web-portal has been re-designed in order to improve both the user experience and the look and feel. The current version of the portal is based on bootstrapping CSS and html5 and provides a simple but user-friendly interface. The welcome page of the portal is shown in Figure 33. As it can be seen, the simple interface provides a left vertical menu, leaving the rest of the page free for the actual content.

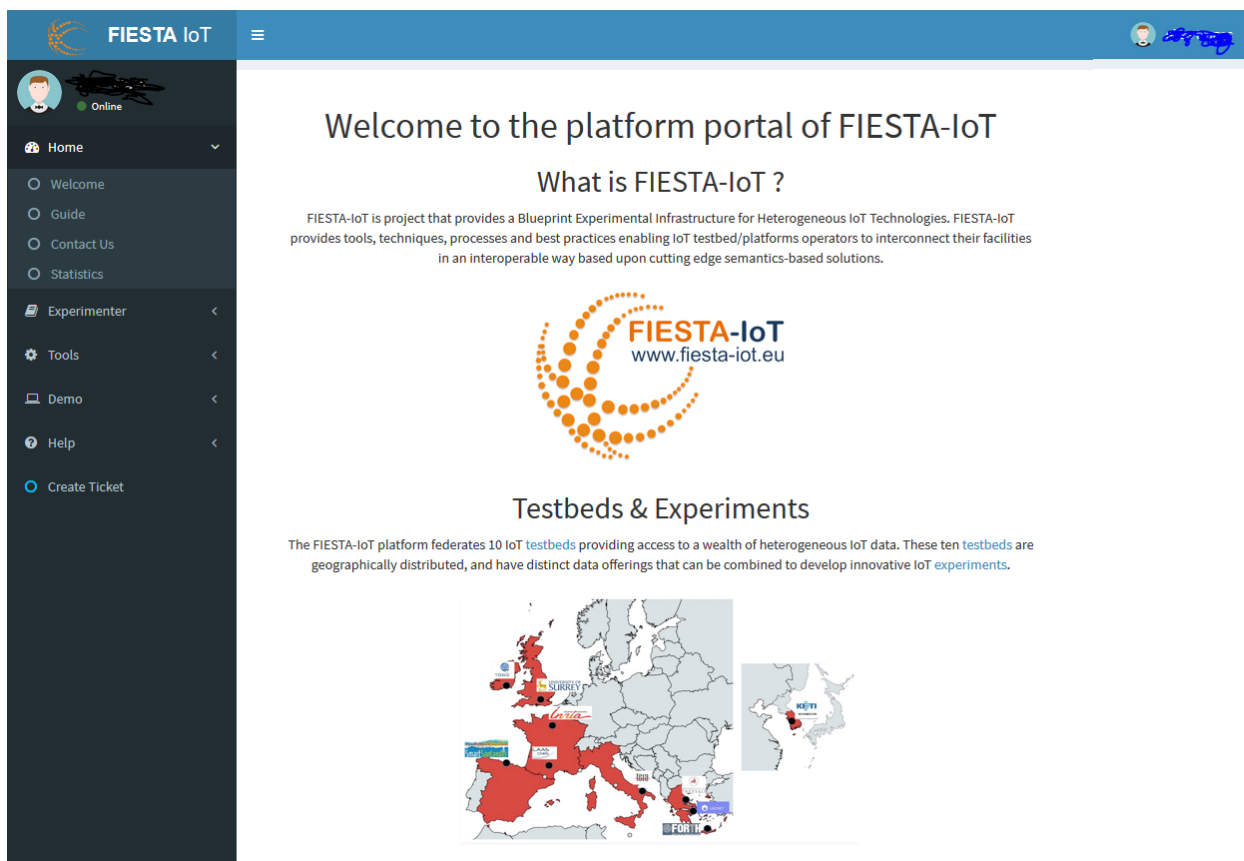



Figure 33: Portal welcome page

7.1.1 Signing in

The FIESTA-IoT portal is not accessible publicly and only registered and accredited users have access. In order to login to the portal, the users should access the page: <https://platform.fiesta-iot.eu> that will redirect automatically to the login page of the OpenAM [10], as shown in Figure 34.

After using the correct credentials, the users are redirected to <https://platform.fiesta-iot.eu/portalui> that displays the initial web page of the portal, as shown in Figure 33.



The logo for FIESTA-IoT features a stylized orange circular graphic composed of dots, with the text "FIESTA-IoT" in orange and blue, and the website "www.fiesta-iot.eu" in blue below it.

SIGN IN TO FIESTA-IOT

User Name

Password

☐ Remember my username

LOG IN

[Forgot Username?](#) | [Forgot Password?](#)

[New here? Create an account](#)

Figure 34: Portal login page

7.1.2 Menus

The portal provides five different menu categories:

- **Home:** this is the initially displayed menu, which includes some generic interest pages that are accessible by all users that are registered to the portal. This category includes the following pages:
 - **Welcome:** this displays the initial welcome page with general information about the FIESTA-IoT project.
 - **Guide:** this page shows some basic information with links to the FIESTA-IoT Moodle pages for the complete guides for experimenters and testbed providers.
 - **Contact us:** this page includes information for contacting the support team or the project management team.
 - **Statistics:** this page includes some sample statistics about the usage of the FIESTA-IoT platform. It includes two tables, with (i) mapping of experiments per testbed and (ii) mapping of testbeds per domain. It also includes two graphs with statistics for the reasoning tool and the number of registered devices per quantity kind. This page is shown in Figure 35.
- **Experimenter:** this is the main menu category for experimenters to create, edit and manage their experiments using the developed user interfaces. This menu includes the following web pages:

- Experiment editor: this is the tool for creating and editing experiments (see Section 5.3.1 for more details).
- Experiment Register Client: this is the tool for uploading and registering experiments via a FEDSPEC file.
- Management console: this is the tool for managing, scheduling and running experiments (see Section 5.3.2 for more details).
- Testbed provider: this is the main menu category used by testbed providers for registering their testbed, the resources and configuring them. It includes the following pages:
 - Register testbed: this page displays the tool for the online registration of a new testbed.
 - Register resources: this page displays the tool for the online registration of new devices for the selected testbed in various ways (by text, by upload or manually).
 - TPI configurator: this page includes the tool for configuring the testbed provider interface.
- Tools: this menu category includes additional tools that can be used by experimenters (and/or testbed providers), with extra functionalities that are useful but are not mandatory. These functionalities are:
 - Testbed monitoring: this page displays the tool for monitoring the status of the testbeds (for more information see Section 5.3.5).
 - Certification [14]: this link redirects the testbed user to the certification portal of the FIESTA-IoT project, where the testbed providers can get validation for the standardised way their testbeds are integrated in the platform.
 - Reasoning: this is another menu category to be used by experimenters for creating, registering and running rules that can be helpful for their experiments (see Section 5.3.3 for more details).
- Demo: this menu category includes sample experiment demos that show the full functionality of the FIESTA-IoT platform.
- Help: this menu category provides several helping pages:
 - About FIESTA-IoT: this is a link to the web-page of the project.
 - Support: this is a link to the support page of the website of the project.
 - Documentation: this is a list of web pages providing the documentation of all tools and functionalities developed by the FIESTA-IoT project.
 - Social Media Resources: this provides links to the youtube channel, and the twitter and slideshare accounts of the FIESTA-IoT project.
- Create Ticket: this link provides a quick access tool for the users in order to create tickets for asking help by the FIESTA-IoT team or for submitting issues and problems.



Figure 35: Portal statistics page

7.1.2.1 Access control / Roles for Menu

For the FIESTA-IoT portal there are four main access role categories:

- Registered users
- Experimenters
- Testbed providers
- Administrators

The portal has been designed to provide different access to the menus and the web pages according to the role of the user. Access is controlled by the usage of a JSON file, with roles, URLs, targets, CSS icon styles, etc. Based on the current logged on user, the data are filtered in the JSON file to disable/enable menu on the portal. Table 11 shows a mapping of the portal menus to the user roles.

Table 11: Access roles per portal menu

Menu Category	Registered user	Experimenter	Testbed provider	Administrator
Home	X	X	X	X
Experimenter		X		X
Testbed Provider			X	X
Tools		X	X	X
Demo	X	X	X	X
Help	X	X	X	X
Create Ticket	X	X	X	X

An example for the controlling of the access to the Testbed provider menu is given below, showing that only the administrator and the testbed provider are allowed access:

```
{
  "name": "Testbed Provider",
  "roles": ["fiestaAdmin", "testbedAdmin"],
  "cssClass": "fa fa-snowflake-o",
  "submenus": [
    {
      "name": "Register Testbed",
      "url": "https://platform.fiesta-iot.eu/ui.testbed-registry/#/register-testbeds",
      "roles": ["fiestaAdmin", "testbedAdmin"],
      "cssClass": "fa fa-circle-o"
    },
    {
      "name": "Register Resources",
```

```
    "url": "https://platform.fiesta-iot.eu/ui.testbed-  
registry/#/register-devices",  
    "roles": ["fiestaAdmin", "testbedAdmin"],  
    "cssClass": "fa fa-circle-o"  
  },  
  {  
    "name": "TPI Configurator",  
    "url": "https://platform.fiesta-  
iot.eu/tpi.configurator/index.zul",  
    "roles": ["fiestaAdmin", "testbedAdmin"],  
    "cssClass": "fa fa-circle-o"  
  }  
]  
}
```

7.2 Usage

7.2.1.1 Testbeds

The tools for registering a Testbed, registering resources and managing the testbed interface have been provided in the rest of WP3 and WP4 Deliverables (especially [15], [16]) so will not be described again here to avoid repetition.

7.2.1.2 Experimenters

7.2.1.2.1 Create an Experiment

FIESTA-IoT provides a web tool to create and edit experiments called Experiment Editor. There are a couple of ways to create experiment using the Experiment Editor: one is to create a new experiment, another is to duplicate an existing experiment option.

To create a new experiment using the Experiment Editor we have to click on the add icon on the rectangular block, as shown in Figure 7, which would then redirect us to the new experiment template. This template can be divided in to three blocks: FEMO, FISMOs and Query. Note that the experiment editor follows the defined FIESTA-IoT experiment DSL.

The screenshot shows the 'New FEMO' form. It has three main sections: 'FEMO Name:', 'FEMO Description:', and 'Domain of interest:'. The 'FEMO Name' field contains 'New FEMO'. The 'FEMO Description' field contains 'New FEMO'. The 'Domain of interest' field contains a URL 'http://purl.org/iot/vocab/m3-lite#DomainOfInterest' and a button 'Add a new DomainOfInterest'.

Figure 36: Experiment Template FEMO

The FEMO block contains three fields FEMO Name, FEMO Description and Domain of Interest as shown in the Figure 36. For details on the FEMO we refer the readers to [12].

A FEMO should have at least one FISMO that is created from the initial template. An experimenter can add multiple FISMOS depending on the experiment requirements by clicking the add icon in the FISMO block. Each created FISMOS are listed in align with the FISMO block and every FISMO comes with two immediate options next to its name, Duplicate FISMO operation and Delete FISMO operation. Duplicating a FISMO would create a new FISMO with the same parameters as the existing FISMO, while clicking on the delete icon would remove that particular FISMO from the FEMO. Note that the changes will not take place unless the save button is clicked.

The screenshot shows the 'New FISMO' form. It has several sections: 'FISMO Name:', 'FISMO Description:', 'Discoverable:' (with a toggle switch), 'Experiment control:', 'Scheduling:' (with 'Start time', 'Stop time', and 'Periodicity' fields), and 'Experiment output:'. The 'Experiment output' section has a 'Location' field, a 'File' field, and a 'Widget:' section with a 'Create widget' button.

Figure 37: Experiment template FISMO

A FISMO template consists of seven fields, FISMO Name, FISMO Description, Discoverable, Experiment Control, Experiment Output and Widget as shown in Figure 37. For details on the FISMO we refer the readers to [12].

Every FISMO contains a single Query. The query block contains Quantity Kind, Static Location, Query Interval, and Dynamic Attributes as shown in the Figure 38. For details on the Query Control we refer the readers to [12].

Quantity kind:

Static location:
 Latitude: Longitude:

Query interval:
 From date: To date: Interval now to past:

[Build new query from user input](#)

Query request:

```

1 PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
2 PREFIX ssn: <http://purl.oclc.org/NET/ssn/ssn#>
3 PREFIX iot-lite: <http://purl.oclc.org/NET/UNIS/firmware/iot-lite#>
4 PREFIX owl: <http://www.w3.org/ns/owl#>
5 PREFIX time: <http://www.w3.org/2006/time#>
6 PREFIX cu: <http://purl.oclc.org/NET/cu/quantity#>
7 PREFIX geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>
8 PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
9 # url: https://platform-dev.fiesta-iot.eu/iot-registry/api/queries/execute/global
10 SELECT DISTINCT ?sensor ?val ?timeStamp
11 WHERE {
12   # remove the hash sign from the filters to enable them
  
```

Dynamic Attributes:

Predefined Dynamic Attribute:
 Dynamic Geo Location:
 Latitude: Longitude:
 Dynamic Query Interval:
 From date time: To date time: Interval now to past:

Dynamic Attribute:

Name	Default Value
<input type="text" value="input new name"/>	<input type="text" value="input new value"/>

Figure 38: Experiment Template Query

Using the duplicate option in the FEMO block at start of Experiment Editor will result in creating a new experiment with all the FISMO and Queries in the existing experiment.

7.2.1.2.2 Register new Experiment

FIESTA-IoT is currently offering a simple interface in order to store, update and delete experiments called Experiment Register Client. This UI is used in case experiment was using FEDSpec based execution and created the FEDSpec using proprietary tool other than Experiment Editor. The Experiment Register Client can be found at the Experimenter menu of the FIESTA-IoT portal (see Figure 39).

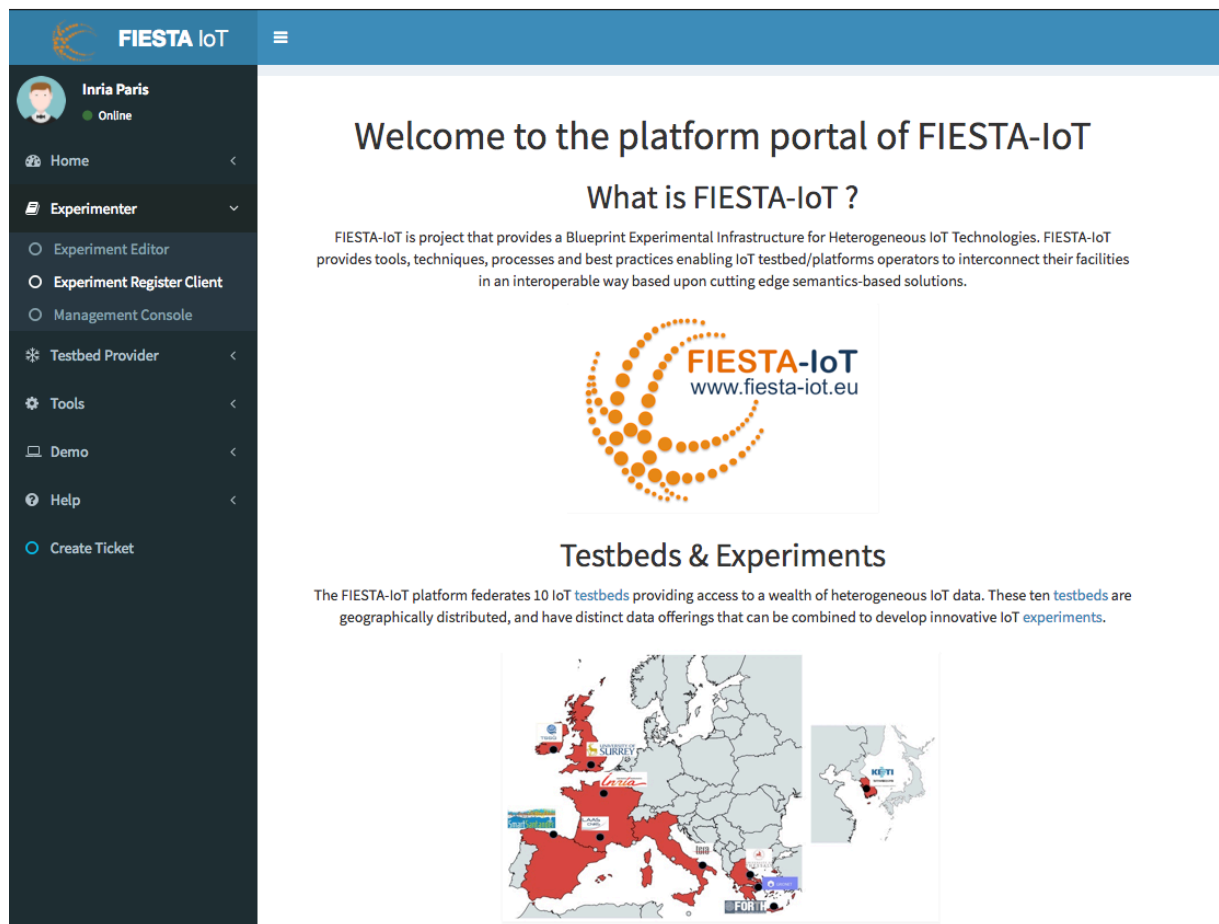


Figure 39: Portal Experimenter Menu

The Experiment Register Client provides the ability to store an experiment at the FIESTA-IoT platform in the form of a FEDSpec. The defined FEDSpec could be as simple as a single service (FISMO) or as complex as multiple experiments (FEMOs). To upload a FEDSpec first one should identify the location of it by hitting the “Open FEDSpec” (see Figure 40 below) and then by hitting the “Save FEDSpec” button. As soon as the FEDSpec is saved the included FEMOS appears in the available experiments list (FEMOS) as shown in Figure 40. When uploading a FEDSpec the FEMO/FISMO IDs should be empty, as they will be automatically assigned by the system.

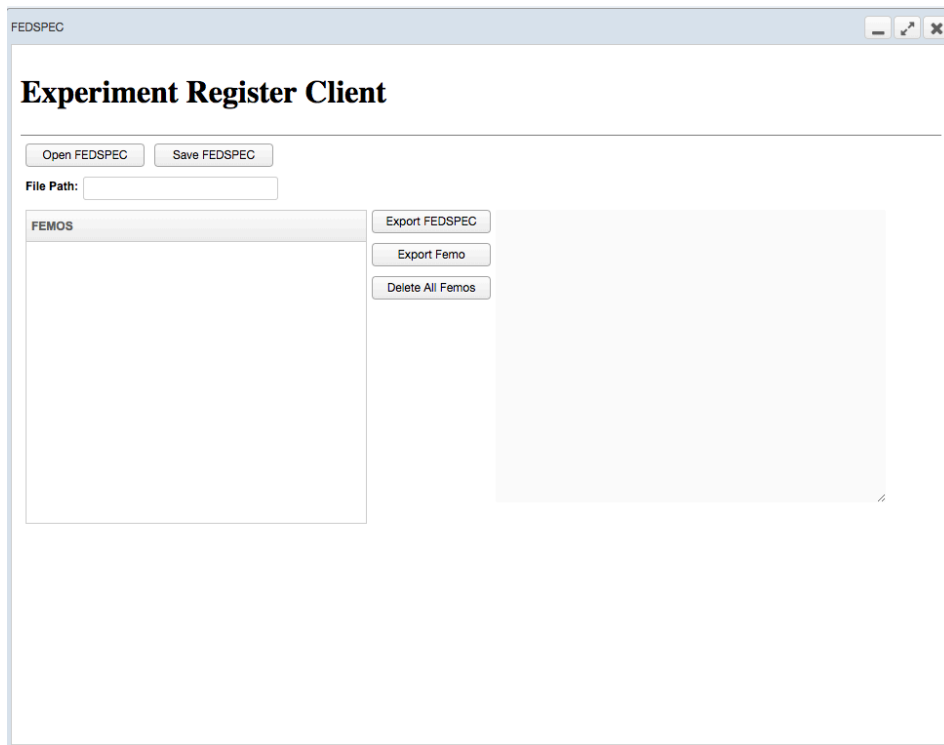


Figure 40: Experiment Register Client

By choosing a FEMO from the list, the User is capable to have a quick overview of it as shown in Figure 41 below.

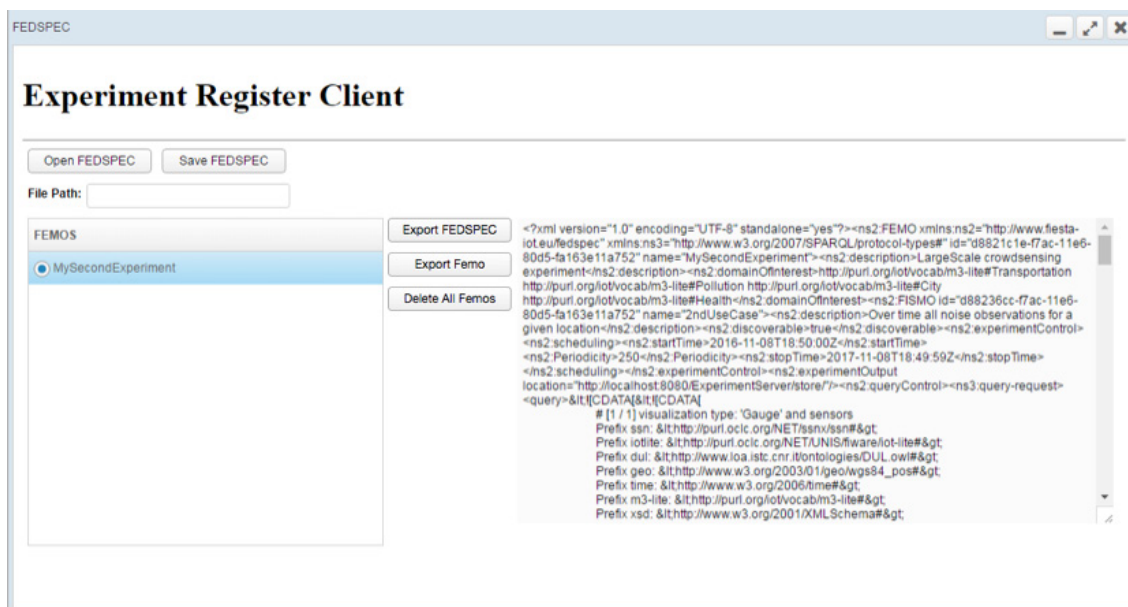


Figure 41: Experiment Register Client - Experiment Browser

The tool provides also the ability to export a FEMO by hitting the “Export FEDSPEC” button after choosing the FEMO of interest from the provided list. The FEDSpec that will be exported will now contain the FEMO/FISMO IDs assigned from the FIESTA-IoT platform. This will give the Experimenter the ability to update the exported

FEMO/FISMO by updating the XML file and saving it again to the Experiment Repository following the same process described above.

7.2.1.2.3 Execute Experiment

An experiment can be executed in many ways and FIESTA-IoT provides solutions for the execution of experiment for two categories of users (novice, advanced). Further, for the advanced user case FIESTA-IoT provides 2 solutions: one based on APIs of EEE and another one based on directly accessing IoT-Registry APIs. Novice experimenters are advised to use the method described in this section.

As said experiment execution is handled by a component called “Experiment execution Engine” or EEE. This module uses and supports the experiment description written by an experimenter in the DSL format specified by FIESTA-IoT (for reference on the DSL refer to [12]). Amongst the available features in the DSL, in the current version, EEE supports only a few. These include starting an experiment service (FISMO), pausing a FISMO, restarting a FISMO, subscribing to already existing and discoverable FISMOS, unsubscribing from subscribed FISMOS, and polling a FISMO (executing one time a FISMO on the FIESTA-IoT platform). The EEE specific APIs are available⁴ for developers or experimenters for testing and more in-depth knowledge about specific APIs. Note that in case an experimenter wants to use the EEE API they should still upload the FEDSpec either using the ERM API [12] or the ERM Client. Nevertheless, experimenters can also use Experiment Management Console and perform actions on the FISMO. This option is to be used by novice experimenters.

In order to execute an experiment using Experiment Management Console that is described by its FISMOS, the Experimenter first need to go to:

[https://platform.fiesta-
iot.eu/experimentConsole/experimentConsole.jsp](https://platform.fiesta-
iot.eu/experimentConsole/experimentConsole.jsp)

You can also use the cookie version of the console by just using the link above. Upon successful authentication, the list of experiments associated with the experimenter or the user is retrieved as shown in Figure 10. Note that this is also available via portal. The experimenter needs to go to the Experimenter Menu and click the “Experiment Management Console”

From this view, experimenters can then select whichever experiment they want to work on from the list using the “SELECT” button next to each experiment. Once a particular experiment is selected, this would open another UI (as shown from Figure 42 to Figure 44). The entire UI is divided into 3 panes: Experiment Details, Associated FISMOS, and Subscription Pane) where experiment name, experiment description, a list of experiment Domain of Interest along with Associated FISMOS and available FISMOS for subscription is shown.

An experimenter can choose to update the metadata of the experiment that he/she has created using “EXPERIMENT EDITOR”. This will open the UI provided in Section 5.3.1 where experimenter can resubmit their updated Experiments. Upon these

⁴ <https://platform.fiesta-iot.eu/EEEapidocs/>

resubmissions, a service in EEE is triggered that changes only the scheduling interval. If the scheduling interval is not changed nothing is updated on the EEE.

The “Associated FISMOS” pane shows the “meta” information about the FISMOS that are associated to a particular experiment. The “meta” information includes: if the FISMO was Owned or Subscribed within the frame of an experiment, the status of the FISMO (either NOT YET SCHEDULED, NORMAL, PAUSED, etc.), past execution history and polling option. Once scheduled a “delete job” button will appear that will let experimenters delete any reference of a particular FISMO from EEE. Upon deleting the FISMO will not be executed any more. In order to check for the description of the FISMO, experimenter can click on the name. This will open a snackbar in the bottom of the page and will show the description of the selected FISMO.

Initially, all the FISMOS have the NOT YET SCHEDULED status. If the experimenter wants to start the FISMO, they can switch the toggle button. Upon first toggle, the FISMO will be scheduled by the EEE with the NORMAL status. Another toggle would PAUSE the FISMO execution. Yet another toggle would restart the PAUSED FISMO. In order to successfully schedule the FISMO execution, the current version of the EEE supports all that is specified in Section 3.2.

A sample of <fed:scheduling> is provided below:

```
<fed:scheduling>
  <fed:startTime>2016-11-08T18:13:51.0Z</fed:startTime>
  <fed:Periodicity>600</fed:Periodicity>
  <fed:stopTime>2017-11-08T18:13:50.0Z</fed:stopTime>
</fed:scheduling>
```

The <fed:scheduling> would provide the EEE with the start date, end date and the periodicity of the FISMO execution. Thus making these attributes essential in the FISMO description. Once the schedule is set in the EEE, EEE provides a JOB ID that is used for internal purposes. This JOB ID is then provided with the status NORMAL. Upon the schedule, the <query> is read by the EEE from the FISMO description and is sent to FIESTA-IoT Meta-Cloud. The Meta-Cloud executes the query and sends back the results to the EEE. The EEE stores the result internally and pings the location specified in the location specified by the <fed:experimentOutput> (<fed:experimentOutput location=“location”/>). Upon success, the results are sent to the specified location and deleted from the internal repository. Currently, EEE assumes that the “location” here is a URL, where the specified credentials are granted to the EEE to write the results in a file. For reference and ease, a sample code that experimenters can execute on their server can be found in the following public repository⁵. It is thus noteworthy to state that currently EEE only supports one mechanism right now to send the information to the experimenter. Given the above, it is thus essential to specify <fed:scheduling> <experimentControl> attribute of FISMO, <query> under <prt:query-request> under <fed:queryControl> and <fed:experimentOutput location=“location”>. If the experimenter wants to just execute the FISMO and not to wait for the EEE to trigger the execution of the FISMO,

⁵ <https://github.com/fiesta-iot/experiment.data.receiver>

the experimenter can use POLL NOW. The POLL NOW will execute the <query> defined within the FISMO and would return the results to the same URL that is specified (i.e. the URL where results of scheduled execution are being sent).

Experiment Management Console

Experiment Details

Experiment ID and Name:	42e99dca-711f-11e7-b02b-fa163e11a752	JunkExperiment
Experiment Description:	Some Junk	
Experiment Domain of Interest List:	List Transportation	

You can download results of past executions of FEMO/FISMOs using <https://platform-dev.fiesta-iot.eu/experiment-result-store> for more details see training material.

Figure 42: Part 1: Experiment Detail Pane

Associated FISMOs

JobID	FISMO Name	Ownership	Status	Stopped/Started	
3779d063-f057-4b30-b7ec-e8e5ee3f14db	myUseCase	Owner	NORMAL	<input checked="" type="checkbox"/>	VIEW LOGS POLL NOW DELETE JOB
22a1b10c-16da-4f40-99b7-39ded784faa7	FirstFISMO	Subscribed	NORMAL	<input checked="" type="checkbox"/>	VIEW LOGS POLL NOW UNSUBSCRIBE

Figure 43: Part 2: Associated FISMOs Pane

Other available FISMO ID for Subscription

Available FISMOs (choose one)

2ndUseCase: Over time all noise observations for a given location

To Send data to: url

[VIEW SELECTED](#) [SUBSCRIBE](#)

Figure 44: Part 3: Subscription Pane

Nonetheless, apart from the above functionality, an experimenter can also subscribe to an already existing FISMO⁶. In case there are many FISMOs available, an experimenter can choose a particular FISMO from the dropdown list and provide the URL information (see Figure 44). Note that as EEE only support URL, experimenters must specify a valid endpoint. Only after validating the experimenter's URL the "SUBSCRIBE" button will be unlocked. The experimenter can currently only choose one FISMO at a time.

⁶ If there were no FISMOs available for subscription this part would be disabled.

Once successfully subscribed, the list of Associated FISMOS is updated to show the subscriptions. Each new subscription would provide a new JOB ID where the status of the JOB would be NORMAL to the subscribed FISMO and its execution would begin as the schedule specified in the description of that particular subscribed FISMO (see Figure 43). Moreover, the URL specified in the FISMO will not be used. Instead the URL specified by the subscriber would be used to forward the results. An experimenter, on demand, can unsubscribe the subscribed FISMO by clicking “UNSUBSCRIBE”. This will delete the JOB associated from the EEE.

An experimenter is also given a capability to see the details of past executions of the “Associated FISMOS”. The details are provided in the form of a graph and contains information like how much time did it take to execute the FISMO and how much data was obtained from the Meta-Cloud. This graph however does not show how much time did it take to execute the FISMO and how much data was obtained from the Meta-Cloud when the FISMO is polled.

In order to delete the experiment, it is advised that experimenters first stop/delete the execution of any related FISMO objects on the EEE using the EMC. Once this is done, they are advised to remove the experiment from the Experiment Registry Client. We acknowledge this workflow because this will give experimenters a view of what all services are running and if it is really required to remove them at all.

8 IMPLEMENTATION

In this section, we provide details of the installation procedures for the different components we have built.

8.1.1 Source Code Availability

In the first version of the document [1], we listed that FIESTA-IoT components are available on private Gitlab⁷. Nevertheless, FIESTA-IoT consortium members privately use Gitlab. A public version of the components is also available for the experimenters or testbeds for their use. The public versions of the components are available via Github⁸. Within Github FIESTA-IoT components that are provided are: ontology, TPI, sample experiment and Experiment Data Receiver.

8.1.2 Components

All of the described components are maven⁹ projects and are deployable within WILDFLY¹⁰ container. The Experiment Data Receiver however is the only component that currently only executes on Tomcat.

8.1.2.1 Experiment Editor

8.1.2.1.1 System Requirements

Table 12 lists the system requirements that are needed to build and deploy the Experiment Editor. Experiment Editor is built using Node¹¹. Once the component is successfully deployed its services can be accessed via `http://[HOST]:[PORT]/expeditor` where [HOST] is the host and [PORT] the port on which the Node is running.

Table 12: System Requirements for Experiment Editor

Requirements	Version
Node	v.6.xx

8.1.2.1.2 Dependencies

The Experiment editor requires certain dependencies that form the core of the component. These include those listed in the Table 13.

⁷ <https://gitlab.fiesta-iot.eu/platform/core/>

⁸ <https://github.com/fiesta-iot>

⁹ <https://maven.apache.org>

¹⁰ <http://wildfly.org/>

¹¹ <https://nodejs.org/>

Table 13: Dependencies for Experiment Editor

Requirements	Version
PM2	v.2.x.x

8.1.2.1.3 Install and Run

To install and run the experiment editor, the following steps should be followed in a chronological order. Note that to correctly install the Experiment Editor, no requirements or dependencies should be previously installed, as there might exist configuration issues.

Install Node JS

The documentation and package files required for the installation in any system can be found at <https://nodejs.org/en/>. To install on an Ubuntu machine, use either:

```
$ curl -sL https://deb.nodesource.com/setup_6.x | sudo -E bash -
```

or

```
$ sudo apt-get install -y nodejs
```

Install Production Process Manager for Node js ‘PM2’

PM2 is a production process manager for Node.js applications with a built-in load balancer. It allows applications to be kept alive forever, to reload them without downtime and to facilitate common system admin tasks. To install PM2 use:

```
$ npm install pm2 -g
```

Setup git access key set

Generation of the RSA key pair is needed to pull the Expeditor from fiesta-ui.git. Copy the result of public key and send it to the administrator to get access to the server and the key can be added to the ssh trust store. Following is an example of where a sample file can be placed/added

```
$ cat ~/.ssh/fiesta.expeditor.git.pub
```

Clone and Pull the source from git

Clone the Experiment Editor source code from git using:

```
$ git clone https://thyunkim@bitbucket.org/synctechoinc/fiesta-ui.git
```

Pull the Experiment Editor source code from git using

```
$ cd fiesta-ui
```

```
$ git pull
```

Installing Required Libraries and Start

After pulling the code from the git, the required libraries must be installed using:

```
$ npm install
```

```
$ bower install
```

Start Experiment Editor using

```
$ pm2 start npm --name expeditor -- start
```

Restarting the Experiment Editor

When code changes, the administrator should first pull from the git and restart pm2 as follows:

```
$ git pull
```

```
$ npm install
```

```
$ bower install
```

```
$ pm2 restart expeditor
```

Logs

The log file of the Experiment Editor can be also accessed using

```
$ cd ~/.pm2/logs
```

```
$ pm2 logs expeditor
```

8.1.2.2 Portal

8.1.2.2.1 System Requirements

Table 14 lists the system requirements that are needed to build and deploy the Portal. Once the portal is successfully deployed it can be accessed via `http://[HOST]:[PORT]/portalui` where [HOST] is the host and [PORT] the port on which the WILDFLY is running.

Table 14: System Requirements for Portal

Requirements	Version
Memory	>512MB
Wildfly	10.0.0

8.1.2.2.2 Dependencies

The Portal requires certain dependencies that form the core of the component. These include those listed in the Table 15.

Table 15: Dependencies for Portal

Requirements	Version
Java	Java version 8 or later
Spring boot	V1.5.8
Thymeleaf	V3

AdminLTE	V2.4
----------	------

8.1.2.2.3 Install and Run

Below are the commands that someone should use to build the portal on the FIESTA-IoT development machine, using maven:

```
$ ./mvnw -DskipTests=true -Pdev clean package
```

For building the portal on a test environment, one should use the command

```
$ ./mvnw -DskipTests=true -Ptest clean package
```

For building the portal on the production environment, one should use the command

```
$ ./mvnw -DskipTests=true -Pprod clean package
```

For running the portal on any environment. Targeting to be displayed at the “portalui” address, one should use the command

```
$ java -jar target/portalui.war
```

The portal should be deployed by uploading the portal war file via WILDFLY using Java 8 and WILDFLY 10.0.0 or a later version. The portal also saves logs on the portalui.log file.

8.1.2.3Experiment ResultSet Storage

8.1.2.3.1 System Requirements

The following Table 16 lists the system requirements that are needed to build and deploy the component on the WILDFLY container. Once the component is successfully deployed its services can be accessed via `http://[HOST]:[PORT]/experiment-result-store` where [HOST] is the host and [PORT] the port on which WILDFLY is running.

Table 16: System Requirements for ERS

Requirements	Version
Java JDK	1.8
Maven	2.x
MySQL Server Community Edition	5.x
Wildfly	10

8.1.2.3.2 Dependencies

The Experiment Result Store (ERS) requires certain dependencies that form the core of the component. These include those listed in the Table 17. To know the complete list we redirect the readers to the pom.xml of the component that is made available via:

<https://gitlab.fiesta-iot.eu/platform/core/tree/develop/modules/experiment/ers>

Table 17: Dependencies for ERS

Requirements	Version
javaee-web-api	6.0
org.restlet	2.3.9
org.restlet.jee	2.3.8
mysql-connector-java	5.1.22
jackson-databind	2.7.0

8.1.2.3.3 Install and Run

Below we list various steps that need to be performed in order to successfully install the component.

As the first step one has to setup a schema and table in the MySQL database. The following SQL script can be used to create it:

```
CREATE SCHEMA IF NOT EXISTS ers;
CREATE TABLE IF NOT EXISTS ers.experiments (
    USER_ID varchar(255),
    FEMO_ID varchar(255),
    JOB_ID varchar(255),
    TIME_STAMP varchar(255),
    EXPR_RESULT MEDIUMTEXT
);
```

The above script to generate the database in the MySQL can be found under the folder /WEB-INF/sql-scripts/create-expr-store.sql.

Once the database is setup, code properties need to be set up. There are two Properties files under /WEB-INF/config (a) db.properties: dedicated for database connection setting, and (b) global.properties: for global properties (note, this file is substituted with the common properties file (fiesta-iot.properties) for the core platform. The db.properties file consists of following properties:

```
#hostname for database
hostname = localhost
#port number for database
port=3307
#username for database
username = {username}
```

```
#password for database
#on first run a database will be created with the password set below
password = {password}
#path for database
#N.B: MAKE SURE DIRECTORY PATH IS ACCESSIBLE.
#for mysql use database name, e.g. "/s2w"
#for H2 use path and database name, e.g. "~//"
#path=/~/sdr/
name=test2
```

The `global.properties` file consists of following properties:

```
#hostname for OpenAM authentication Proxy.
hostname = {hostname}/openam/json/users?_action=idFromSession
```

Once the above is done, do the following to generate the WAR file:

```
$ cd <PATH TO EXPERIMENTRESULTSTORE>
$ mvn clean install
```

Once all these have been set and the WAR file generated, developers can deploy the WAR file on the WILDFLY container. The MySQL server instance should be running before the deployment is done. Once deployed the ERS services can be accessed at `http(s)://<HOST>:<PORT>/experiment-result-store` where [HOST] is the host and [PORT] the port on which WILDFY container is running.

8.1.2.1 Experiment Data Receiver

A sample code¹² is provided for experimenters to receive data provided by EEE.

8.1.2.1.1 System Requirements

The following Table 18 lists the system requirements that are needed to build and deploy the component on the experimenter's side. Once the component is successfully deployed its services can be accessed via `http://[HOST]:[PORT]/ExperimentServer/store/` where [HOST] is the host and [PORT] the port that Tomcat uses. The component is tested to be successfully executed on Tomcat.

Table 18: System Requirements for Data Receiver

Requirements	Version
--------------	---------

¹² <https://github.com/fiesta-iot/experiment.data.receiver/tree/master/ExperimentServer>

Java Platform, Standard Edition	1.8.0_25
Maven	3.1.1
Apache Tomcat	8

8.1.2.1.2 Dependencies

The Experiment Data Receiver requires certain dependencies that form the core of the component. These include those listed in the Table 19. (Note we do not list all the dependencies needed. To know the complete list we redirect the readers to the pom.xml of the component that is made available via <https://github.com/fiesta-iot/experiment.data.receiver/blob/master/ExperimentServer/pom.xml>):

Table 19: Dependencies for Data Receiver

Requirements	Version
xml-apis	1.4.01
Resteasy (jaxrs, jaxb-provider, html)	3.0.6.Final
portlet-api	2.0
servlet-api	2.5
jsp-api	2.1

8.1.2.1.3 Install and Run

To deploy, on the experimenter side following has to be done before the deployment

- in the web.xml change the location entry for multipart-config with the desired location

```
<multipart-config>
<location>#LOCATION#</location>
</multipart-config>
```
- In the
src/eu/fiesta_iot/experimentServer/ExperimentServerService.java
change the \$(LOCATION) to match the location set in the web.xml

```
File file = new File("${LOCATION}", fileName)
```

Note that this location is the desired location where you want to store the received files.
- Make sure that the #LOCATION# has read-write permissions to the Tomcat user and group (under the name and group Tomcat is running).
- In the Tomcat server change the following line in the conf/content.xml

```
<Context>      ...  </Context>
```

with the following

```
<Context allowCasualMultipartParsing="true">
...
</Context>
```

- restart Tomcat server

Once the above is done, do the following

```
$ cd <PATH TO EXPERIMENTSERVER>
```

```
$ mvn clean install
```

```
$ cp <PATH TO EXPERIMENTSERVER>/target/ExperimentServer.war <PATH TO
TOMCAT WEBAPPS>
```

Your service is running at `http(s)://<HOST>:<PORT>/ExperimentServer/store/` and thus your URLLOCATION should be

`http(s)://<HOST>:<PORT>/ExperimentServer/store/`

This will enable you to receive the resultsets that are generated after the execution of your FISMO. As also stated before the name of the file received follows a naming convention. Again, it is:

```
String filename = JOBID.replace("-", "")+URLLOCATION.replace(":",
 "").replace("/", "_")+ "_" +LONG_TIMESTAMP;
```

Note here JOBID is a UUID, URLLOCATION is the location that you provide and LONG_TIMESTAMP is a timestamp in long (milliseconds after epoch).

If the experimenter is using HTTPS, then they should use LetsEncrypt certificate for this API/URL. All other certificates other than those available in default JVM configuration will fail. This is because of the JVM does not have all the certificates installed. However, if the URL is HTTP, it will pass through

8.1.2.2 Experiment/Testbed Monitoring Tool

The Monitoring Tool is based on python and uses a Mongo DB to store data in an edited way. This is done to prepare the data for other analysis and also to visualize it in a proper way.

As the tool is not running in a WILDFLY container, it will be integrated into the portal via an iframe. For this the nginx server, that is serving the portal, is configured to map the monitoring tool into the namespace of the portal. The configuration also makes sure that it is only available via HTTPS. This integrates it also into the security framework, so that users cannot bypass it.

8.1.2.2.1 System Requirements

Following Table 20 list requirements for monitoring tool for the correct execution.

Table 20: System Requirements for Experiment/Testbed Monitoring Tool

Requirements	Version
python	2.7.x
python-virtualenv	1.11.4
mongoDB	>2.4

8.1.2.2.2 Dependencies

Following Table 21 list python dependencies that are needed for monitoring tool for the correct execution

Table 21: Dependencies for Experiment/Testbed Monitoring Tool

Requirements	Version
Flask	0.12.2
Flask-RESTFUL	0.3.6
PyYAML	3.12
pymongo	3.5.1
gevent	1.2.2
requests-futures	0.9.7
python-dateutil	2.6.1

8.1.2.2.3 Install and Run

In the following section, we list all necessary installations and configurations that should be performed.

- **Python and virtualenv creation**

The monitoring tool is based on python and so needs a python environment. This is shipped and preinstalled in all major Linux distributions. As the tool is using several python modules that are installed via pip, a virtualenv is used to isolate the needed modules and to not interfere with the modules that are installed system-wide.

To create a virtualenv, the package python-virtualenv is needed:

```
$ sudo apt install python-virtualenv
```

Then the virtualenv can be created:

```
$ virtualenv ${HOME}/.virtualenv/testbed-monitoring
```

To activate the virtualenv, either it has to be activated or its python binary can be directly used to run a python file.

```
$ source ${HOME}/.virtualenv/testbed-monitoring/bin/activate
```

The dependencies need to be installed in the virtualenv:

```
(testbed-monitoring)$ pip install -r  
${TESTBED_MONITORING_HOME}/requirements.txt
```

- **mongoDB**

The monitoring tool uses a Mongo database in order to store the extracted data in an edited way. The data will be transformed and all not needed parts will be removed.

To install mongoDB:

```
$ sudo apt install mongodb
```

- **Upstart**

To control the monitoring tool as a service, an upstart script is used. It can be invoked to start and stop the system and also to enable the automatic start of the system.

The Upstart script that is used:

```
description "Testbed Monitoring"  
start on runlevel [2345]  
stop on runlevel [016]  
setuid ubuntu  
setgid ubuntu  
script  
    export HOME=/home/ubuntu  
    cd ${HOME}/fiesta-tools/testbed-monitoring  
    exec ${HOME}/.virtualenv/testbed-monitoring/bin/python run.py  
end script
```

The testbed monitoring is installed under /home/Ubuntu/fiesta-tools/testbed-monitoring. The run.py file, which is the start file will be invoked directly with the python binary from the virtualenv.

- **Nginx configuration**

As mentioned before, to enable the monitoring tool to be accessible in the portal, some changes in the nginx configuration are necessary. The monitoring tool is configured to listen to port 4000 for HTTP connections. The basic namespace is /dashboard. So, the nginx configuration needs to map in its HTTPS configuration the /dashboard namespace to the Monitoring tool by using proxying.

The relevant entry inside of the /etc/nginx/sites-enabled/default file:

```
location /dashboard {  
    proxy_set_header    X-Real-IP $remote_addr;
```

```
proxy_set_header    Host        $http_host;
proxy_pass           http://127.0.0.1:4000;
proxy_read_timeout  90;
}
```

It will simply pass all URIs starting with dashboard directly to the Monitoring Tool.

- **Monitoring Tool configuration**

The Testbed Monitoring Tools config file, which can be found under {TESTBED_MONITORING_HOME}/config.yml, is a yaml file, which can be configured in an easy way:

```
monitoring:
  iot_registry: http://localhost:8080/iot-registry/api
  testbeds_update_time: 120 # minutes
  sensors_update_time: 120 # minutes
  observations_update_time: 10 # minutes
  observation_time_span: 7 # days
  max_query_span: 1 # days
web:
  host: 0.0.0.0
  port: 4000
  overall_duration: 1
db:
  host: localhost
  port: 27017
  db_name: monitoring
  drop: False
```

In the monitoring section the tool itself can be configured, web is for adapting the web server and db is used to configure the access to the mongoDB.

The monitoring section has the URI to the internal port of the IoT-Registry. The fields *_update_time are to configure the interval of the internal tasks to query the IoT-Registry. The field observation_time_span is to limit the maximum days the Testbed Monitoring Tool will store observations for each sensor. The field max_query_span is used to limit the maximum query range of the IoT-Registry to not ask for too much data.

9 CONCLUSION

This is the last deliverable with respect to the tasks within WP4. This deliverable reports advancements done within Task 4.4 and Task 4.5 and updates that were performed to [1]. Via this deliverable, we provide our advancements with respect to how experimenters could create, deploy and manage experiments, giving as well an overview about the FIESTA-IoT portal with respect to experimenters. Note that the portal is not only limited to the tools that are applicable to experimenters but it also supports tools available for testbed owners (some of which are presented in [15]). Further, other user roles defined within FIESTA-IoT framework would also use the FIESTA-IoT portal.

This deliverable mainly reports issues identified by the reviewers and provides new tools that were developed. Nonetheless, the EEE, EMC and Portal were updated to support new functionalities, APIs and tools to help experimenters achieve their goals. The updates mainly relate to inclusion of new accounting API within EEE, more restricted APIs now being public, and revamped UI for EMC and Portal. Other than the updates to the afore-mentioned tools, within this deliverable, new tools such as: Experiment editor using which experimenters can create configuration/DSL for EEE, Experiment/Testbed monitoring tool using which experimenters can monitor the status of the testbed etc., Experiment Data receiver using which experimenters can receive the resultset, Experiment Result store using which experimenters can download previously available resultset are also reported.

It is worth mentioning that the provided/discussed tools will be updated on need basis after analyzing the requirements, if any, from the Open Call/other participants. Of course, continuous support, integration and bug fixing will be inevitably part of it. As the tools are also available to public, these tools are well documented and the APIs within are supported by the documentation where the experimenters can possibly execute the APIs if they have the right credentials.

REFERENCES

- [1] FIESTA-IoT, “Deliverable 4.7: Infrastructure for Submitting and Managing IoT Experiments,” 2017.
- [2] FIESTA-IoT, “Deliverable 2.1: Stakeholders Requirements.”
- [3] FIESTA-IoT, “Deliverable 2.3: Specification of Experiments, Tools and KPIs.”
- [4] FIESTA-IoT, “Deliverable 5.1: Experiments Design and Specification.”
- [5] FIESTA-IoT, “Deliverable 5.2: Experiments Implementation, Integration and Evaluation,” 2017.
- [6] FIESTA-IoT, “Deliverable 3.6: Concept and Development for IoT Data Analytics and IoT Stream and Service Management,” 2017.
- [7] FIESTA-IoT, “Deliverable 4.6: Tools and Techniques for Managing Interoperable Data sets,” 2017.
- [8] FIESTA-IoT, “Deliverable 4.1: EaaS Model Specification and Implementation,” 2016.
- [9] FIESTA-IoT, “Deliverable 2.4: FIESTA-IoT Meta Cloud Architecture,” 2015.
- [10] FIESTA-IoT, “Deliverable 4.4: Authentication, Authorization, Data Protection and Reservation of Resources V2,” 2017.
- [11] A. H. Soukhanov, K. Ellis, and M. Severynse, *The American Heritage Dictionary of the English Language*. Boston: Houghton Mifflin, 1992.
- [12] FIESTA-IoT, “Deliverable 4.2: EaaS Model Specification and Implementation,” 2017.
- [13] FIESTA-IoT, “Deliverable 3.1: Semantic models for testbeds, interoperability and mobility support and best practices,” 2016.
- [14] FIESTA-IoT, “Deliverable 6.2: Certification suite V1,” 2017.
- [15] FIESTA-IoT, “Deliverable 3.3: Specification and implementation of common Testbed interfaces,” 2016.
- [16] FIESTA-IoT, “Deliverable 3.4: Specification and implementation of common Testbed interfaces,” 2017.