Title:	Document Version:
D6.3 Flexible and scalable integrated platform v1	1.0

Project Number:	Project Acronym:	Project title:
H2020-863927	X-Flex	LC-SC3-ES-1-2019 – Flexibility and retail market options for the distribution grid.

Contractual Delivery Date:	Actual Delivery Date:	Deliverable Type* Security**:
M24 (September 2021)	M24 (September 2021)	R-PU

*Type: P: Prototype; R: Report; D: Demonstrator; O: Other.

**Security Class: PU: Public; PP: Restricted to other programme participants (including the Commission); RE: Restricted to a group defined by the consortium (including the Commission); CO: Confidential, only for members of the consortium (including the Commission).

Responsible:	Organisation:	Contributing WP:
Diego García-Casarrubios Alberto Zambrano	ETRA	WP6

Authors (organisation):

Jurij Bizjak (Petrol), Jure Bartol (Petrol), Bojan Stojanović (Petrol), Dušan Krovinović (Petrol), Othonas Zacharias (SUITE5)

Abstract:

This deliverable provides the technical details of the implementation of the different modules of X-FLEX's flexible and scalable integrated platform, i.e., X-FLEX Platform. This document is a companion report to the implementation of the first prototype of the platform. The document pretends to be both informative for the general public and useful for developers that would like to integrate with the platform.

Keywords:

Platform, prototype, middleware, flexible, interoperable, scalable, integrated, CITRIC, microservices, architecture, modular, layer.

Revision History

Revision	Date	Description	Author (Organisation)
0.1	15.06.2021	Table of Contents	Diego Garcia-Casarrubios (ETRA)
0.2	09.07.2021	Platform APIs description	Alberto Zambrano (ETRA)
0.3	28.07.2021	Internal modules description	Alberto Zambrano (ETRA)
0.4	20.08.2021	All sections completed	Diego Garcia-Casarrubios (ETRA)
0.5	30.08.2021	Ready for peer review	Alberto Zambrano (ETRA)
0.6	10.09.2021	Internal Peer review	Anton Zvonko Gazvoda (UL), Chloe Fournely (UL), Grigoris Kanellos (HEDNO), Maria Symponi (HEDNO), Tadej Smogavec (Petrol), Polona Štumpfl Erjavec (Petrol), Jurij Gerbec (Petrol), Anton Vučko (Petrol), Jurij Bizjak (Petrol), Jure Bartol (Petrol), Bojan Stojanović (Petrol), Dušan Krovinović (Petrol), Konstantinos Tsatsakis (SUITE5), Zacharias Othonas (SUITE5), Denitsa Kuzeva (ALBENA)
1.0	25.09.2021	Final version ready for submission	Alberto Zambrano (ETRA), Diego Garcia-Casarrubios (ETRA), Lola Alacreu (ETRA)

This project has received funding from the

European Union's Horizon 2020 Research and Innovation Programme

Under Grant Agreement Nº 863927

More information available at https://xflexproject.eu

Copyright Statement

Executive Summary

The deliverable D6.3 "Flexible and scalable integrated platform v1" provides the technical details of the implementation of the different modules of X-FLEX's flexible and scalable integrated platform. Specifically, this document includes the details of the first prototype of the tool, more details about the final version of the prototype will be included in D6.4 "Flexible and scalable integrated platform v2", to be submitted in month 36. The document pretends to be both informative for the general public and useful for developers that would like to integrate with the platform.

The development of the X-FLEX Platform is built upon the work performed in WP2, especially T2.2 "Use cases and requirements definition", T2.4 "System architecture", and T2.5 "KPI identification and monitoring preparation". Previous work carried on in WP6 has been the basis for the implementation of the services and the general architecture of the platform: T6.1 "Standards and data models analysis" extracted the required information in order to identify the standards and data models that the platform needed to implement and/or integrate with, while T6.2 "Design of the X-FLEX Flexible and scalable integrated platform" defined the architecture of the tool and the different modules that form it and how they communicate among them.

It is important to highlight that the platform is not built from scratch. The use of a central ESB for communication and horizontal functionalities is frequent in H2020 ecosystems, especially in the field of Energy. Having several years of experience in this kind of projects, the X-FLEX Platform is built upon previous work with similar requirements.

The implementation of the platform is modular, based on the microservice paradigm: a set of diverse, independent services focused on a single or a reduced set of specific well-defined functionalities that communicate among them to share data, events, and Services, contrary to the classical approach of a single, monolithic application that manages every aspect of the system, the microservice paradigm enhances the scalability and the resilience of the system.

The different modules of the platform have been detailed from a technical perspective. When available, documentation on how to use these modules has been provided, so the document is not purely theoretical but also serves to developers using the tool when integrating their services and systems with the X-FLEX Platform. The same approach will be used for D6.4 "Flexible and scalable integrated platform v2" for those modules whose implementation has not been covered in the present deliverable. Any possible modification, correction, or extensions to the modules in the first prototype that may emerge from the first round of tests will be included in v2 deliverable as well.

Index

List of Figures

List of Tables

1 INTRODUCTION

1.1 Purpose of the document

The present document provides the technical details of the implementation of the different modules of X-FLEX's flexible and scalable integrated platform, i.e., X-FLEX Platform. This document is the companion report to the implementation of the first prototype of the tool. The document intends to be both informative for the general public and useful for developers that would like to integrate with the platform.

1.2 Scope of the document

The development of the X-FLEX Platform is built upon the work performed in WP2, especially T2.2 "Use cases and requirements definition", T2.4 "System architecture", and T2.5 "KPI identification and monitoring preparation". Previous work carried on in WP6 has been the basis for the implementation of the services and the general architecture of the platform: T6.1 "Standards and data models analysis" extracted the required information in order to identify the standards and data models that the platform needed to implement and/or integrate with, while T6.2 "Design of the X-FLEX Flexible and scalable integrated platform" defined the architecture of the tool and the different modules that form it and how they communicate among them.
The present deliverable describes the first prototype of the tool. The second and final version of the X-FLEX Platform is expected to be reported in D6.4 "Flexible and scalable integrated platform v2" in M36. The functionalities of the tool will be subsequently demonstrated and evaluated as part of WP7 and WP8 respectively.

1.3 Structure of the document

The rest of the document is structured as follows:

- Section 2 presents the foundations of the tool, including relation with other H2020 projects, an overview of the architecture of the platform, and a mapping of the associated requirements from WP2 to the elements of the tool that validate them.
- Section 3 describes the implementation of each of the modules of the tool, including the technical details and documentation on how to use them (when applicable). The services are grouped in three main subsections: internal modules, horizontal modules, and interfaces and wrappers.
- Section 4 provides details on the infrastructure where the tool will be deployed and tested.
- Section 5 concludes the report with a summary of the content and the next steps towards the second version of the X-FLEX Platform

2 X-FLEX PLATFORM FOUNDATIONS

2.1 Relation to other projects

2.1.1 Background

The development of the X-FLEX Platform has not been performed from scratch, but rather built upon the results from previous projects where similar functionalities were required. The basic functionalities of a middleware or an Enterprise Service Bus (ESB) are usual in IT projects, and especially H2020 projects in the field of energy. In order to cover these necessities, ETRA has been developing and extending a platform (known as CITRIC) that will serve as a foundation to build the X-FLEX Platform.

CITRIC is a city platform created by ETRA that follows the RIVER architecture: a Reactive, Interoperable, Visible, Elastic and Resilient architecture oriented to microservices and events. It is an open architecture with capacity to grow its service network, reactive because it is event-driven, interoperable because it is supported by standard protocols and agnostic models of data, visible because it is monitored in its operation, elastic because it can be scaled out and independently in each of its services to support any magnitude of city, and resilient because it is orchestrated and monitored to be fault tolerant. CITRIC's microservices distribution fully complies with norms IEC30182:2017 [1] and UNE178104:2017 [2] in its orientation towards functional layer.

The foundations, basic functionalities, and specific services have been developed as part of the H2020 projects ASSISTANCE [3], SAURON [4], and MATCHUP [5].

2.1.2 Main functionalities

The architecture of the CITRIC platform is presented in the Figure below:

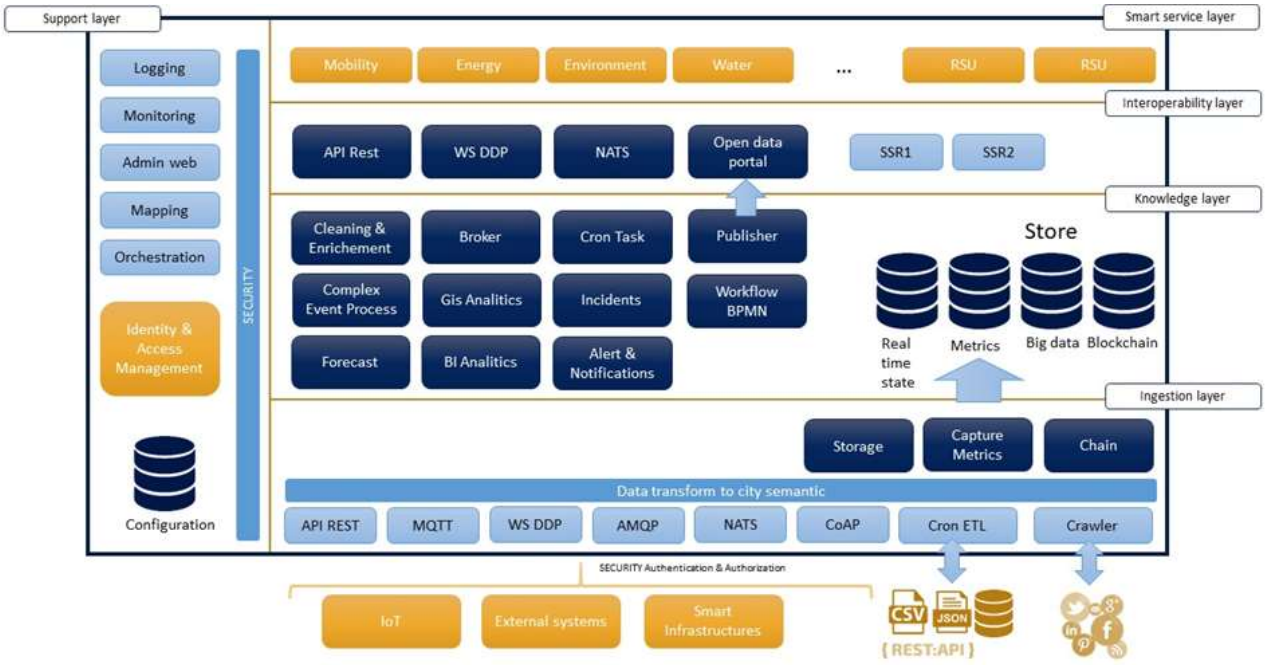

Figure 1 – CITRIC architecture

The general CITRIC architecture is formed by modules (or microservices) grouped by functionality in a set of layers:

- **Support layer**
  - **Orchestration:** The orchestration of all components is performed by Docker Swarm [6] or Kubernetes [7] allowing flexible and monitored deployment.
  - **Monitoring:** The CITRIC management website allows to monitor all the microservices graphically, as well as the activity of the data flowing through the platform.
  - **Logging:** The entire logging system is handled by the orchestrator and stored for forensic analysis of errors and alerts.
  - **Admin web:** CITRIC offers a multilingual web interface for management and monitoring with presentations of metrics, mapping, incidents, data, multimedia, etc.
  - **Mapping:** A 3D map is available in the management interface to navigate platform data. This map allows you to make analytic representations of the data such as aggregations of means, maximums, deviations, etc., apart from the geolocated representation of the elements. The visualizer is optimized for bulk rendering of web data using WebGL [8] technology.
  - **Identification and access management:** CITRIC has a component for authentication, authorization, and profiling that allows to determine how the platform is accessed under particular credentials. Use basic authentication, as well as SSO protocols OpenID [9] and SAML [10]. It can be federated with a client LDAP [11] server. All this functionality is supported by a Keycloak [12] backend.
  - **Configuration:** All CITRIC configuration resides in a NoSQL database, particularly MongoDB [13]. It stores all the information necessary for the operation of all its components.
- **Ingestion layer**
  - **API REST:** CITRIC has a robust secure and protected HTTPS Rest API with rate limit control to prevent attacks and allows a lot of flexibility for data ingestion through it. Besides authentication, API security allows to authorize only a certain set of data to each authenticated user. The implementation of this API has been done with Express [14].
  - **MQTT:** The MQTT [15] protocol is widely used by lightweight devices (IoT) and supported by CITRIC. For this purpose, CITRIC uses RabbitMQ [16] as the MQTT ingestion service.
  - **AMQP:** It is possible to ingest data directly with AMQP [17] protocol to ingest directly over queues that are consumed by microservices, which transform and store information in the platform storage system.
  - **WS DDP:** It is possible to connect to the platform using websockets (e.g., from web browsers). The protocol over WS implemented is Distributed Data Protocol (DDP) [18]. A number of methods are implemented over this protocol that provide a very efficient way to do mass ingestion.
  - **NATS:** NATS [19] is the underlying broker on the platform and is responsible for managing the entire microservices communication network. It acts as an enterprise service bus (ESB); communication through it makes use of topics that are mapped to services. CITRIC offers a set of topics that allow to interact directly with storage microservices with their corresponding level of security.
  - **CoAP:** CoAP [20] is a lightweight protocol similar to MQTT but communicating over UDP. It is normally used by moving devices that jeopardize the quality of communication. CITRIC offers this ingestion service to integrate data coming from moving vehicles that interact with the platform.

- **Cron ETL:** This service is responsible for ingesting data offered in external files, databases or web services. The process can be triggered in a time-basis or by modifying or creating new ready-to-load files on the platform. As an ETL tool, CITRIC uses Node-RED [21], an interactive tool to design workflows to ingest data with possible transformation.
- **Crawler:** Natural language processing service on social networks such as Twitter and Instagram for sentiment analysis against a linguistic corpus defined in the platform settings. This service scans networks by searching for keywords or hashtags and analyses the context to determine the "meaning" of the same and generate these semantic inputs to the platform. This microservice is natively developed with Python and natural language processing frameworks such as spaCy [22].
- **Transform:** Any ingestion process previously goes through a transformation process to normalize the data before entering it into the platform. Transformation schemas are pre-configured for each source in the configuration database and are particular to each platform deployment as they must be tailored to particular data sources and how they are stored in the storage service and then served to the higher layers.
- **Security:** Every ingestion process is authenticated and authorized by a layer of security in each microservice. Each credential per token or user/password has an authorization scheme to access a subset of platform data at three levels of security: read, write, and only public attributes.
- **Storage:** This microservice handles the storage of data when it is injected or modified. It manages the real-time database supported by MongoDB and the big data database, supported in our case by an Elasticsearch [23] cluster.
- **Metrics:** This microservice extracts the precise information to create KPIs from each type of modelled data. The information is stored in a time-oriented database such as InfluxDB [24] or Elasticsearch. The choice of one or the other depends on the magnitude of the KPIs to be stored.
- **Chain:** This microservice captures data transactions that are required to be stored on a blockchain to ensure immutability and data distribution. CITRIC uses as blockchain BigchainDB [25], which guarantees the three precepts of immutability, encryption, and distribution, and also speed in transactions.

- **Knowledge layer**
  - **Cleaning & Enrichment:** ETL batch process for cleaning and enriching the information stored in the big data repository. CITRIC makes use of an ETL + Analytics tool such as KNIME Analytics Platform [26] for data cleansing and enrichment of data stored in the big data repository.
  - **CEP Complex Event Process:** Streaming event processing supported by the versatile Siddhi [27] processor with its own scripting language that allows to define pipelines and rules, as well as temporary processing windows. It is the processor used by Uber with great scaling capacity.
  - **Forecast:** This microservice is responsible for generating Machine Learning models on data collections stored in the storage repository. Makes use of Facebook's Prophet [28] framework. This microservice creates models and makes predictions about data series. It offers NATS interaction to integrate with the entire CITRIC ecosystem and a Rest API.
  - **Broker:** The entire microservices architecture is supported by a robust and scalable messaging broker such as NATS. This key, fault-tolerant and redundant element is the one that sorts the entire service network on the platform.

- **GIS Analytics:** Different utilities are available for geospatial data analysis: CITRIC's own management interface, the geographic components available on dashboards, and for more sophisticated analysis Kepler.gl is used.
- **BI Analytics:** Kibana and ElasticSeach are used as storage for data analytics, dashboard generation, and data visualizations. Kibana is a powerful environment for designing dashboards and canvas graphics mixed with KPIs for an effective way to represent metrics.
- **Cron Task:** This microservice serves a job planning configuration. Using the stored configuration, it performs work on a schedule. Jobs consist of launching orders through the broker or calls to web services for other microservices to perform tasks. For example, schedule the recreation of machine learning models by the Forecast microservice. Schedule enrichment of geolocated data with postal address, upload data from external databases, directory files, etc.
- **Incidents:** This microservice is responsible for observing incidents that occur in the installation (e.g. events happening on a pilot site); it has a complex notification system with different means such as Twitter, Telegram, Email, external systems by web services, or triggering of workflow processes that assist in the development of the incident.
- **Alert and Notifications:** Alerting and reporting events in the installation is done by various means. Like incidents, notifications are integrated with different means such as Twitter, Telegram, Email, external systems by web services, or triggering of workflow processes. In the real-time data analytics part, specifically in the metrics and big data aspect, ElasticSearch offers an entire alerting service when the data is outside of the scheduled thresholds.
- **Workflow BPMN:** When complex events or incidents are forecasted, a BPMN process that meets its resolution by users can be triggered. CITRIC integrates with Flowable [29] to cover this functionality.
- **Publisher:** This microservice is responsible for generating the datasets that will be published in the organisation's transparency and open data portal.

- **Interoperability layer**
  - **API Rest:** CITRIC has a portion of its REST API for CRUD operations that verticals can use to interact with the platform, with its corresponding level of security.
  - **WS DDP:** In the same way, customers can make use of websockets to interact with the platform. This service is especially useful for web applications, as websockets are the only way to interact with external systems with bidirectional flow. CITRIC distributes an NPM package that encapsulates the entire protocol and makes it very easy to design web clients in JavaScript to do developments with the platform.
  - **Open data portal:** For publishing open data in the form of datasets, CITRIC integrates with CKAN [30] to keep a collection of datasets fresh and available to the community.
  - **SSR plugins:** CITRIC has a reverse integration mechanism, so that verticals can be integrated into the CITRIC management interface to display particular dialogues or actions of verticals. This is done using the rendering technique on the SSR server. CITRIC provides documentation for third parties to create information rendering microservices as plugins and to be integrated into the management interface itself.
- **Smart service layer**
  - Includes any vertical service or application using the platform and its services. These external agents are usually grouped by its field of knowledge (e.g. Mobility, Energy, Environment), which defines the type of data (entities, metrics, etc.) the application will use.

The CITRIC platform has the ability to federate with other CITRIC instances in other deployments, so that applications or verticals interoperating with the platform would be able to "read" information from other deployments or platforms. This is thanks to the concept of *supercluster* supported by the messaging broker that underlies the core of the platform. CITRIC's internal messaging broker can be clustered. It can also be configured so that certain message patterns "travel" outside, to be handled by other clusters or nodes.

2.1.3 Modifications and new modules

In the previous section, an overall vision of the architecture of the CITRIC platform was described. In the table below, the subset of modules and services from the platform that will be used in X-FLEX is presented, including their associated development as part of the project. Additional modules not present in the general CITRIC architecture that will be implemented as part of X-FLEX are included as well.

Table 1 – CITRIC modules implemented and/or used in X-FLEX

Layer	Module	Status and development
Support layer	Orchestration	Kubernetes and Docker Swarm technologies will be used to deploy and orchestrate the different modules of the X-FLEX Platform. New modules will follow the proper microservice architecture requirements to allow scalability and replicability.
	Monitoring	New modules will implement the necessary functionalities to report their status to this service.
	Logging	New modules will implement the necessary functionalities to log their most relevant events and report them to this service.
	Admin web	The graphical user interface of the platform will be adapted and augmented as part of X-FLEX, including refinement of current functionalities and translation into English.
	Mapping	The 3D map used in the platform will be nourished with data coming from field assets and external systems. This functionality will complement that of the X-FLEX tools related to georepresentation.
	Identification and access management	Every external system to be connected to the platform will be provided with the proper authentication and authorization settings. A restrictive policy will be applied, meaning that each agent will only be able to access the minimum set of data required for their correct functioning.
	Configuration	The platform will be properly configured using its administration web.
Ingestion layer	API REST	These modules will be used for data ingestion from field assets, external systems, and third-party services. No major development is foreseen.
	MQTT	
	AMQP	
	WS DDP	

	NATS	This module will be used mainly for internal communication in the data ingestion process. No major development is expected.
	KAFKA	Integration with KAFKA has been built from scratch for X-FLEX. It will be used mainly by SERVIFLEX.
	Transform	The core modules for data ingestion in the platform are in an advanced stage and will be used in X-FLEX. They will be refined, corrected, and extended when deemed necessary.
	Security	
	Storage	
	Metrics	
Knowledge layer	Cleaning & Enrichment	The cleaning and enrichment of data—including bad data detection—and the integration with the CEP will be developed as part of the second version of the X-FLEX Platform.
	CEP Complex Event Process	
	Forecast	The forecast module, which will be used by several tools and pilots in the project, has been implemented from scratch for X-FLEX.
	Broker	The NATS broker will be used for internal communication among the services of the knowledge layer of the platform.
	Cron Task	Its use in X-FLEX is currently under analysis.
	Incidents	These two modules combined will be the basis for the end-user notification functionalities of some X-FLEX tools. The services, currently in beta version, will be evolved and adjusted as part of X-FLEX project.
	Alert and Notification	
	Workflow BPMN	The module will be integrated as part of X-FLEX.
	Virtual sensors	New module developed in X-FLEX. Periodically generates and broadcasts data from “virtual” entities, calculated from existing or newly received measurements from real sensors.
Interoperability layer	API Rest	Already in an advanced stage, these modules will be used by some tools in X-FLEX to retrieve the data stored in the platform.
	WS DDP	
	KAFKA	Access to the data in the platform using KAFKA has been developed from scratch in X-FLEX.
Smart service layer	GRIDFLEX	Tools in the X-FLEX ecosystem will use the platform to different degrees—from a simple communication hub to a fully fledge system for data ingestion, storage, analysis, and actuation.
	SERVIFLEX	
	MARKETFLEX	

2.1.4 Instantiation and configuration

CITRIC can be deployed over Docker Swarm nodes or over Kubernetes cluster based on the magnitude of the deploymnet’s required computing capacity. It can be therefore easily deployed in the Google, AWS, or Azure clouds.

All microservices are scaled out using replicas and load balancers based on the particular needs of each installation. An overview of the configuration of each module is provided in the corresponding subsections of Section 3. Details about the actual environment that will be used in X-FLEX are provided in Section 4.

2.2 Architecture overview

In D6.2 "Architecture of the flexible and scalable integrated platform", the architecture of the X-FLEX platform was defined. In the present section, the different layers and elements in this architecture are mapped to the ones of CITRIC, presented in the previous subsections.

The following figures highlight with colours different parts of both X-FLEX Platform and CITRIC architectures:

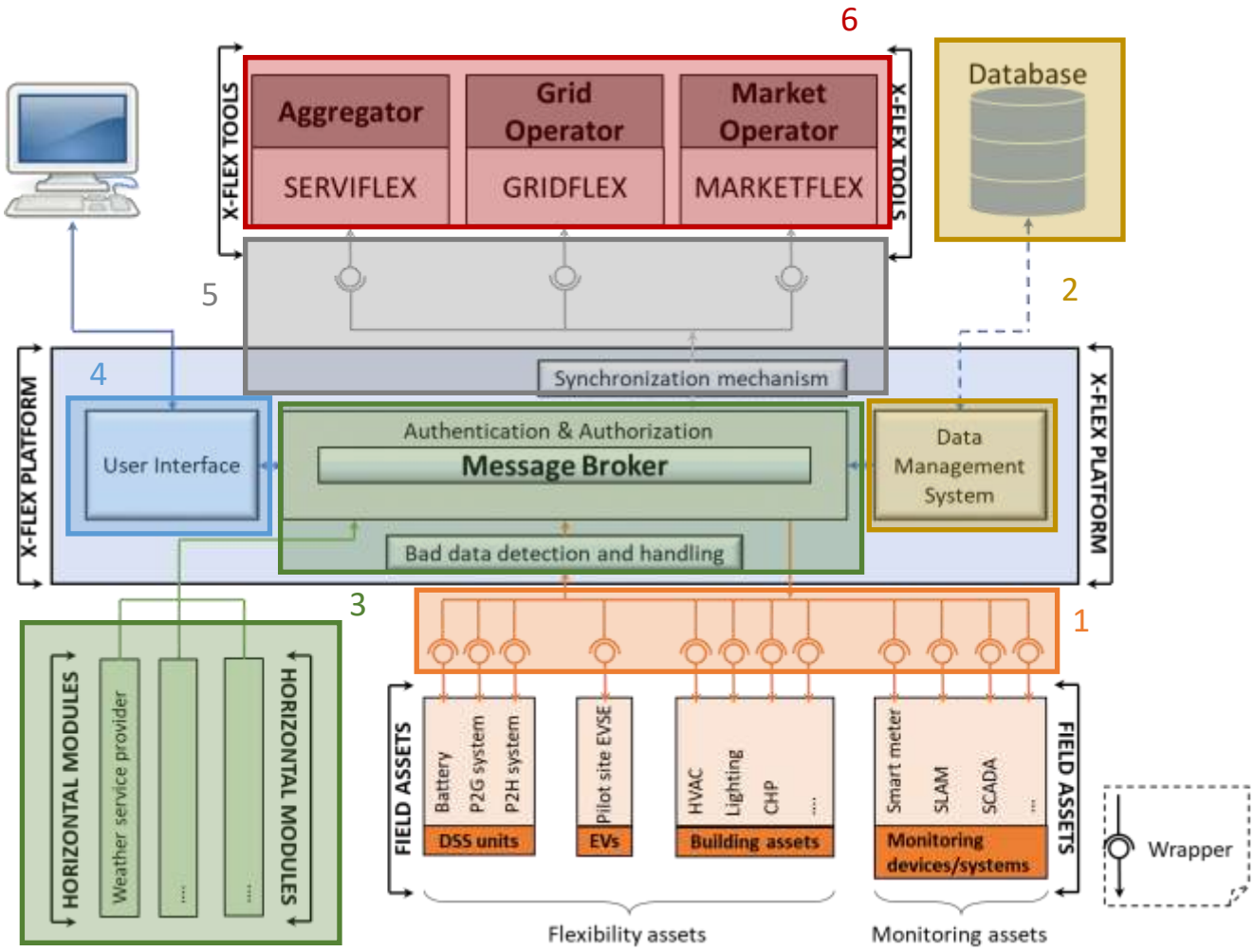

Figure 2 – X-FLEX platform architecture (with highlighted modules)

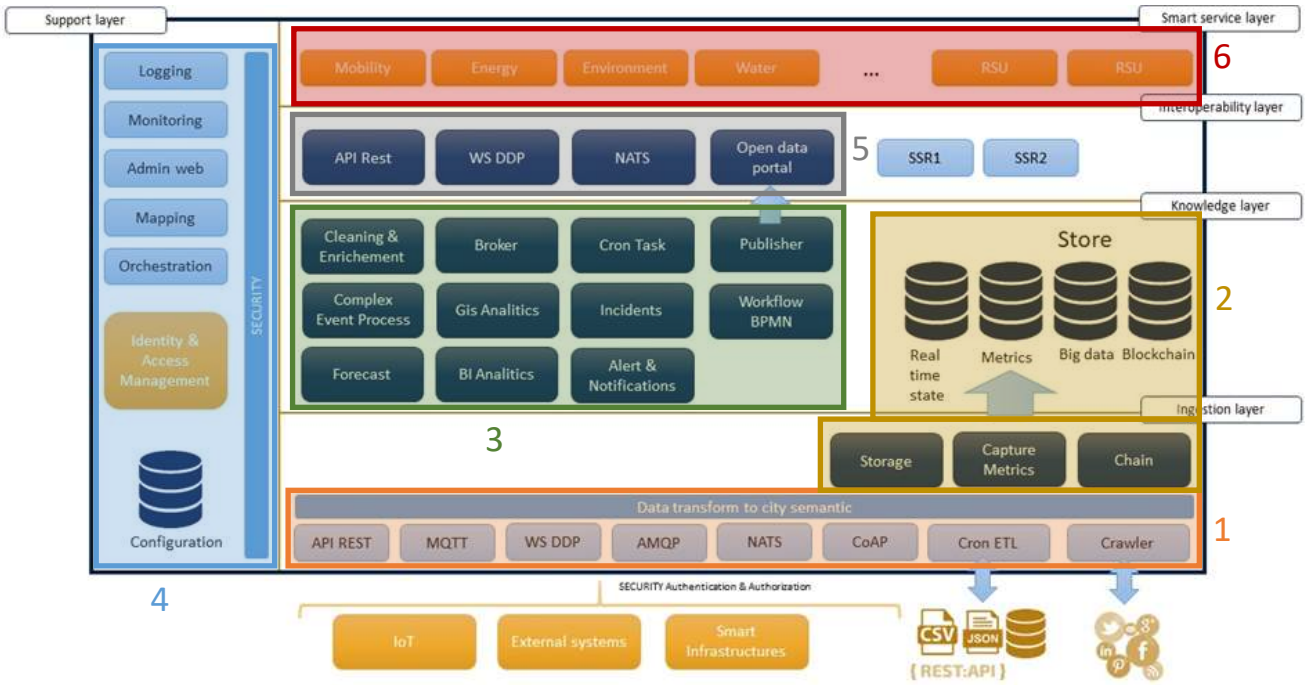

Figure 3 – CITRIC architecture (with highlighted modules)

1. **Data ingestion interfaces** [Orange]. Encompasses the different protocols and methods that the platform exposes to third parties (field assets, external systems, etc.) to send their data to the system. The specific protocols implemented by the X-FLEX Platform are detailed in Section 3.3. The data model transformation of the messages from external sources before their insertion into the platform is managed by a dedicated module, as described in Section 3.2.3.
2. **Data Management System** [Yellow]. Includes the different repositories of the X-FLEX Platform and the modules in charge of their management. Their details are reported in Section 3.1.2 of this document.
3. **Internal and horizontal modules** [Green]. Comprises a broad set of services for communication and analysis of data. The modules used in the X-FLEX platform as described in this deliverable are:
    - Message broker (Section 3.1.1)
    - Bad data detection and handling (Section 3.1.4)
    - Weather service provider (Section 3.2.1)
    - Temporal series forecasting (Section 3.2.2)
    - BPMN workflows (Section 3.2.4)
    - Virtual sensors (Section 3.2.5)
    - Complex Event Processor (Section 3.2.6)
4. **Support layer and user interface** [Blue]. Involves the Graphical User Interface of the X-FLEX Platform (Section 3.1.5) and all the mechanisms surrounding the configuration and management of the system (authentication, authorisation, configuration, orchestration, etc.).
5. **Interoperability** [Grey]. Covers the access of third-party applications to the data managed by the X-FLEX Platform. Includes both a subset of the interfaces described in Section 3.3 and the synchronization mechanisms to keep the data updated at any time (Section 3.1.3).
6. **Smart service layer/X-FLEX tools** [Red]. Is the layer of vertical applications or tools that use the platform, which in the case of X-FLEX will be formed by the three main tools: GRIDFLEX, SERVIFLEX, and MARKETFLEX. Though not strictly a part of the X-FLEX Platform, it involves some configuration (layers, collections, credentials, permissions, etc.) that is stored in and managed by the platform.

2.3 Requirements mapping

The requirements of the X-FLEX Platforms were defined in WP2 as part or T2.2 “Use cases and requirements definition” and reporting in the corresponding D2.2. The table below maps the 17 requirements associated to the Platform with a specific module or layer of its architecture.

Table 2 – X-FLEX Platform Requirements mapped to modules and layers

Requirement ID	Description	Type	Module/layer
PLF_001	A synchronization mechanism of all the tools is required	Functional and data requirements	Interoperability layer, WS DDP
PLF_003	X-FLEX platform must implement access control mechanisms over the information exchanged.	Functional and data requirements	Identity & Access Management
PLF_004	It collects and transfers heterogeneous and massive data stream coming from energy network.	Functional and data requirements	Ingestion layer
PLF_006	X-FLEX platform must interface with different energy distributed resources of the smart grid and energy market actors.	Functional and data requirements	Ingestion layer
PLF_007	X-FLEX platform is based on standards and interoperable technologies based on web services.	Performance requirements	All
PLF_008	X-FLEX platform must manage data according to privacy policy.	Functional and data requirements	Identity & Access Management
PLF_009	X-FLEX platform has to assure security.	Security requirements	Identity & Access Management
PLF_010	X-FLEX platform has to assure high performance and efficiency on data aggregation and filtering.	Performance requirements	Metrics, Big Data, CEP
PLF_011	Bad data (wrong metering data) coming for metering equipment should be detected as soon as possible and handled appropriately.	Functional and data requirements	Cleaning & enrichment
PLF_012	Communication structure/protocols used for data transfer between field assets and platform should be secure and in line with current standards.	Functional and data requirements	Ingestion layer, Interoperability layer
PLF_013	The X-FLEX Platform must support MQTT protocol	Functional and data requirements	MQTT, Broker
PLF_014	The X-FLEX Platform must support AMQP protocol	Functional and data requirements	AMQP, Broker
PLF_015	The X-FLEX platform must provide message persistence	Functional and data requirements	Knowledge layer, Store
PLF_016	The X-FLEX Platform must support REST/API services	Operational requirements	API REST (Ingestion layer &

			Interoperability layer)
PLF_017	The X-FLEX Platform should support integration with external sources (e.g., weather/weather forecasting services)	Functional and data requirements	External systems
PLF_018	The X-FLEX Platform should act as DER registry containing information about technical and operational limitations of the different assets	Functional and data requirements	Store, Real time state
PLF_019	The X-FLEX Platform should act as DER registry containing information about nominal parameters of the different assets	Functional and data requirements	Store, Real time state

3 X-FLEX PLATFORM IMPLEMENTATION

3.1 Internal modules

This section provides the technical documentation and details of the core internal modules of the X-FLEX Platform, which enable the operation of the platform itself.

3.1.1 Message broker

Two main systems manage the bulk of communications in the X-FLEX Platform: A RabbitMQ broker and a NATS broker.

The X-FLEX Platform relies on an instance of the RabbitMQ open source message broker for its ESB functionality for MQTT and AMQP protocols. It is mainly used for external communication, both for external assets that send data to the platform and third parties that consume this data using the aforementioned protocols. The RabbitMQ software is deployed with the following set of plugins:

- **rabbitmq_management:** user interface for RabbitMQ management.
- **rabbitmq_mqtt:** plugin implementing a MQTT broker integrated within RabbitMQ.
- **rabbitmq_shovel:** plugin that allows the replication of data managed by RabbitMQ in a managed manner. It enables the possibility to build complex ESB structures with different ESB instances replicating data among each other.
- **rabbitmq_shovel_management:** plugin that allows management of the *rabbitmq_shovel* plugin from the user interface.

A NATS broker is used mainly for internal communication among internal modules of the platform, although third-parties are also welcomed to integrate with NATS to receive updates from specific services of the tool. The NATS server is deployed with the following subject hierarchy [31]:

<domain>.<service>.<action>.<additional_details>

Where:

- **domain:** High-level label that identifies the instance of the platform. It is used so several instances of the platform can coexist using the same NATS broker.
- **service:** Name of the service managing the message. The service publishing the information must use its own identifier in the subject. Services subscribed to messages must use the identifier of the service they expect the information from.
- **action:** Identifier of the action performed by the service, e.g., get, getAll, create, update, delete
- **additional_details:** Custom information depending on the specific service and actions. Can involve more than one token.

3.1.2 Data Management System

The X-FLEX platform offers features that facilitate data ingestion to third party applications, also offering the possibility to outsource the storage of the real-time state and historical data within the X-FLEX Platform itself. The configuration with regards to data management is flexible in order to allow multitenancy, and appropriate authentication and authorization mechanisms are included to assure the data access is restricted only to the relevant parties.

3.1.2.1 Layers

The *_layers* collection defines the different systems that are handled by the X-FLEX Platform. A system may represent a particular third-party application (e.g. GRIDFLEX), or a data domain (e.g. distribution grid, AMI, etc.) depending on the approach taken.

```
<layer>: {
  _id: "name",
  label: "label",
  description: "description",
  owner: "owner",
  collections:[ ({<collection>}), ],
  (config:{})
}
```

Each *layer* defines the data *collections* to be handled.

```
{
      "_id" : "gridflex",
      "id" : "gridflex",
      "label" : "GRIDFLEX",
      "description" : "GRIDFLEX assets",
      "category" : "energy",
      "collections" : [
            {
                  "name" : "substations",
                  "geo" : true
            },
            {
                  "name" : "buses",
                  "metrics" : [
                        "gridflex.buses",
                        "gridflex.buses.forecast"
                  ]
            },
            {
                  "name" : "lines",
                  "geo" : true,
                  "metrics" : [
                        "gridflex.lines",
                        "gridflex.lines.forecast"
                  ]
            },
            {
                  "name" : "weather",
                  "metrics" : [
                        "gridflex.weather",
                        "gridflex.weather.forecast"
                  ]
            },
            ...
```

```
      ],
      ...
}
```

Each layer is configured with the following set of minimum attributes:

Table 3 – Set of minimum attributes

Attribute	Description
_id	Layer identifier
label	Layer label
description	Textual description of the layer
collections	Collections managed within the layer

X-FLEX Platform is agnostic with regards to the data model of the documents managed by the collections. Only the attribute *_id* is mandatory and interpreted as the identifier of the asset the document's information refers to.

3.1.2.2 Collections

Collections store all the information related to an asset, including static and real-time data. All data ingested by the X-FLEX Platform is mapped to a particular collection. Collections are the basic element that is referred by any query or subscription to data hold by the X-FLEX Platform.

```
// collection: gridflex_generators
{
    "_id": "TSO",
    "_expired": false,
    "_updated": ISODate("2021-07-28T07:48:17.111+02:00"),
    "substation": "CR_I8",
    "voltageLevel": "CR_I8_1",
    "measurements": {
        "activePower": {
            "quality": 1,
            "value": 3660.165534732381,
            "timestamp": ISODate("2021-07-28T07:48:17.103+02:00")
        },
        "reactivePower": {
            "quality": 1,
            "value": 366.0165534732381,
            "timestamp": ISODate("2021-07-28T07:48:17.103+02:00")
        }
    },
    "measurementsTimestamp": ISODate("2021-07-28T07:48:17.103+02:00")
}
```

3.1.3 Synchronization mechanism

All core services of the X-FLEX Platform have been built with 2 principles into account:

1. Functionality provided by each one of the services must be minimal, consistent and well-defined

2. All micro-services must implement a NATS interface, and are therefore reachable by each other by making the appropriate RPC calls to the NATS broker

By using these principles, no centralized orchestration mechanisms are required. Instead, each micro-service is in charge of triggering required actions from other services as needed, and to react to external actions triggered to itself. This way, orchestration happens in a distributed manner, with no central orchestrator in place.

3.1.4 Bad data detection and handling

As a receptor of a considerable amount of data that is subsequently used for analysis, the X-FLEX Platform must ensure all stored and/or broadcasted information will be useful for this purpose. In order to ensure this feat, the platform must count with the proper mechanisms to detect and manage bad data received from external sources and apply them at the very first stage of ingestion, before the anomalous information propagates to other modules or systems.

The Bad data detection and handling will be part of the Complex Event Processor of the tool (Section 3.2.6), and will be based on a set of rules, associated to one or more data sources and/or types of message, that will be assessed every time a new message arrives. The rules will have associated actions (discard, replace, etc.) to be performed with the messages that are not compliant with the defined directives.

The Complex Event Processor and the Bad data detection and handling module will be integrated in the tool for its second prototype.

3.1.5 User Interface

The X-FLEX Platform will present a Graphical User Interface to provide the system manager with the ability to create, update, and delete the different structural and security elements managed internally by the tool, including:

- Layers and collections,
- Users, profiles and permissions.

Moreover, the graphical interface of the tool will allow to check the general status of the system and the data stored in the platform. This includes:

- Last information updated in each layer and collection,
- Status of geo-located assets in a map visualization,
- Status of the different microservices that form the platform ecosystem, and
- Incidents generated by these services.

The core functionality of the GUI of the X-FLEX Platform will be developed in the second period of the implementation of the tool, and the related work and results will be reported in D6.4 Flexible and scalable integrated platform v2, due M36. Additionally, dedicated user interfaces from third-party solutions will be used to define the settings of specific services, e.g., RabbitMQ management user interface to define exchanges and queues.

3.2 Horizontal modules

This section provides the technical documentation and details of the horizontal modules of the X-FLEX Platform. Those services intend to provide features and services to be used by the applications integrating with the platform, or as generic out-of-the-box services that enrich the deployments.

3.2.1 Weather service provider

Microservice that provides current and forecasted weather information from **weatherbit.io**. The service is implemented as a Meteor application and uses the **weatherbit.io** REST API to retrieve the data. The specifications of the service are described below.

3.2.1.1 REST API

As a microservice, a RESTful API is exposed with the following endpoints:

- **/api/v1/weather**: Retrieve current weather information.
- **/api/v1/weatherforecast**: Retrieve forecast weather information.

Both endpoints can be queried using HTTP GET. Query parameters are:

- **cityName**: *(Mandatory)* Name of the place, used as a kind of ID, e.g. "Valencia". It will be included in the message.
- **lat**: *(Query option 1)* Latitude, e.g. "39.469917".
- **lon**: *(Query option 1)* Longitude, e.g. "-0.376300".
- **city**: *(Query option 2)* Name of the city, e.g. "Valencia", "Valencia,ES". Can be combined with the property "country".
- **postalCode**: *(Query option 3)* Postal code, e.g. "46014".
- **country**: *(Query options 2 and 3)* Country in ISO 3166-1 alpha-2 code format [32], e.g. "ES". It is mandatory for query option 3 and optional for query option 2.
- **hours**: *(Optional, only for forecast)* Number of hours in the future to retrieve. Default (and maximum) value: 48.

Examples:

- *HTTP GET /api/v1/weather?cityName=Valencia&lat=39.469917&lon=-0.376300*
- *HTTP GET /api/v1/weather?cityName=Valencia&city=Valencia,ES*
- *HTTP GET /api/v1/weather?cityName=Valencia&city=Valencia&country=ES*
- *HTTP GET /api/v1/weather?cityName=Valencia&postalCode=46014&country=ES*
- *HTTP GET /api/v1/weatherforecast?cityName=Valencia&lat=39.469917&lon=-0.376300*
- *HTTP GET /api/v1/weatherforecast?cityName=Valencia&city=Valencia,ES&hours=12*

Check the corresponding section below for descriptions of the data models of the responses.

3.2.1.2 AMQP integration

Additionally, the service can be configured to periodically request current and forecasted weather to **weatherbit.io** and send the results to a RabbitMQ exchange.

3.2.1.3 Properties file

The properties file contains the following fields:

- **auto**: If true, the server will periodically query the weather service.
- **refreshRateSecsCurrent**: Refresh rate in seconds to request new current weather data.
- **refreshRateSecsForecasted**: Refresh rate in seconds to request new forecasted weather data.
- **weatherServer**: Information related to external weather server.
- **dataUrl**: URL of the service for current weather.
- **forecastUrl**: URL of the service for forecasted weather.
- **APIkey**: API key to make requests.
- **hours**: Number of hours of prediction. By default, **weatherbit.io** send 48 hours of prediction, but it can be fewer. If it is set to 0, the request to forecasting won't be done. If it is greater than 48, the request to forecasting will be with 48 hours.
- **sites**: Array of sites we need weather information from. Each pilot site can be defined with different parameters, indicated below by different query options:
- **cityName**: *(Mandatory)* Name of the place, for instance "Valencia,ESP".
- **topic**: *(Optional)* Used to replace '#' in weatherTopic and forecastTopic (see option "amqp" below).
- **lat**: *(Query option 1)* Latitude, e.g. "39.469917".
- **lon**: *(Query option 1)* Longitude, e.g. "-0.376300".
- **city**: *(Query option 2)* Name of the city, e.g. "Valencia", "Valencia,ES". Can be combined with the property "country".
- **postalCode**: *(Query option 3)* Postal code, e.g. "46014".
- **country**: *(Query options 2 and 3)* Country in ISO 3166-1 alpha-2 code format [32], e.g. "ES". It is mandatory for query option 3 and optional for query option 2.
- **amqp**: Data related to connection to RabbitMQ.
    - **host**: Host.
    - **port**: Port.
    - **user**: Username.
    - **password**: Password.
    - **vhost**: Virtual host.
    - **exchange**: Exchange to send messages to.
    - **weatherTopic**: Topic used to send current weather messages.
    - **forecastTopic**: Topic used to send weather forecast messages.

3.2.1.4 Message data models

The following data models are used for current and forecasted weather messages.

3.2.1.4.1 Current weather

```
{
    "location": {
        "name": "Valencia,ESP",
        "latitude": 39.469917,
        "longitude": -0.3763
    },
    "timestamp": "2019-11-20T13:50:00Z",
    "atmosphericPressure": 1004.1, // Pressure (mbar)
    "seaLevelPressure": 1008.3, // Sea level pressure (mbar)
    "windSpeed": 3.13, // Wind speed (m/s)
```

```
    "windDirection": 222, // Wind direction (degrees)
    "windDirectionCardinal": "SW", // Abbreviated wind direction
    "temperature": 18.3, // Temperature (Celcius)
    "apparentTemperature": 18.4, // Apparent/"Feels Like" temperature
(Celcius)
    "relativeHumidity": 37, // Relative humidity (%)
    "dewPoint": 3.4, // Dew point (Celcius)
    "cloudCoverage": 33, // Cloud coverage (%)
    "visibility": 5, // Visibility (km)
    "precipitationAccumulation": 0, // Liquid equivalent precipitation rate
(mm/hr)
    "snowfallAccumulation": 0, // Snowfall (mm/hr)
    "ultravioletIndex": 4.34596, // UV Index (0-11+)
    "airQualityIndex": 0, // Air Quality Index [US - EPA standard 0 - +500]
    "diffuseHorizontalSolarIrradiance": 85.14, // Diffuse horizontal solar
irradiance (W/m^2) [Clear Sky]
    "directNormalSolarIrradiance": 729.53, // Direct normal solar
irradiance (W/m^2) [Clear Sky]
    "globalHorizontalSolarIrradiance": 363.91, // Global horizontal solar
irradiance (W/m^2) [Clear Sky]
    "solarRadiation": 357.2, // Estimated Solar Radiation (W/m^2)
    "elesolarElevationAnglevAngle": 23.05, // Solar elevation angle
(degrees)
    "solarHourAngle": 54 // Solar hour angle (degrees)
}
```

3.2.1.4.2 Forecasted weather

```
// Only metrics not present in the previous message are explained

[{
        "location": {
            "name": "Valencia,ESP",
            "latitude": 39.469917,
            "longitude": -0.3763
        },
        "timestamp": "2019-11-20T15:00:00Z",
        "atmosphericPressure": 1007.61,
        "seaLevelPressure": 1008.73,
        "windSpeed": 2.74225,
        "windDirection": 247,
        "windDirectionCardinal": "WSW",
        "windGustSpeed": 6.59272, // Wind gust speed (m/s)
        "temperature": 16.4,
        "apparentTemperature": 16.4,
        "relativeHumidity": 34,
        "dewPoint": 0.4,
        "cloudCoverage": 33,
```

```
        "cloudCoverageHighLevel": 0, // High-level (>5km AGL) cloud
coverage (%)
        "cloudCoverageMidLevel": 9, // Mid-level (~3-5km AGL) cloud
coverage (%)
        "cloudCoverageLowLevel": 25, // Low-level (~0-3km AGL) cloud
coverage (%)
        "visibility": 24.135,
        "precipitationAccumulation": 0,
        "precipitationProbability": 0, // Probability of Precipitation (%)
        "snowfallAccumulation": 0,
        "snowDepth": 0, // Snow Depth (mm)
        "ultravioletIndex": 1.79043,
        "diffuseHorizontalSolarIrradiance": 69.5,
        "directNormalSolarIrradiance": 616.06,
        "globalHorizontalSolarIrradiance": 226.87,
        "solarRadiation": 222.945,
        "ozone": 331.598 // Average Ozone (Dobson units)
    },
    ... // One message per hour
]
```

3.2.2 Temporal series forecasting

Microservice providing generalist temporal series forecasting. It is based on Prophet library [28], and provides REST and NATS interfaces. It is implemented in Python and uses the *prophet* package [33] from PIP.
The main characteristics of this forecasting module are:

- Configurable queries to retrieve train and regressor data
- Support for gaps in the source data
- Easy support for holidays/special dates in the model

3.2.2.1 Settings

Settings are provided as JSON in the SETTINGS environment variable

```
{
  "rest" : { -- OPTIONAL
    "port" : REST_port -- OPTIONAL
  },
  "nats" : { --OPTIONAL
    "connection" : {NATS_connection_object},
    "domain" : "NATS_domain"
  }
}
```

- **rest:** is an optional object containing REST interface details. By default, REST interface is disabled.
- **port:** port where the REST server will listen (default port is 5000)
- **nats:** is an optional object containing NATS interface details. By default, NATS interface is disabled.
    - **NATS_connection_object** is an object containing any required NATS connection parameters as documented here. Most usual parameters are:

```
{
```

```
  "servers": ["nats://127.0.0.1:4222"],
  "token": "auth_token"
}
```

- **NATS_domain** is the domain to be used in the NATS subscriptions (see below)

3.2.2.2 REST interface

An up-to-date swagger definition of the REST service is always reachable at the root URL of the REST server.

URL	METHOD	QUERY PARAMS	BODY PARAMS	RESPONSE	DESCRIPTION
/forecast/model	POST		MODEL_PARAMS	MODEL_DESCRIPTION	Creates a new (untrained) model
/forecast/model/{id}	GET	id: id of the model		MODEL_DESCRIPTION	Retrieves current model details
/forecast/model/{id}	DELETE	id: id of the model			Deletes model
/forecast/model/{id}/train	POST	id: id of the model	TRAIN_PARAMS(optional)		Triggers asynchronous training of model
/forecast/model/{id}/forecast	POST	id: id of the model	FORECAST_PARAMS	FORECAST_RESULT	Triggers model forecast

3.2.2.3 NATS interface

NATS interface receives a domain as parameter, which is used to setup the topics where the module will provide its service.

TOPIC	INPUT	RESPONSE	DESCRIPTION
{domain}.forecast.model.create	MODEL_PARAMS	MODEL_DESCRIPTION	Creates a new (untrained) model
{domain}.forecast.model.{id}.get		MODEL_DESCRIPTION	Retrieves current model details
{domain}.forecast.model.{id}.delete		MODEL_CONFIRMATION	Deletes model
{domain}.forecast.model.{id}.train	TRAIN_PARAMS	MODEL_CONFIRMATION	Triggers asynchronous training of model. Upon completion of training, MODEL_DESCRIPTION is published on {domain}.forecast.model.{id}.trainingcompleted
{domain}.forecast.model.{id}.forecast	FORECAST_PARAMS	MODEL_CONFIRMATION	Triggers asynchronous model forecast. Upon completion, FORECAST_RESULT is published on {domain}.forecast.model.{id}.forecastingcompleted

3.2.2.4 JSON objects description (common to both interfaces)

3.2.2.4.1 MODEL_PARAMS

```
{
    "name": "MODEL_NAME",
    "input": [
            INPUT_STAGE_1,
            INPUT_STAGE_2,
            ...
    ]
}
```

- **name:** name of the model (free text)
- **input:** array of INPUT_STAGE (optional)

3.2.2.4.2 INPUT_STAGE

The module can receive different input stages in the form of database queries, whose results will be used to train the model. All input stages are executed, and results are combined with the results of the first stage using INNER JOIN mechanism. In general, the module expects that, overall, the results of the queries provide the following columns:

- **ds:** timestamp of the value
- **y:** target variable to be forecasted
- Any other column existing in the results will be used as regressor in the model

It is recommended that the first stage is in charge of return values for the target variable *y*

3.2.2.4.2.1 MongoDB input stage

These input stages are identified by the existence of the property named mongo. They provide the results by executing an aggregate pipeline in the specified MongoDB collection.

```
{
    "mongo" : {

"uri":"mongodb://user:password@172.23.10.253:27017/?authSource=admin",
      "database":"rb_bigdata ",
      "collection":"gridflex_values_15m",

"aggregate":"[{\"$match\":{\"id\":\"TOTAL\"}},{\"$group\":{\"_id\":{\"tim
estamp\":{\"year\":{\"$year\":\"$timestamp\"},\"month\":{\"$month\":\"$ti
mestamp\"},\"day\":{\"$dayOfMonth\":\"$timestamp\"},\"hour\":{\"$hour\":\
"$timestamp\"}}},\"production\":{\"$avg\":\"$measures.Instantaneous Active
Power                                                    -
\"}}},{\"$project\":{\"_id\":0,\"ds\":{\"$dateFromParts\":{\"year\":\"$_i
d.timestamp.year\",\"month\":\"$_id.timestamp.month\",\"day\":\"$_id.time
stamp.day\",\"hour\":\"$_id.timestamp.hour\"}},\"y\":\"$production\"}}]"
    }
  }
```

- **uri:** URI of the mongo database, including credentials
- **database:** name of database

- **collection:** name of collection
- **aggregate:** string with the aggregate pipeline to be executed

3.2.2.4.2.2 HTTP input stage

These input stages are identified by the existence of the property named http. They provide the results by executing a query to the specified URI. A JSON response is expected.

```
{
    "http" : {
        "request": {
            "method": "GET",

"url":"http://192.168.1.1:8666/STH/v1/contextEntities/type/IoTGateway/id/
GATEWAY-01",
            "params" {
                "lastN": 1000
            },
            "headers": {
                "Accept" : "application/json",
                "Fiware-Service" : "FWSVC",
                "Fiware-ServicePath" : "/pilot1"
            },
        },
        "mapping": "S('contextResponses',0, 'contextElement','attributes',
0,  'values')  >>  ForallBend({'ds':  S('timestamp')  >>  F(lambda  t  :
datetime.strptime(t.lstrip(), '%Y-%m-%d %H:%M:%S')), 'y': S('temperature')
>> F(float)})",
    }
}
```

- **request:** parameters for the requests library request method
- **mapping:** data mapping configuration, as documented by JSONBender library [34]

3.2.2.4.2.2.1 X-FLEX Platform via HTTP API

```
{
    "name": "name",
    "input": [{
            "http": {
                "request": {
                    "method": "POST",
                    "url":
"http://xflexplatform:3333/api/collection/query/pnh_forecast_regs",
                    "headers": {
                        "Accept": "application/json",
                        "X-Auth-Token":
"fOlf28dD3ZhKshhcQnsyZXlcCL6ZFCvMmbUaf0fq-Gs",
                        "X-User-Id": "WrFWu5cncDj8Wb4sf",
                        "Content-Type": "application/json"
                    },
```

```
                    "json": {
                        "selector": {
                            "gateway": "XANTHI-GATEWAY"
                        }
                    }
                },
                "mapping":       "S('data')       >>       ForallBend({'y':
S('light_consumption')  >>  F(float),'ds':  S('date')  >>  F(lambda  t  :
dateutil.parser.isoparse(t).replace(tzinfo=None)),'clouds'  :  S('clouds')
>> F(float),'visibility' : S('visibility') >> F(float)})"
            }
        }
    ]
}
```

3.2.2.4.2.2.2 InfluxDB via HTTP API

```
{
    "name": "name",
    "input": [
        {
            "http" : {
                "request": {
                    "method": "GET",
                    "url": "http://linuxdemo.lab.id:8086/query",
                    "headers": {
                        "Accept" : "application/json"
                    },
                    "files": {
                        "q": "SELECT mean(\"req\") AS \"mean_req\" FROM
\"_internal\".\"monitor\".\"write\" GROUP BY time(1h) FILL(none)"
                    }
                },
                "mapping": "S('results', 0, 'series', 0, 'values') >>
ForallBend({'ds':        S(0)        >>        F(lambda        t        :
dateutil.parser.isoparse(t).replace(tzinfo=None)), 'y': S(1)})"
            }
        }
    ]
}
```

3.2.2.4.2.2.3 Elasticsearch via HTTP API

```
{
    "name": "name",
    "input": [
        {
            "http" : {
                "request": {
                    "method": "GET",
                    "url":
"http://linuxdemometrics.lab.id:9200/xflex.gridflex.energy/_search",
```

```
                    "params": {
                        "q": "evse:CR1_CENTER_VALENCIA-1"
                    }
                },
                "mapping":   "S('hits',   'hits')   >>   ForallBend({'ds':
S('_source',        'timestamp')        >>        F(lambda        t        :
dateutil.parser.isoparse(t).replace(tzinfo=None)),     'y':     S('source',
'consumptionWh')})"
            }
        }
    ]
}
```

3.2.2.4.2.2.4 *Some notes on jsonbender*

- ISO 8601 datetime parsing

```
S(0) >> F(lambda t : dateutil.parser.isoparse(t).replace(tzinfo=None))
```

- Custom datetime format

```
S('timestamp') >> F(lambda t : datetime.strptime(t.lstrip(), '%Y-%m-%d
%H:%M:%S'))
```

- String to float

```
S('temperature') >> F(float)
```

3.2.2.4.3 *MODEL_DESCRIPTION*

Provides a description and status of the model

```
{
  "id": "f9859fd2ba9d474fa186e819560a7ab6",
  "config": MODEL_PARAMS,
  "regressors": ["temp","clouds","dhi"],
  "status": "trained"
}
```

- **id:** id of the model, to be used in further calls to the API
- **config:** follows the MODEL_PARAMS model described above
- **regressors:** list of regressors used in the model
- **status:** current status of the model
  - *untrained:* the model has never been trained yet
  - *training:* the model is currently being trained. Forecast and train methods cannot be triggered in this state
  - *trained:* the model is ready to provide forecasts
  - *error:* the model couldn't be trained. Forecast method cannot be triggered in this state
- **lastDataTimestamp:** timestamp of last data present in the training set (only for trained models)

3.2.2.4.4 *TRAIN_PARAMS*

This parameter of the train method is optional. It can be used to overwrite default INPUT_STAGE provided in the model creation. It is mandatory if no INPUT_STAGE was provided in the model creation (since at least one is required to train the model)

```
{
```

```
    "input": [ -- OPTIONAL
        INPUT_STAGE_1,
        INPUT_STAGE_2,
        ...
    ]
}
```

3.2.2.4.5 FORECAST_PARAMS

Defines the requirements of the forecast. If the model uses regressors, it needs to populate forecasted values for all regressor variables in order to calculate the forecast for the target variable. Access to this information is provided in the form of INPUT_STAGE parameters to be used to get access to all required regressor information

```
{
    "periods": 24,
    "freq": "H",
    "input": [ -- REQUIRED ONLY IF WORKING WITH REGRESSORS
        INPUT_STAGE_1,
        INPUT_STAGE_2,
        ...
    ]
}
```

- **periods:** number of timesteps to forecast, starting at lastDataTimestamp value of the model
- **freq:** size of timestep. Any valid frequency for pd.date_range, such as 'D' or 'M'.

3.2.2.4.6 FORECAST_RESULT

Results of the forecast

```
[
    {
        "ds": "2020-05-22T11:00:00Z",
        "yhat": 3443.446599616558,
        "yhat_lower": 2714.4500966042624,
        "yhat_upper": 4105.982381406659
    },
    {
        "ds": "2020-05-22T12:00:00Z",
        "yhat": 3531.32472370326,
        "yhat_lower": 2864.308088210606,
        "yhat_upper": 4222.155932536654
    },
    ...
]
```

- **ds:** timestamp
- **yhat:** forecasted target value
- **yhat_lower:** lower value for 80% confidence interval
- **yhat_upper:** upper value for 80% confidence interval

3.2.3 Data mapping/data model converters

Several services in the X-FLEX Platform use a JSON data model transformation library to easily configure changes in the structure of the data before being stored in the database. Transformations can be defined in a MongoDB collection and allow two types of transformation: by template or by function. The data model to define transformations is defined as follows:

Transformation by template

```
{
    "_id" : "my-transformation",
    "template" : {
        "id" : "{{id}}",
        "date" : "=> toISO(timestamp)",
        "measurements" : {
            "activeEnergyExported" : {
                "timestamp" : "{{timestamp}}",
                "value" : "{{AEexp}}"
            }
        }
    },
    "functions" : {
        "toISO" : "(ts) => { return new Date(ts).toISOString(); }"
    }
}
```

- **_id:** ID of the transformation, to be referenced in other services
- **template:** JSON template to be used to format the result of the transformation. The fields of the original object are referenced with their name between double keys (“{{}}”). Additional JavaScript functions to perform more complex transformations can be referenced with the syntax “=> function(params)”.
- **functions** (optional): Definition of functions to be used in the template. A dictionary with the name of the function as key and the definition of the function (parameters and body) as a string value.

Transformation by function

```
{
    "_id" : "my-transformation",
    "code" : "(d) => { d.newField = 'myData'; return d; }"
}
```

- **_id:** ID of the transformation, to be referenced in other services
- **code:** String containing a JavaScript function that returns the object result of the transformation. The original object is available as the first parameter of the function.

3.2.4 BPMN workflows

BPMN workflows are included in the X-FLEX Platform as a possibility to configure specific workflows and business logic, relevant to specific pilot sites, without the need to address those needs by developing specific

software modules. This feature is supported by the inclusion as part of the X-FLEX Platform of the Flowable open-source solution.

As part of the commissioning of the system for a particular site, the required specific workflows can be defined using the Flowable Modeler tool, a BPMN WYSIWYG editor.

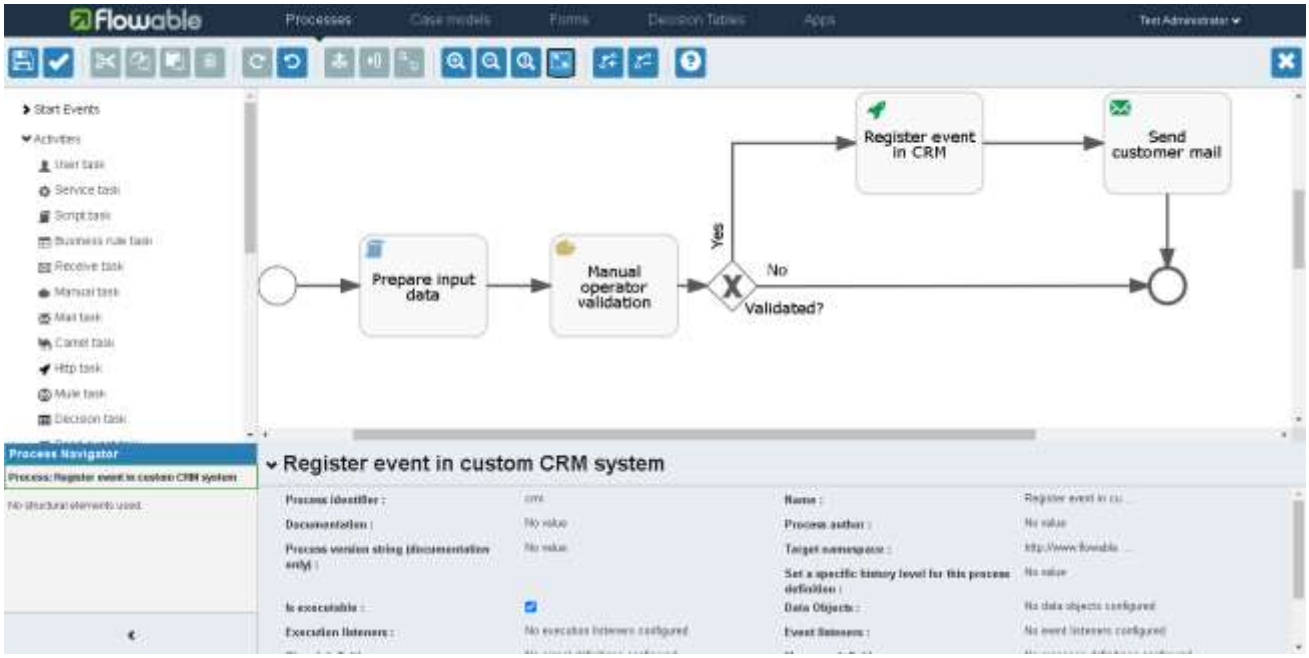

Figure 4: Flowable BPMN modeler

The X-FLEX platform can be configured to automatically start a new instance of any workflow upon the creation of a new document with certain characteristics. This configuration is performed by including the required settings in the *incidents* layer, which has an items collection whose documents are observed and evaluated to trigger the workflows as configured. Please note that the name of the *incidents* layer is that for historical reasons, but the functionality can be applied to any event of interest.

```
{
      "_id" : "incidents",
      "label" : "INCIDENTS",
      "config" : {
            "notify" : {
                  "OVERVOLTAGE" : {
                        "url"    :    "http://linuxtest1.lab.id:9081/flowable-
rest/service/runtime/process-instances",
                        "auth" : "user:pass",
                        "body" : {
                              "processDefinitionKey" : "overvoltage",
                              "businessKey" : "meterevents",
                              "returnVariables" : false,
                              "variables" : [
                                    {
                                          "name" : "data",
                                          "value" : "VALUE_INCIDENT"
                                    }
```

```
                              ]
                        },
                        "method" : "POST"
                  }
            }
      },
      "collections" : [
            {
                  "name" : "items",
                  "geo" : true,
                  "label" : "Incidents"
            },
            {
                  "name" : "protocols"
            }
      ]
}
```

This layer defines the *categories* of the events, and their *types*.
The *events* documents in the collection *incidents_items* must contain the *type* attribute in order to be automatically managed by the platform.

```
{
      "_id" : "z5uikc3dbfNvFssjs",
      "type" : "OVERVOLTAGE",
      "details" : {
            "meterId" : "SM001",
            "timestamp" : ISODate("2020-11-22T11:30:00.000+01:00"),
            "voltage" : 287
      },
      "description" : "An overvoltage was detected by SM001 (287V) at 12:30
22/11/2020"
}
```

The type attribute is used to crosscheck and retrieve the corresponding details about the HTTP endpoint to be triggered. By configuring the appropriate Flowable endpoint, the corresponding workflow gets triggered (even though this mechanism is flexible enough to allow triggering any other service offering REST API). Attribute with value *VALUE_INCIDENT* is replaced automatically by the complete content of the document that triggered the notification.

3.2.5 Virtual sensors

Allows definition of virtual sensors, whose data depends on other data available in X-FLEX Platform.

3.2.5.1 Settings

```
{
  "xflexplatform": {
```

```
    "url": "http://linuxtest1.lab.id:5553",
    "appId": "......",
    "keyId": "......"
  },
  "sensors":[
    {
      "collection": "gridflex_loads",
      "virtualsensors":[
        {
          "id":"LOAD01",
          "vars": {
            "Q":
"{gridflex_generators.TSO>measurements.reactivePower.value}                +
{gridflex_generators.PV1>measurements.reactivePower.value}",
            "CONST": 100
          },
          "template": {
            "measurements":{
              "activePower":{
                "value":
"({gridflex_generators.TSO>measurements.activePower.value}                 +
{gridflex_generators.PV1>measurements.activePower.value}) / 100",
                "quality": 1,
                "virtual": true,
                "timestamp": "{timestamp}"
              },
              "reactivePower":{
                "value": "{Q} / {CONST}",
                "quality": 1,
                "virtual": true,
                "timestamp": "{timestamp}"
              }
            },
            "measurementsTimestamp": "{timestamp}"
          },
          "window": 10
        },
        {
          "id":"LOAD02",
          "template": {
            "measurements":{
              "activePower":{
                "value":
"({gridflex_generators.TSO>measurements.activePower.value|0.0}             +
{gridflex_generators.PV1>measurements.activePower.value|0.0}) / 100",
                "quality": 1,
                "virtual": true,
                "timestamp": "{timestamp}"
              }
```

```
            },
            "measurementsTimestamp":
"toISOString(\"{gridflex_generators.TSO>measurementsTimestamp}\")"
          },
          "poll": true,
          "window": 60
        }
      ]
    }
  ]
}
```

- **xflexplatform:** contains all details required to connect to X-FLEX Platform using application login.
- **sensors:** array of definition of virtual sensors. Each element of this array contains the following properties:
  - **collection:** name of the collection where the virtual sensors will be published
  - **virtualsensors:** array of virtual sensors to be created, each of whom containing
    - **id:** id of the virtual sensor
    - **vars** (optional): calculated variables to be used in the template. May contain placeholders that will be updated with actual data, as explained below. Evaluated in order, a variable can reference a previously defined variable.
    - **template:** template of the message that will be used to update virtual sensor data in the X-FLEX Platform. May contain placeholders that will be updated with actual data, as explained below
    - **poll** (optional, default: false): by default, virtual sensors are calculated by observing updates from X-FLEX Platform. If poll is set to true, values are periodically polled from X-FLEX Platform instead -- see window and defaultValue in templates.
    - **window** (optional, seconds, default: 10s): updates on the virtual sensor will be produced only if all required data has been updated within the duration of the window. If poll is set to true, the window value refers to the period to generate new values for the virtual sensor, regardless of the last updated value of the values in X-FLEX Platform.

3.2.5.2 Template placeholders

The template property of the virtual sensors admits the following placeholders for actual data:

- {timestamp}: timestamp of the update, in ISO8601 format
- {var}: name of the calculated variable referenced in property vars
- Mathematical expressions using references to {collection.id>propertyRoute|defaultValue}, referencing to data of a particular element of X-FLEX Platform, identified by:
  - collection
  - id: id of the element
  - propertyRoute: route to a particular property of the element
  - defaultValue (optional): when poll is set to true, if the property is not available in X-FLEX Platform, this value will be used instead to complete the calculations
- Special methods:
  - toISOString: transforms a property of type Date into an equivalent ISO 8601 string. It must be defined as follows:

```
"toISOString(\"{collection.id>timestampProperty}\")"
```

3.2.6 Complex Event Processor

In order to provide high-performance analysis over big flows of streaming events, the X-FLEX Platform will integrate with Siddhi [27], a versatile, cloud-based Complex Event Processor with capabilities such as real-time analytics, data integration, notification management, and adaptive decision making. The processor provides its own SQL-like scripting language to integrate with external data sources, define pipelines and rules, and publish results to different data stores.
The integration with the CEP module is scheduled for the second prototype of the X-FLEX Platform, due to M36 of the project. The details of the integration and configuration of the service will be provided in D6.4 "Flexible and scalable integrated platform v2".

3.3 Interfaces and wrappers

This section provides the technical documentation of the different available interfaces that can be used by application developers or system integrators to integrate their developments with the X-FLEX Platform.

Interface	Description
REST API	HTTP(s)-based REST API to ingest and access data from the X-FLEX platform
DDP (websockets)	Websocket-based API to ingest and access data from the X-FLEX platform, enabling subscriptions to data updates
NATS	Message broker focusing on integration of RPC calls to micro-services
MQTT	Message broker focusing on data ingestion from IoT sensors
AMQP	Message broker focusing on data ingestion with data persistence and advanced message routing features
KAFKA	Message broker focusing on data ingestion with data persistence and high throughput
Modbus	Wrapper to integrate devices and systems exposing a Modbus interface

3.3.1 REST API

3.3.1.1 Summary

Path	Method	Description
/api/v2/health	GET	Path to check the service availability
/api/v2/login	POST	Authentication via user (ID, name or email) and password
/api/v2/:layer/:col	GET	Query documents from a collection
	POST	Insert one or multiple documents
	PATCH	Update one or multiple documents
	DELETE	Delete all the documents
/api/v2/:layer/:col/:id	GET	Get a single document
	PATCH	Update a document

	DELETE	Delete a document
/api/v2/config/:layer/:item	GET	Get a collection or serie configuration
	PUT	Update a collection or serie configuration
/api/v2/range/:layer/:col	GET	Query documents on a range of dates
/api/v2/near/:layer/:col	GET	Query documents near a geographic point
/api/v2/notify/twitter	POST	Post a tweet
/api/v2/notify/email	POST	Send an email
/api/v2/resources	GET	Get files info
	POST	Upload a new file
/api/v2/resources/:id	GET	Get a single file
	DELETE	Delete a single file
/api/v2/resources/:id/info	GET	Get a single file, including its metadata
/api/v2/resources/:id/metadata	PATCH	Modify the metadata of a file
/api/v2/gateway	POST	Provides access X-FLEX Platform microservices via NATS

This API is additionally provided as an openAPI specification under Annex 1. REST API openAPI specification.

3.3.1.2 Paths description

3.3.1.2.1 /api/v2/health [GET]

Path to check the service availability. Although it would usually be called with GET, it is possible to use any method.

Response:

```
{
  status: 'success',
  data: VERSION,
}
```

3.3.1.2.2 /api/v2/login [POST]

Authentication via user (ID, name or email) and password. Stores and returns the generated token.

Request body:

Field	Description
user	User ID, name or email
password	User password

Response:

```
{
  status: 'success',
  data: {
    authToken: TOKEN,
    userId: USER_ID,
```

```
  },
}
```

Authentication via login method is only required for regular users.

Application users must use the provided **userId** and **authToken**, there is no need to request those with a login call.

Check Section 3.3.1.3 for details on **userId** and **authToken** usage.

3.3.1.2.3 /api/v2/:layer/:col [GET]

Query documents from a collection.

Path params:

Param	Description
layer	Layer name
col	Collection name

Response:

```
{
  status: 'success',
  data: MATCHING_DOCUMENTS,
}
```

3.3.1.2.4 /api/v2/:layer/:col [POST]

Insert one or multiple documents. The request body must include the document or array of documents to be inserted.

Path params:

Param	Description
layer	Layer name
col	Collection name

Response:

```
{
  status: 'success',
  message: 'Item(s) inserted',
  data: {
    ok: INSERTED_DOCUMENTS_COUNT,
    insertedIds: INSERTED_IDS_ARRAY,
    err: FOUND_ERRORS_COUNT,
    errorIds: ERROR_IDS_ARRAY,
  },
}
```

When inserting multiple documents and some of them are successful, *status* will be *'success'* but *data.err* could be greater than 0; in this case, if the document that produced the error had an _id*, it will be inserted into* errorIds array.

3.3.1.2.5 /api/v2/:layer/:col [PATCH]

Update one or multiple documents. The request body must include an object or an array of objects with the _id and the fields to be updated of each document.

Path params:

Param	Description
layer	Layer name
col	Collection name

Response:

```
{
  status: 'success',
  message: 'Item(s) updated',
  data: {
    ok: UPDATED_DOCUMENTS_COUNT,
    updatedIds: UPDATED_IDS_ARRAY,
    err: FOUND_ERRORS_COUNT,
    errorIds: ERROR_IDS_ARRAY,
  },
}
```

3.3.1.2.6 /api/v2/:layer/:col [DELETE]

Delete all the documents.

Path params:

Param	Description
layer	Layer name
col	Collection name

Response:

```
{
  status: 'success',
  message: 'Item(s) deleted',
  data: {
    ok: DELETED_DOCUMENTS_COUNT,
    deletedIds: DELETED_IDS_ARRAY,
    err: FOUND_ERRORS_COUNT,
    errorIds: ERROR_IDS_ARRAY,
  },
}
```

3.3.1.2.7 /api/v2/:layer/:col/:id [GET]

Get a document.

Path params:

Param	Description
layer	Layer name
col	Collection name
id	Document ID

Response:

```
{
  status: 'success',
  data: REQUESTED_DOCUMENT,
}
```

3.3.1.2.8 /api/v2/:layer/:col/:id [PATCH]

Update a document. The request body must include the fields to be updated.

Path params:

Param	Description
layer	Layer name
col	Collection name
id	Document ID

Response:

```
{
  status: 'success',
  message: 'Item updated',
}
```

3.3.1.2.9 /api/v2/:layer/:col/:id [DELETE]

Delete a document.

Path params:

Param	Description
layer	Layer name
col	Collection name
id	Document ID

Response:

```
{
  status: 'success',
```

```
  message: 'Item deleted',
}
```

3.3.1.2.10 /api/v2/config/:layer/:item [GET]

Get a collection or serie configuration.

Path params:

Param	Description
layer	Layer name
item	Collection or serie name

Response:

```
{
  status: 'success',
  data: CONFIGURATION_OBJECT,
}
```

3.3.1.2.11 /api/v2/config/:layer/:item [PUT]

Update a collection or series configuration. The request body must contain the new configuration.

Path params:

Param	Description
layer	Layer name
item	Collection or serie name

Response:

```
{
  status: 'success',
}
```

3.3.1.2.12 /api/v2/range/:layer/:col [GET]

Query documents on a range of dates. Returns the documents where the corresponding field value is between the initial and end dates.

Path params:

Param	Description
layer	Layer name
col	Collection name

Query params:

Param	Description
from	Initial date *(Required)*
to	End date *(Required)*

field	Name of the field on which to compare the dates. Defaults to *'timestamp'*

Response:

```
{
  status: 'success',
  data: MATCHING_DOCUMENTS,
}
```

3.3.1.2.13 /api/v2/near/:layer/:col [GET]

Query documents near a geographic point. Returns the documents where *geometry* field is less than or equal to the provided distance.

Path params:

Param	Description
layer	Layer name
col	Collection name

Query params:

Param	Description
lat	Latitude *(Required)*
lon	Longitude *(Required)*
distance	Max. distance in meters *(Required)*

Response:

```
{
  status: 'success',
  data: MATCHING_DOCUMENTS,
}
```

3.3.1.2.14 /api/v2/notify/twitter [POST]

Post a tweet. Twitter developer credentials [35] needed.

Request body:

Field	Description
consumer_key	Twitter consumer key *(Required)*
consumer_secret	Twitter consumer secret *(Required)*
access_token_key	Twitter access token key *(Required)*
access_token_secret	Twitter access token secret *(Required)*
status	Text to be tweet *(Required)*

Response:

```
{
```

```
  status: 'success',
}
```

3.3.1.2.15 /api/v2/notify/email [POST]

Send an email. Request body must include valid Nodemailer message options [36].

Response:

```
{
  status: 'success',
}
```

3.3.1.2.16 /api/v2/resources [GET]

Get files info.

Query params:

Param	Description
fs	Bucket in which the files are stored. Initially it only exists one called *fs (Required)*
type	Media type of the files to be returned

Response is an array with the corresponding files.

3.3.1.2.17 /api/v2/resources [POST]

Upload a new file. The request body must contain the file to be inserted encoded as *multipart/form-data*.

Query params:

Param	Description
fs	Bucket in which the file will be stored. Initially it only exists one called *fs (Required)*
type	Media type of the file

Response:

```
{
  _id: '5df0a89f8a60e139366143e4',
  filename: 'file_1'
}
```

3.3.1.2.18 /api/v2/resources/:id [GET]

Get a single file.

Path params:

Param	Description
id	ID or filename of the requested file

Query params:

Param	Description
fs	Bucket in which the file is stored. Initially it only exists one called *fs (Required)*
key	Field in which to compare the given value of *id* path param. Possible values: *[id, filename] (Required)*

The content of the answer is the corresponding file that may be downloaded, shown in the browser, etc.

3.3.1.2.19 /api/v2/resources/:id [DELETE]

Delete a single file.

Path params:

Param	Description
id	ID or filename of the file to be deleted

Query params:

Param	Description
fs	Bucket in which the file is stored. Initially it only exists one called *fs (Required)*
key	Field in which to compare the given value of *id* path param. Possible values: *[id, filename] (Required)*

Response:

```
{
  _id: '5df0a89f8a60e139366143e4',
}
```

3.3.1.2.20 /api/v2/resources/:id/info [GET]

Get a single file, including its metadata.

Path params:

Param	Description
id	ID or filename of the requested file

Query params:

Param	Description
fs	Bucket in which the file is stored. Initially it only exists one called *fs (Required)*
key	Field in which to compare the given value of *id* path param. Possible values: *[id, filename] (Required)*

Response:

```
{
  _id: '5df0a89f8a60e139366143e4',
  length: 10711,
  chunkSize: 261120,
```

```
  uploadDate: '2019-11-10T07:00:00.000Z',
  md5: '8414f8a9ff2b74588c754b26c408dc81',
  filename: 'citric.png',
  metadata: {
    type: 'image/png',
  },
}
```

3.3.1.2.21 /api/v2/resources/:id/metadata [PATCH]

Modify the metadata of a file. The request body must contain the metadata fields to be updated.

Path params:

Param	Description
id	ID or filename of the requested file

Query params:

Param	Description
fs	Bucket in which the file is stored. Initially it only exists one called *fs (Required)*
key	Field in which to compare the given value of *id* path param. Possible values: *[id, filename] (Required)*

Response:

```
{
  key: {
    _id: '5df0a89f8a60e139366143e4',
  },
  update: DB_UPDATE_OPERATION_DATA,
}
```

3.3.1.2.22 /api/v2/gateway [POST]

Provides access X-FLEX Platform microservices via NATs.

Request body:

Param	Description
natsCredentials	Credentials to be used in the NATs connection to the microservice *(Required)*
subject	Subject of the targeted microservice. *Domain* must be omitted *(Required)*
method	NATs method used ("request" or "publish"). *(Optional, default: request)*
params	Parameters object that will be passed to the microservice *(Optional, default: {})*

Example:

```
{
    "natsCredentials":"xxxxxx",
    "subject":"metrics.get",
    "method":"request",
```

```
    "params":{
        "layer":"xflex",
        "collection":"xflex.gridflex.generators",
        "measures":["activePower"],
        "period":"1d",
        "filter":{"generator":["TSO"]}
    }
}
```

3.3.1.3 Authentication

All methods except login require credentials: - userId - authToken (response to login method call for regular users, or provided upon registration to application users)

Credentials must be included in every API call by using one of the following mechanisms: - As "appId" and "keyId" query parameter - As "X-User-ID" and "X-Auth-Token" headers

3.3.2 DDP (webSockets)

Module to connect to an X-FLEX Platform via DDP or HTTP:

- **IOP**. Extends SimpleDDP [37] and provides methods to interact with an X-FLEX Platform via DDP.
- **IOPHTTP**. Provides methods to interact with an X-FLEX Platform via HTTP.

3.3.2.1 Usage

Install the package running:

```
npm install iop-connector
```

3.3.2.2 DDP connector

Available methods:

Method	Parameters	Description
login	user, password	Authentication via user name and password. Returns an object with the fields *userId*, *token* and *tokenExpires*
loginApp	appId, keyId	Authentication via appId and keyId for application-type users
renewToken		Renew authentication token (only needed for regular users)
isLogged		Returns whether the user is logged in or not
findAll	collection, selector	Query documents from a collection
insert	collection, data	Insert one or multiple documents. *data* can be an array of documents
update	collection, data	Update one or multiple documents. *data* can be an array of documents
remove	collection, selector	Delete all the matching documents (Use carefully!)
findOne	collection, id	Get a single document

All methods accept a callback as a last parameter or return a Promise if not provided. Note that the parameter "collection" would be in the form of *layer_collection*.

3.3.2.2.1 DDP example

```
import { IOP } from 'iop-connector';
// or const { IOP } = require('iop-connector');

async function start() {
  // Initialize the connection
  const server = IOP('https://your.server.com');

  // Login as a regular user
  const userLogin = await server.login('<username>', '<password>');
  console.log(userLogin.tokenExpires); // "2020-07-28T11:49:06.455Z"

  // Or login as an application (in case you are provided with credentials
in the form of appId and keyId)
  await server.loginApp('<appId>', '<keyId>');

  // Subscribe to the desired data. Creating the corresponding subscriptions
is mandatory in order to obtain the data
  const subscription = server.subscribe('<layer>_<collection>.all');
  await subscription.ready();

  // Retrieve the desired data
  const allItems = await server.findAll('<layer>_<collection>');
  const    oneItem    =    await    server.findOne('<layer>_<collection>',
'<item_id>');

  // Remove one element
  await server.remove('<layer>_<collection>', { _id: '<item_id>' });

  // Manage the token renovation when needed
  await server.renewToken();
}

start();
```

3.3.2.3 HTTP connector

Available methods:

Method	Parameters	Description
health		Check the server availability
login	user, password	Authentication via user name and password
loginApp	appId, keyId	Authentication via appId and keyId for application-type users
renewToken		Renew authentication token (only needed for regular users)

me		Get logged user ID and profile
get	layer, collection, selector	Query documents from a collection
add	layer, collection, data	Insert one or multiple documents. *data* can be an array of documents
mod	layer, collection, data	Update one or multiple documents. *data* can be an array of documents
del	layer, collection, selector	Delete all the matching documents
getOne	layer, collection, id	Get a single document
modOne	layer, collection, id, data	Update a document. *data* is an object with the changes to be applied
delOne	layer, collection, id	Delete a document
range	layer, collection, data	Query documents on a range of dates. *data* is an object with the fields *from*, *to* and *field* (if the field in which to compare the dates is different than *timestamp*)
near	layer, collection, data	Query documents near a geographic point. *data* is an object with the fields *lat*, *lon* and *distance*

All methods accept a callback as a last parameter or return a Promise if not provided. The response is an object with the fields *status* (that should be *"success"*) and *data*.

3.3.2.3.1 HTTP example

```
import { IOPHTTP } from 'iop-connector';
// or const { IOPHTTP } = require('iop-connector');

async function start() {
  // Initialize the connection
  const server = IOPHTTP('https://iop.server.com');

  // Login
  await server.login('<username>', '<password>');

  // Retrieve the desired data
  const allItems = await server.get('<layer>', '<collection>');
  const oneItem = await server.getOne('<layer>', '<collection>',
'<item_id>');

  // Manage the token renovation if needed
  await server.renewToken();
}

start();
```

If you are provided with application credentials in the form of *appId* and *keyId*, there are two options to manage the authentication:

1. Pass your credentials when initializing the connector:

```
const server = IOPHTTP('https://your.server.com/api/v2', '<appId>', '<keyId>');
```

2. Use the *loginApp* method:

```
server.loginApp('<appId>', '<keyId>')
```

Note that it is a synchronous method so doesn't accept a callback nor return a Promise.

3.3.3 NATS

Within the X-FLEX Platform, a NATS broker is included to simplify the configuration of the communication channels among horizontal modules and applications or other modules of the platform, particularly in those communication flows based on RPC calls (i.e., one entity calling functions from other entities).
The NATS broker instance included in the X-FLEX Platform is configured as follows:

- Authentication is enabled: any module must be in possession of valid credentials in order to be able to connect to the NATS broker.
- Authorization mechanisms are enabled: each set of credentials (user, password) is configured with the appropriate set of permissions (publish, subscribe, allow_responses) on the appropriate set of topics
- Multitenancy: topics within the X-FLEX Platform must start with a first level defining the *domain*. This allows the coexistence of different domains within the X-FLEX Platform that operate independently from each other a logical perspective. Security configuration enables the effective compartmentalisation of these domains.

The X-FLEX Platform manager is responsible to appropriately define and maintain these security rules.

3.3.4 MQTT/AMQP

MQTT and AMQP protocols are supported by the Message broker of X-FLEX Platform, an instance of the RabbitMQ open source message broker.
Access to the ESB via AMQP protocol is effectively secured by the following mechanisms enabled by RabbitMQ, namely:

- Virtual hosts: within RabbitMQ, all elements can be organised within different virtual hosts, which enable the effective logical separation of the resources (e.g., exchanges, queues) managed by the ESB. This is especially useful when used in combination with authorization and authentication mechanisms (e.g. a particular user can be administrator of one virtual host, having no access to the other ones)
- Authentication: connections to the broker are secured by requiring valid credentials
- Authorization: each user is assigned with the appropriate set of permissions that provides access only to the required resources. Permissions can be given at virtual host level, and/or at exchange level.

Access to the ESB via MQTT protocol is also secured:

- Authentication: MQTT connections require of valid credentials, defined in the same manner as described above.
- Authorization: since the MQTT implementation provided by the RabbitMQ broker effectively maps the complete MQTT broker to a particular exchange (amq.topic), permissions under the MQTT protocol can be managed by using the RabbitMQ authorization configuration at the corresponding exchange.

The X-FLEX Platform manager is responsible to appropriately define and maintain these security rules.

3.3.5 Modbus

This plugin implements 2 main features:

1. Periodically polls MODBUS slaves and publishes status using MQTT
2. Listens to specific topics and writes MODBUS variables according to the content

3.3.5.1 Settings

All settings are provided as a JSON using the *SETTINGS* environment variable

```
{
    "modbusMaps": {
        "PANASONIC": [{
                    "name": "temperature",
                    "type": "holding",
                    "address": 1,
                    "datatype": "16uint",
                    "factor": 1
                }, {
                    "name": "luminance",
                    "type": "holding",
                    "address": 2,
                    "datatype": "16int",
                    "factor": 10
                }, {
                    "name": "mode",
                    "type": "holding",
                    "address": 3,
                    "datatype": "16int",
                    "translation": [[1, "COLD"], [2, "HEAT"], [3, "FAN"]]
                }
            ]
    },
    "mqttMessages": {
        "SENSOR": {
            "timestamp": "_DATE_",
            "temperature": "_temperature_",
            "luminance": "_luminance_",
            "mode": "_mode_"
        }
    },
    "modbusSlaves": [{
            "ip": "172.31.2.43",
            "slaveId": 1,
            "map": "PANASONIC",
            "mqttPublications": [{
                    "topic": "SENSOR_ABC/STATUS",
```

```
                    "messageBase": "SENSOR",
                    "messageExtras": {
                        "id": "SENSOR_ABC"
                    }
                }
            ],
            "mqttSubscriptions":                    ["SENSOR_ABC/DIMMING",
"SENSOR_ABC/THERMOSTAT"]
        }
    ],
    "mqtt": {
        "host": "192.168.168.59",
        "credentials": {
            "username": "etra",
            "password": "etra"
        }
    },
    "period": 10,
    "logLevel": "Info"
}
```


- **modbusMaps:** Object with the MODBUS map definition implemented by the MODBUS assets (see details below)
- **mqttMessages:** Templates for the messages that will be published using MQTT
- **modbusSlaves:** Array of modbus connection settings
    - **ip:** IP address of MODBUS slave (mandatory for MODBUS TCP connections)
    - **serialPort:** serial port to be used for MODBUS RTU connections (mandatory for serial connections)
    - **baudRate:** baud rate of serial port (mandatory for serial connections)
    - **byteSize:** byte size of serial port (mandatory for serial connections)
    - parity: parity of serial port (mandatory for serial connections)
    - **stopBits:** stop bits of serial port (mandatory for serial connections)
    - **slaveId:** MODBUS Id of the slave device
    - **map:** reference to the implemented MODBUS map
    - **mqttPublications:** array of status messages to be sent via MQTT (see below)
    - **mqttSubscriptions:** array of topics that will publish information affecting this MODBUS slave (see below)
- **mqtt:**
    - **host:** IP address of MQTT broker
    - **credentials:** credentials to connect to MQTT broker (optional)
- **period:** MODBUS poll period (in seconds). Status will be published via MQTT using the same period
- **logLevel:** Log level of standard output (optional, default value = Error). Possible values include:
    - **Info:** everything is logged
    - **Error:** only errors are logged

3.3.5.2 MODBUS map definition

Each variable to be read is defined as follows:

- **name:** name of the variable
- **type:** type of the variable:
  - holding
  - input
  - coil
- **address:** address of the variable (0-based index)
- **datatype:** data type of the variable. Possible values include:
  - string. Requires an additional property *datasize* indicating the length
  - bits
  - 8int
  - 8uint
  - 16int
  - 16uint
  - 32int
  - 32uint
  - 16float
  - 32float
  - 64int
  - 64uint
  - ignore. Requires an additional property *datasize* indicating the length
  - 64float
- **factor:** factor to be applied to the reading (applicable to numeric data types) (optional, default value = 1)
- **shift:** number of bits to be shifted - right shifting (optional, default value = 0)
- **mask:** mask to be applied to read value after shifting (optional, do not apply mask by default)
- **translation:** dictionary of translations to be applied to the value (after previous value transformations). Useful to homogenize states that are encoded in a different way by different vendors (e.g. HVAC operation mode - COLD, HEAT, FAN) (optional)

3.3.5.3 MQTT publications

Each message to be published via MQTT is defined as follows:

- **topic:** topic to be used in the MQTT protocol
- **messageBase:** reference to the MQTT template of the message to be published
- **messageExtra:** JSON structure of fields that will be appended to the message to be published

3.3.5.3.1 Placeholders

The messages can contain placeholders for values read from the MODBUS slave. This is indicated by referencing the name of the MODBUS variable between lowdashes (_*VARIABLE*_). See example above. _*DATE*_ is a special placeholder that will be substituted by the ISO8601 representation of current timestamp

3.3.5.4 Programming MODBUS slaves using MQTT

When *mqttSubscription* parameter is used, the module will subscribe to the given topics and bind the received messages to the MODBUS slave variables, performing the necessary writes on the device. *NOTE: currently, this feature only is supported for holding registers and variables of length 1* E.g. the following message will be translated in a write command to MODBUS slave with Id 1, on holding register on position 1 (temperature variable)

```
TOPIC: SENSOR/THERMOSTAT
PAYLOAD:
{
    "temperature": 21
}
```

If translations are provided for this variable, the translation must be used in the message. The module will handle the writing of the proper value to the MODBUS device.

```
TOPIC: HVAC/COMMAND
PAYLOAD:
{
    "mode": "FAN"
}
```

3.3.6 OPC-UA

Although initially considered for native inclusion in the X-FLEX Platform, all systems in the pilot sites that could potentially use OPC-UA for data extraction implement a wrapper to MQTT. As the Platform already integrates with the MQTT protocol, the implementation of OPC-UA has been deemed unnecessary and thus dismissed in the scope of X-FLEX.

3.3.7 KAFKA

The X-FLEX Platform includes a KAFKA cluster, to which any field asset wrapper or application can connect to produce and/or consume message from a certain topic.
Authorization mechanisms have been configured based on the SASL_PLAINTEXT (SCRAM-SHA-512) security protocol of KAFKA. In practical terms, this means that, in general, any external party willing to use KAFKA on the X-FLEX Platform will follow the following procedure in order to get connection:

1. Provide to X-FLEX Platform manager a list of topics and operations (read/write) required on it.
2. X-FLEX Platform manager will create in the KAFKA cluster the required new topics and users, assign the required permissions to those users, and communicate back the list of credentials to access the different created topics

```
Topic: asset1
Users: etra1:***** (PRODUCER), etra2: ***** (CONSUMER)
```

3. External party will be able to establish connection to the KAFKA cluster by appending the following configuration to the corresponding KAFKA client software/library (it should be adapted to match parameters of the specific client software/library)

```
security.protocol=SASL_PLAINTEXT
sasl.mechanism=SCRAM-SHA-512
sasl.jaas.config=org.apache.kafka.common.security.scram.ScramLoginModule
required username="etra1" password="****";
```

3.3.7.1 KAFKA, AMQP and MQTT interoperability

The X-FLEX Platform enables 3 main communication protocols for data ingestion, namely AMQP, MQTT and KAFKA. These 3 protocols are different in a way that makes each of them more suitable for scenarios with different requirements, but in general they serve the same purpose. As an added service, the X-FLEX Platform

implements interoperability among all three protocols, making it possible to a data producer and a data consumer to communicate with each other using any combination of the three.

- Interoperability among AMQP and MQTT is provided out of the box by *MQTT Plugin* of RabbitMQ, which implements a MQTT broker that maps to a topic exchange on the AMQP protocol, which allows bidirectional interoperability (i.e. for subscriptions and publications)
- Interoperability among KAFKA and AMQP is provided by the *kafka-rabbit-bridge* micro-service, which performs the following configurable mappings:
    - For data flows whose source is a KAFKA topic, the KAFKA topic (producer) y mapped to a topic exchange (consumer) on the AMQP protocol
    - For data flows whose source is an AMQP queue, the AMQP queue (producer) is directly mapped to a KAFKA topic (consumer)
    - Interoperability among KAFKA and MQTT can be achieved with the proper configuration of a combination of the previous two points

4 REQUIRED INFRASTRUCTURE

All modules composing the X-FLEX Platform are bundled in the form of docker images. This principle facilitates the deployment of the platform and enables several possible options with regards to the required infrastructure.

In the context of the X-FLEX project, 2 different deployment options are under consideration depending on each pilot site requirements mainly with regards to physical location of the data storage:

- On-premise deployment, for which the X-FLEX Platform is recommended to be deployed in a docker swarm cluster located at pilot site's data center
- On-cloud deployment, where a single instance of the X-FLEX Platform can be deployed to provide service to more than one pilot site, following a SaaS approach.For this case, and in order to be flexible and scalable, a Kubernetes cluster [7] has been selected, based on the Autopilot mode of operation of Google Kubernetes Engine [38].

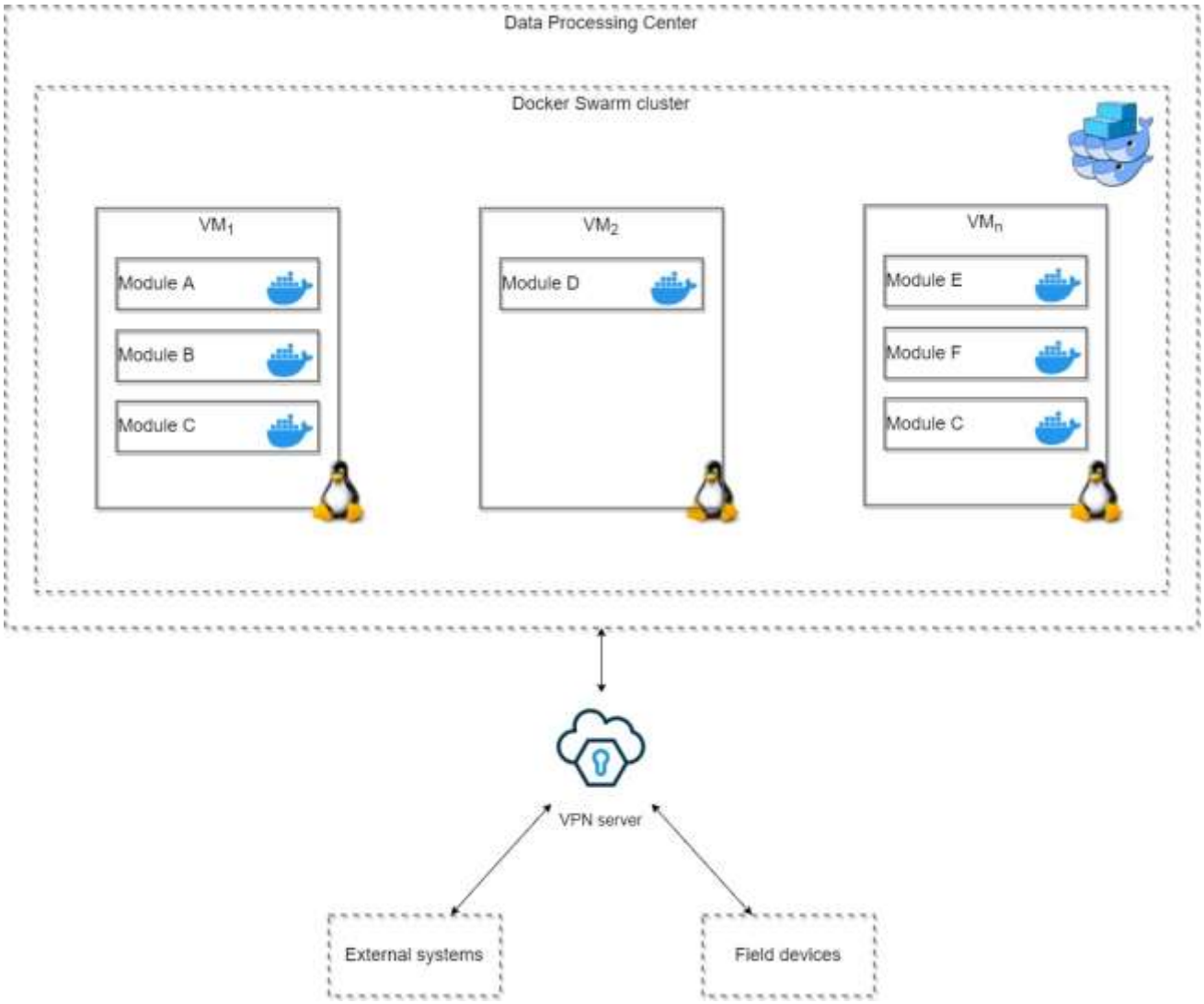

Figure 5 – Infrastructure set up for X-FLEX Platform deployment

In any of both approaches, the security of the cluster is ensured by a VPN server, which can only be accessed by those field devices and external systems that have been explicitly allowed to connect. Inside the cluster, a

Kubernetes cluster [6] or docker swarm cluster is set up in order to deploy the necessary services that form the tool.

In particular for the on-cloud foreseen deployment, Kubernetes allows to deploy as many instances of each service as necessary to support the load of the system in each moment. This process, which is automatically managed by Kubernetes, makes the X-FLEX Platform ecosystem scalable for future integration of further pilots and new field infrastructure.

5 CONCLUSIONS

The present document has provided the technical details surrounding the first prototype of X-FLEX's flexible and scalable integrated platform (i.e., X-FLEX Platform). The document builds upon the work performed in T6.2 "Design of the X-FLEX Flexible and scalable integrated platform", where the design of the architecture and the different modules that form the system were provided.

As explained in the document, the platform is not built from scratch. The use of a central ESB for communication and horizontal functionalities is frequent in H2020 ecosystems, especially in the field of Energy. Having several years of experience in this kind of projects, the decision of ETRA has been to create a tool for such purposes, whose implementation will be incremented and refined in each of these projects. The X-FLEX Platform is based on this tool, and work carried out within the X-FLEX project has the objective of implementing new features and advancing existing ones in order to make X-FLEX Platform cover all the requirements imposed by the project.

The implementation of the platform is modular, based on the microservice paradigm: a set of diverse, independent services focused on a single or a reduced set of functionalities that communicate among them to share data, events, and request. Contrary to the classical approach of a single, monolithic application that manages every aspect of the system, the microservice paradigm enhances the scalability and the resilience of the system.

The different modules of the platform have been detailed from a technical perspective. When available, documentation on how to use these modules has been provided, so the document is not purely theoretical but also serves to developers using the tool when integrating their services and systems with the X-FLEX Platform. The same approach will be used for D6.4 "Flexible and scalable integrated platform v2" for those modules whose implementation has not been covered in the present deliverable. Any possible modification, correction, or extensions to the modules in the first prototype that may emerge from the first round of tests will be included in v2 deliverable as well.

6 REFERENCES

[1] https://www.iso.org/standard/53302.html. [Online].

[2] https://www.en-standard.eu/une-178104-2017-sistemas-integrales-de-gestion-de-la-ciudad-inteligente-requisitos-de-interoperabilidad-para-una-plataforma-de-ciudad-inteligente/. [Online].

[3] "ASSISTANCE Project," [Online]. Available: https://assistance-project.eu/.

[4] "SAURON Project," [Online]. Available: https://www.sauronproject.eu/.

[5] "MATCHUP Project," [Online]. Available: https://www.matchup-project.eu/.

[6] "Docker Swarm mode overview," [Online]. Available: https://docs.docker.com/engine/swarm/.

[7] "Kubernetes," [Online]. Available: https://kubernetes.io/.

[8] "WebGL," [Online]. Available: https://get.webgl.org/.

[9] "OpenID - The Internet Identity Layer," [Online]. Available: https://openid.net/connect/.

[10] "Security Assertion Markup Language (SAML) V2.0 Technical Overview," [Online]. Available: http://docs.oasis-open.org/security/saml/Post2.0/sstc-saml-tech-overview-2.0.html.

[11] "Lightweight Directory Access Protocol (LDAP): The Protocol," [Online]. Available: https://datatracker.ietf.org/doc/html/rfc4511.

[12] "Keycloak," [Online]. Available: https://www.keycloak.org/.

[13] "MongoDB," [Online]. Available: https://www.mongodb.com/.

[14] "Express - Node.js web application framework," [Online]. Available: http://expressjs.com/.

[15] "MQTT - The Standard for IoT Messaging," [Online]. Available: https://mqtt.org/.

[16] "RabbitMQ -- Messaging that just works," [Online]. Available: https://www.rabbitmq.com/.

[17] "AMQP," [Online]. Available: https://www.amqp.org/.

[18] "Publications and Data Loading | Meteor Guide," [Online]. Available: https://guide.meteor.com/data-loading.html#publications-and-subscriptions.

[19] "NATS.io – Cloud Native, Open Source, High-performance Messaging," [Online]. Available: https://nats.io/.

[20] "The Constrained Application Protocol (CoAP)," [Online]. Available: https://datatracker.ietf.org/doc/html/rfc7252.

[21] "Node-RED," [Online]. Available: https://nodered.org/.

[22] "spaCy · Industrial-strength Natural Language Processing in Python," [Online]. Available: https://spacy.io/.

[23] "Elastic Stack: Elasticsearch, Kibana, Beats & Logstash | Elastic," [Online]. Available: https://www.elastic.co/elastic-stack/.

[24] "InfluxDB: Purpose-Built Open Source Time Series Database | InfluxData," [Online]. Available: https://www.influxdata.com/.

[25] "BigchainDB • • The blockchain database," [Online]. Available: https://www.bigchaindb.com/.

[26] "KNIME Analytics Platform | KNIME," [Online]. Available: https://www.knime.com/knime-analytics-platform.

[27] "Siddhi," [Online]. Available: https://siddhi.io/.

[28] “Prophet | Forecasting at scale.,” [Online]. Available: https://facebook.github.io/prophet/.

[29] “Flowable - Award-winning Intelligent Automation Platform,” [Online]. Available: https://flowable.com/.

[30] “CKAN - The open source data management system,” [Online]. Available: https://ckan.org/.

[31] “Subject-Based Messaging - NATS Docs,” [Online]. Available: https://docs.nats.io/nats-concepts/subjects.

[32] “ISO 3166-1 alpha-2 # Officially assigned code elements - Wikipedia,” [Online]. Available: https://en.wikipedia.org/wiki/ISO_3166-1_alpha-2#Officially_assigned_code_elements.

[33] “prophet · PyPI,” [Online]. Available: https://pypi.org/project/prophet/.

[34] “GitHub - Onyo/jsonbender: Library for transforming JSONs,” [Online]. Available: https://github.com/Onyo/jsonbender.

[35] “Developer documentation - Twitter,” [Online]. Available: https://apps.twitter.com/.

[36] “Message configuration :: Nodemailer,” [Online]. Available: https://nodemailer.com/message/.

[37] “SimpleDDP,” [Online]. Available: https://gregivy.github.io/simpleddp/.

[38] “Autopilot overview | Kubernetes Engine Documentation | Google Cloud,” [Online]. Available: https://cloud.google.com/kubernetes-engine/docs/concepts/autopilot-overview.

7 ACRONYMS

Acronym List	
AMI	Advanced Metering Infrastructure
AMQP	Advanced Message Queuing Protocol
API	Application Programming Interface
BPMN	Business Process Model and Notation
CEP	Complex Event Process
CoAP	Constrained Application Protocol
DDP	Distributed Data Protocol
ESB	Enterprise Service Bus
GIS	Geographic Information System
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
NATS	Neural Autonomic Transport System
REST	Representational state transfer
RPC	Remote Procedure Call
SaaS	Software as a Service
SSR	Server Side Rendering
URI	Uniform Resource Identifier
VPN	Virtual Private Network
WS DDP	Websockets Distributed Data Protocol
WYSIWYG	What You See Is What You Get

ANNEX 1. REST API OPENAPI SPECIFICATION


```
{
  "openapi": "3.0.1",
  "info": {
    "title": "X-FLEX Platform API",
    "description": " X-FLEX Platform API documentation.",
    "version": "v2.0"
  },
  "servers": [
    {
      "url": "{protocol}://citric-api.tec.etra-id.com/api/v2",
      "variables": {
        "protocol": {
          "enum": ["http", "https"],
          "default": "https"
        }
      }
    }
  ],
  "tags": [
    {
      "name": "general",
      "description": "General routes"
    },
    {
      "name": "collections",
      "description": "Routes related to Collections"
    },
    {
      "name": "resources",
      "description": "Routes related to Resources"
    },
    {
      "name": "config",
      "description": "Routes related to collections or series configuration"
    },
    {
      "name": "utils",
      "description": "Routes to execute utilities"
    }
  ],
  "paths": {
    "/health": {
      "get": {
        "tags": [
          "general"
        ],
        "summary": "Service availability",
        "description": "Path to check the service availability. Although it would
usually be called with GET, it is possible to use any method.",
        "operationId": "health",
        "responses": {
          "200": {
            "description": "Success",
            "content": {
              "application/json": {
                "schema": {
                  "properties": {
```

```
                    "status": {
                      "type": "string",
                      "example": "success"
                    },
                    "data": {
                      "type": "string",
                      "example": "v2.0"
                    }
                  }
                }
              }
            }
          }
        }
      },
      "/login": {
        "post": {
          "tags": [
            "general"
          ],
          "summary": "Authentication",
          "description": "Authentication via user (ID, name or email) and password.
Stores and returns the generated token.",
          "operationId": "login",
          "requestBody": {
            "content": {
              "application/json": {
                "schema": {
                  "properties": {
                    "user": {
                      "type": "string",
                      "description": "User ID, name or email",
                      "example": "my_user_name"
                    },
                    "password": {
                      "type": "string",
                      "format": "password",
                      "example": "secure_password"
                    }
                  },
                  "required": [
                    "user",
                    "password"
                  ]
                }
              }
            },
            "required": true
          },
          "responses": {
            "200": {
              "description": "Success",
              "content": {
                "application/json": {
                  "schema": {
                    "properties": {
                      "status": {
                        "type": "string",
```

```
                    "example": "success"
                  },
                  "data": {
                    "type": "object",
                    "properties": {
                      "userId": {
                        "type": "string",
                        "description": "User ID to be used in X-User-ID header
for requests that require authentication",
                        "example": "5df0a89f8a60e139366143e4"
                      },
                      "authToken": {
                        "type": "string",
                        "description": "Token to be used in X-Auth-Token header
for requests that require authentication",
                        "example": "ccd99a1368f9a5788f82938508a5ff94"
                      }
                    }
                  }
                }
              }
            }
          },
          "401": {
            "$ref": "#/components/responses/UnauthorizedError"
          }
        }
      }
    },
    "/{layer}/{collection}": {
      "get": {
        "tags": [
          "collections"
        ],
        "summary": "Query documents from a collection",
        "description": "Query documents from a collection.",
        "operationId": "layer_col_get",
        "parameters": [
          {
            "$ref": "#/components/parameters/layer"
          },
          {
            "$ref": "#/components/parameters/collection"
          }
        ],
        "responses": {
          "200": {
            "description": "Success",
            "content": {
              "application/json": {
                "schema": {
                  "properties": {
                    "status": {
                      "type": "string",
                      "example": "success"
                    },
                    "data": {
                      "type": "array",
```

```
                    "description": "Matching documents",
                    "items": {
                      "type": "object"
                    },
                    "example": [
                      {
                        "_id": "5df0a89f8a60e139366143e4",
                        "name": "document_1"
                      },
                      {
                        "_id": "5dc3c7319862be913bde4577",
                        "name": "document_2"
                      }
                    ]
                  }
                }
              }
            }
          },
          "401": {
            "$ref": "#/components/responses/UnauthorizedError"
          }
        },
        "security": [
          {
            "userId": [],
            "authToken": []
          }
        ]
      },
      "post": {
        "tags": [
          "collections"
        ],
        "summary": "Insert documents",
        "description": "Insert one or multiple documents. The request body must
include the document or array of documents to be inserted.",
        "operationId": "layer_col_post",
        "parameters": [
          {
            "$ref": "#/components/parameters/layer"
          },
          {
            "$ref": "#/components/parameters/collection"
          }
        ],
        "requestBody": {
          "description": "Document or array of documents to be inserted",
          "content": {
            "application/json": {
              "schema": {
                "type": "object"
              },
              "examples": {
                "single": {
                  "summary": "Insert one document",
                  "value": {
                    "name": "document_1"
```

```
                }
              },
              "multiple": {
                "summary": "Insert multiple documents",
                "value": [
                  {
                    "name": "document_1"
                  },
                  {
                    "name": "document_2"
                  }
                ]
              }
            }
          }
        },
        "required": true
      },
      "responses": {
        "200": {
          "description": "Success",
          "content": {
            "application/json": {
              "schema": {
                "properties": {
                  "status": {
                    "type": "string",
                    "example": "success"
                  },
                  "message": {
                    "type": "string",
                    "example": "Items inserted"
                  },
                  "data": {
                    "type": "object",
                    "properties": {
                      "ok": {
                        "type": "integer",
                        "format": "int32",
                        "minimum": 0,
                        "example": 2
                      },
                      "insertedIds": {
                        "type": "array",
                        "items": {
                          "type": "string"
                        },
                        "example": [
                          "5df0a89f8a60e139366143e4",
                          "5dc3c7319862be913bde4577"
                        ]
                      },
                      "err": {
                        "type": "integer",
                        "format": "int32",
                        "minimum": 0,
                        "example": 0,
                        "description": "When inserting multiple documents and
some of them are successful, *status* will be *'success'* but *data.err* could be
```

```
greater than 0; in this case, if the document(s) that produced the error had an
*_id*, it/they will be inserted into *errorIds* array"
                        },
                        "errorIds": {
                          "type": "array",
                          "items": {
                            "type": "string"
                          },
                          "example": []
                        }
                      }
                    }
                  }
                }
              }
            },
            "401": {
              "$ref": "#/components/responses/UnauthorizedError"
            }
          },
          "security": [
            {
              "userId": [],
              "authToken": []
            }
          ],
          "x-codegen-request-body-name": "body"
        },
        "patch": {
          "tags": [
            "collections"
          ],
          "summary": "Update documents",
          "description": "Update documents. The request body must include an object
or an array of objects with the *_id* and the fields to be updated of each
document.",
          "operationId": "layer_col_patch",
          "parameters": [
            {
              "$ref": "#/components/parameters/layer"
            },
            {
              "$ref": "#/components/parameters/collection"
            }
          ],
          "requestBody": {
            "description": "Document or array of documents to be updated",
            "content": {
              "application/json": {
                "schema": {
                  "type": "object"
                },
                "examples": {
                  "single": {
                    "summary": "Update one document",
                    "value": {
                      "_id": "5df0a89f8a60e139366143e4",
                      "name": "document_1_mod"
```

```
            }
          },
          "multiple": {
            "summary": "Update multiple documents",
            "value": [
              {
                "_id": "5df0a89f8a60e139366143e4",
                "name": "document_1_mod"
              },
              {
                "_id": "5dc3c7319862be913bde4577",
                "name": "document_2_mod"
              }
            ]
          }
        }
      }
    },
    "required": true
  },
  "responses": {
    "200": {
      "description": "Success",
      "content": {
        "application/json": {
          "schema": {
            "properties": {
              "status": {
                "type": "string",
                "example": "success"
              },
              "message": {
                "type": "string",
                "example": "Items updated"
              },
              "data": {
                "type": "object",
                "properties": {
                  "ok": {
                    "type": "integer",
                    "format": "int32",
                    "minimum": 0,
                    "example": 2
                  },
                  "updatedIds": {
                    "type": "array",
                    "items": {
                      "type": "string"
                    },
                    "example": [
                      "5df0a89f8a60e139366143e4",
                      "5dc3c7319862be913bde4577"
                    ]
                  },
                  "err": {
                    "type": "integer",
                    "format": "int32",
                    "minimum": 0,
                    "example": 0,
```

```
                        "description": "When updating multiple documents and
some of them are successful, *status* will be *'success'* but *data.err* could be
greater than 0; in this case, the *_id* of the document(s) that produced the
error(s) will be inserted into *errorIds* array"
                      },
                      "errorIds": {
                        "type": "array",
                        "items": {
                          "type": "string"
                        },
                        "example": []
                      }
                    }
                  }
                }
              }
            }
          },
          "401": {
            "$ref": "#/components/responses/UnauthorizedError"
          }
        },
        "security": [
          {
            "userId": [],
            "authToken": []
          }
        ],
        "x-codegen-request-body-name": "body"
      },
      "delete": {
        "tags": [
          "collections"
        ],
        "summary": "Delete documents",
        "description": "Delete all the documents.",
        "operationId": "layer_col_delete",
        "parameters": [
          {
            "$ref": "#/components/parameters/layer"
          },
          {
            "$ref": "#/components/parameters/collection"
          }
        ],
        "responses": {
          "200": {
            "description": "Success",
            "content": {
              "application/json": {
                "schema": {
                  "properties": {
                    "status": {
                      "type": "string",
                      "example": "success"
                    },
                    "message": {
                      "type": "string",
```

```
                    "example": "Items deleted"
                  },
                  "data": {
                    "type": "object",
                    "properties": {
                      "ok": {
                        "type": "integer",
                        "format": "int32",
                        "minimum": 0,
                        "example": 2
                      },
                      "deletedIds": {
                        "type": "array",
                        "items": {
                          "type": "string"
                        },
                        "example": [
                          "5df0a89f8a60e139366143e4",
                          "5dc3c7319862be913bde4577"
                        ]
                      },
                      "err": {
                        "type": "integer",
                        "format": "int32",
                        "minimum": 0,
                        "example": 0,
                        "description": "When deleting multiple documents and
some of them are successful, *status* will be *'success'* but *data.err* could be
greater than 0; in this case, the *_id* of the document(s) that produced the
error(s) will be inserted into *errorIds* array"
                      },
                      "errorIds": {
                        "type": "array",
                        "items": {
                          "type": "string"
                        },
                        "example": []
                      }
                    }
                  }
                }
              }
            }
          },
          "401": {
            "$ref": "#/components/responses/UnauthorizedError"
          }
        },
        "security": [
          {
            "userId": [],
            "authToken": []
          }
        ]
      }
    },
    "/{layer}/{collection}/{id}": {
      "get": {
```

```
        "tags": [
          "collections"
        ],
        "summary": "Get a document",
        "description": "Get a document.",
        "operationId": "layer_col_id_get",
        "parameters": [
          {
            "$ref": "#/components/parameters/layer"
          },
          {
            "$ref": "#/components/parameters/collection"
          },
          {
            "$ref": "#/components/parameters/doc_id"
          }
        ],
        "responses": {
          "200": {
            "description": "Success",
            "content": {
              "application/json": {
                "schema": {
                  "properties": {
                    "status": {
                      "type": "string",
                      "example": "success"
                    },
                    "data": {
                      "type": "object",
                      "description": "Requested document",
                      "example": {
                        "_id": "5df0a89f8a60e139366143e4",
                        "name": "document_1"
                      }
                    }
                  }
                }
              }
            }
          },
          "401": {
            "$ref": "#/components/responses/UnauthorizedError"
          }
        },
        "security": [
          {
            "userId": [],
            "authToken": []
          }
        ]
      },
      "patch": {
        "tags": [
          "collections"
        ],
        "summary": "Update a document",
        "description": "Update a document. The request body must include the
fields to be updated.",
```

```
      "operationId": "layer_col_id_patch",
      "parameters": [
        {
          "$ref": "#/components/parameters/layer"
        },
        {
          "$ref": "#/components/parameters/collection"
        },
        {
          "$ref": "#/components/parameters/doc_id"
        }
      ],
      "requestBody": {
        "description": "Fields of the document to be updated",
        "content": {
          "application/json": {
            "schema": {
              "type": "object",
              "example": {
                "name": "document_1_mod"
              }
            }
          }
        },
        "required": true
      },
      "responses": {
        "200": {
          "description": "Success",
          "content": {
            "application/json": {
              "schema": {
                "properties": {
                  "status": {
                    "type": "string",
                    "example": "success"
                  },
                  "message": {
                    "type": "string",
                    "example": "Item updated"
                  }
                }
              }
            }
          }
        },
        "401": {
          "$ref": "#/components/responses/UnauthorizedError"
        }
      },
      "security": [
        {
          "userId": [],
          "authToken": []
        }
      ],
      "x-codegen-request-body-name": "body"
    },
    "delete": {
```

```
        "tags": [
          "collections"
        ],
        "summary": "Delete a document",
        "description": "Delete a document.",
        "operationId": "layer_col_id_delete",
        "parameters": [
          {
            "$ref": "#/components/parameters/layer"
          },
          {
            "$ref": "#/components/parameters/collection"
          },
          {
            "$ref": "#/components/parameters/doc_id"
          }
        ],
        "responses": {
          "200": {
            "description": "Success",
            "content": {
              "application/json": {
                "schema": {
                  "properties": {
                    "status": {
                      "type": "string",
                      "example": "success"
                    },
                    "message": {
                      "type": "string",
                      "example": "Item deleted"
                    }
                  }
                }
              }
            }
          },
          "401": {
            "$ref": "#/components/responses/UnauthorizedError"
          }
        },
        "security": [
          {
            "userId": [],
            "authToken": []
          }
        ]
      }
    },
    "/{layer}/{collection}/field/{field}": {
      "get": {
        "tags": [
          "collections"
        ],
        "summary": "Get a specific field of the documents",
        "description": "Query documents from a collection returning only the
requested field (as well as *_id*).",
        "operationId": "layer_col_field",
        "parameters": [
```

```
        {
          "$ref": "#/components/parameters/layer"
        },
        {
          "$ref": "#/components/parameters/collection"
        },
        {
          "$ref": "#/components/parameters/doc_field"
        }
      ],
      "responses": {
        "200": {
          "description": "Success",
          "content": {
            "application/json": {
              "schema": {
                "properties": {
                  "status": {
                    "type": "string",
                    "example": "success"
                  },
                  "data": {
                    "type": "array",
                    "description": "Matching documents",
                    "items": {
                      "type": "object"
                    },
                    "example": [
                      {
                        "_id": "5df0a89f8a60e139366143e4",
                        "name": "document_1"
                      },
                      {
                        "_id": "5dc3c7319862be913bde4577",
                        "name": "document_2"
                      }
                    ]
                  }
                }
              }
            }
          }
        },
        "401": {
          "$ref": "#/components/responses/UnauthorizedError"
        }
      },
      "security": [
        {
          "userId": [],
          "authToken": []
        }
      ]
    }
  },
  "/{layer}/{collection}/{id}/field/{field}": {
    "get": {
      "tags": [
        "collections"
```

```
        ],
        "summary": "Get a specific field of a document",
        "description": "Get a document returning only the requested field (as
well as *_id*).",
        "operationId": "layer_col_id_field",
        "parameters": [
          {
            "$ref": "#/components/parameters/layer"
          },
          {
            "$ref": "#/components/parameters/collection"
          },
          {
            "$ref": "#/components/parameters/doc_id"
          },
          {
            "$ref": "#/components/parameters/doc_field"
          }
        ],
        "responses": {
          "200": {
            "description": "Success",
            "content": {
              "application/json": {
                "schema": {
                  "properties": {
                    "status": {
                      "type": "string",
                      "example": "success"
                    },
                    "data": {
                      "type": "object",
                      "description": "Requested document",
                      "example": {
                        "_id": "5df0a89f8a60e139366143e4",
                        "name": "document_1"
                      }
                    }
                  }
                }
              }
            }
          },
          "401": {
            "$ref": "#/components/responses/UnauthorizedError"
          }
        },
        "security": [
          {
            "userId": [],
            "authToken": []
          }
        ]
      }
    },
    "/range/{layer}/{collection}": {
      "get": {
        "tags": [
          "collections"
```

```
        ],
        "summary": "Query documents on a range of dates",
        "description": "Query documents on a range of dates. Returns the
documents where the corresponding field value is between the initial and end
dates.",
        "operationId": "range",
        "parameters": [
          {
            "$ref": "#/components/parameters/layer"
          },
          {
            "$ref": "#/components/parameters/collection"
          },
          {
            "name": "from",
            "in": "query",
            "description": "Initial date",
            "schema": {
              "type": "string",
              "format": "date-time"
            },
            "required": true,
            "example": "2019-11-10T07:00:00.000Z"
          },
          {
            "name": "to",
            "in": "query",
            "description": "End date",
            "schema": {
              "type": "string",
              "format": "date-time"
            },
            "required": true,
            "example": "2019-11-10T08:00:00.000Z"
          },
          {
            "name": "field",
            "in": "query",
            "description": "Name of the field on which to compare the dates",
            "schema": {
              "type": "string",
              "default": "timestamp"
            }
          }
        ],
        "responses": {
          "200": {
            "description": "Success",
            "content": {
              "application/json": {
                "schema": {
                  "properties": {
                    "status": {
                      "type": "string",
                      "example": "success"
                    },
                    "data": {
                      "type": "array",
                      "description": "Matching documents",
```

```
                    "items": {
                      "type": "object",
                      "example": {
                        "_id": "5df0a89f8a60e139366143e4",
                        "name": "document_1"
                      }
                    }
                  }
                }
              }
            }
          },
          "401": {
            "$ref": "#/components/responses/UnauthorizedError"
          }
        },
        "security": [
          {
            "userId": [],
            "authToken": []
          }
        ]
      }
    },
    "/near/{layer}/{collection}": {
      "get": {
        "tags": [
          "collections"
        ],
        "summary": "Query documents near a geographic point",
        "description": "Query documents near a geographic point. Returns the
documents where *geometry* field is less than or equal to the provided
distance.",
        "operationId": "near",
        "parameters": [
          {
            "$ref": "#/components/parameters/layer"
          },
          {
            "$ref": "#/components/parameters/collection"
          },
          {
            "name": "lat",
            "in": "query",
            "description": "Latitude",
            "schema": {
              "type": "number",
              "format": "float",
              "minimum": -90,
              "maximum": 90
            },
            "required": true,
            "example": 39.4629887
          },
          {
            "name": "lon",
            "in": "query",
            "description": "Longitude",
```

```
        "schema": {
          "type": "number",
          "format": "float",
          "minimum": -180,
          "maximum": 180
        },
        "required": true,
        "example": -0.4082334
      },
      {
        "name": "distance",
        "in": "query",
        "description": "Max. distance in meters",
        "schema": {
          "type": "number",
          "format": "float"
        },
        "required": true,
        "example": 100
      }
    ],
    "responses": {
      "200": {
        "description": "Success",
        "content": {
          "application/json": {
            "schema": {
              "properties": {
                "status": {
                  "type": "string",
                  "example": "success"
                },
                "data": {
                  "type": "array",
                  "description": "Matching documents",
                  "items": {
                    "type": "object",
                    "example": {
                      "_id": "5df0a89f8a60e139366143e4",
                      "name": "document_1"
                    }
                  }
                }
              }
            }
          }
        }
      },
      "401": {
        "$ref": "#/components/responses/UnauthorizedError"
      }
    },
    "security": [
      {
        "userId": [],
        "authToken": []
      }
    ]
  }
```

```
    },
    "/bulk/{layer}/{collection}": {
      "post": {
        "tags": [
          "collections"
        ],
        "summary": "Insert or update multiple documents",
        "description": "Insert or update multiple documents. The request body
must include the document or array of documents to be inserted or updated.",
        "operationId": "bulk",
        "parameters": [
          {
            "$ref": "#/components/parameters/layer"
          },
          {
            "$ref": "#/components/parameters/collection"
          }
        ],
        "requestBody": {
          "content": {
            "application/json": {
              "schema": {
                "properties": {
                  "data": {
                    "type": "array",
                    "description": "Array of documents to be inserted or
updated",
                    "items": {
                      "type": "object"
                    },
                    "example": [
                      {
                        "_id": "5df0a89f8a60e139366143e4",
                        "name": "document_1_mod"
                      },
                      {
                        "_id": "5dc3c7319862be913bde4577",
                        "name": "document_2_mod"
                      },
                      {
                        "name": "document_3"
                      }
                    ]
                  },
                  "init": {
                    "type": "boolean",
                    "description": "If *true*, the possible existing documents
will be deleted before *data* is inserted",
                    "default": false,
                    "example": false
                  }
                }
              }
            }
          },
          "required": true
        },
        "responses": {
          "200": {
```

```
            "description": "Success",
            "content": {
              "application/json": {
                "schema": {
                  "properties": {
                    "status": {
                      "type": "string",
                      "example": "success"
                    },
                    "message": {
                      "type": "string",
                      "example": "Item(s) inserted"
                    },
                    "data": {
                      "type": "object",
                      "properties": {
                        "ok": {
                          "type": "integer",
                          "format": "int32",
                          "minimum": 0,
                          "example": 3
                        },
                        "insertedIds": {
                          "type": "array",
                          "items": {
                            "type": "string"
                          },
                          "example": [
                            "5dc3c5855949108c68a11893"
                          ]
                        },
                        "updatedIds": {
                          "type": "array",
                          "items": {
                            "type": "string"
                          },
                          "example": [
                            "5df0a89f8a60e139366143e4",
                            "5dc3c7319862be913bde4577"
                          ]
                        },
                        "err": {
                          "type": "integer",
                          "format": "int32",
                          "minimum": 0,
                          "example": 0,
                          "description": "When inserting or updating multiple
documents and some of them are successful, *status* will be *'success'* but
*data.err* could be greater than 0; in this case, if the document(s) that
produced the error had an *_id*, it/they will be inserted into *errorIds* array"
                        },
                        "errorIds": {
                          "type": "array",
                          "items": {
                            "type": "string"
                          },
                          "example": []
                        }
                      }
```

```
                    }
                  }
                }
              }
            }
          },
          "401": {
            "$ref": "#/components/responses/UnauthorizedError"
          }
        },
        "security": [
          {
            "userId": [],
            "authToken": []
          }
        ],
        "x-codegen-request-body-name": "body"
      }
    },
    "/config/{layer}/{item}": {
      "get": {
        "tags": [
          "config"
        ],
        "summary": "Get a collection or serie configuration",
        "description": "Get a collection or serie configuration.",
        "operationId": "config_get",
        "parameters": [
          {
            "$ref": "#/components/parameters/layer"
          },
          {
            "$ref": "#/components/parameters/config_item"
          }
        ],
        "responses": {
          "200": {
            "description": "Success",
            "content": {
              "application/json": {
                "schema": {
                  "properties": {
                    "status": {
                      "type": "string",
                      "example": "success"
                    },
                    "data": {
                      "type": "object",
                      "description": "Collection or serie configuration",
                      "example": {
                        "name": "measures",
                        "geo": false,
                        "publish": [
                          "all",
                          "custom"
                        ],
                        "http": false,
                        "track": {
                          "schema": {
```

```
                        "mRID": " ",
                        "timestamp": " ",
                        "measures": {},
                        "status": " "
                      }
                    },
                    "activity": true
                  }
                }
              }
            }
          }
        }
      },
      "401": {
        "$ref": "#/components/responses/UnauthorizedError"
      }
    },
    "security": [
      {
        "userId": [],
        "authToken": []
      }
    ]
  },
  "put": {
    "tags": [
      "config"
    ],
    "summary": "Update a collection or serie configuration",
    "description": "Update a collection or serie configuration.",
    "operationId": "config_put",
    "parameters": [
      {
        "$ref": "#/components/parameters/layer"
      },
      {
        "$ref": "#/components/parameters/config_item"
      }
    ],
    "requestBody": {
      "content": {
        "application/json": {
          "schema": {
            "type": "object",
            "description": "Collection or serie configuration",
            "example": {
              "name": "measures",
              "geo": false,
              "publish": [
                "all",
                "custom"
              ],
              "http": false,
              "track": {
                "schema": {
                  "mRID": " ",
                  "timestamp": " ",
                  "measures": {},
```

```
                    "status": " "
                  }
                },
                "activity": true
              }
            }
          }
        },
        "required": true
      },
      "responses": {
        "200": {
          "description": "Success",
          "content": {
            "application/json": {
              "schema": {
                "properties": {
                  "status": {
                    "type": "string",
                    "example": "success"
                  }
                }
              }
            }
          }
        },
        "401": {
          "$ref": "#/components/responses/UnauthorizedError"
        }
      },
      "security": [
        {
          "userId": [],
          "authToken": []
        }
      ],
      "x-codegen-request-body-name": "body"
    }
  },
  "/notify/twitter": {
    "post": {
      "tags": [
        "utils"
      ],
      "summary": "Post a tweet",
      "description": "Post a tweet. [Twitter developer
credentials](https://apps.twitter.com/) needed.",
      "operationId": "twitter",
      "requestBody": {
        "content": {
          "application/json": {
            "schema": {
              "$ref": "#/components/schemas/TwitterData"
            }
          }
        },
        "required": true
      },
      "responses": {
```

```
          "200": {
            "description": "Success",
            "content": {
              "application/json": {
                "schema": {
                  "properties": {
                    "status": {
                      "type": "string",
                      "example": "success"
                    }
                  }
                }
              }
            }
          },
          "401": {
            "$ref": "#/components/responses/UnauthorizedError"
          }
        },
        "security": [
          {
            "userId": [],
            "authToken": []
          }
        ],
        "x-codegen-request-body-name": "body"
      }
    },
    "/notify/email": {
      "post": {
        "tags": [
          "utils"
        ],
        "summary": "Send an email",
        "description": "Send an email. Request body must include valid
[Nodemailer message options](https://nodemailer.com/message/).",
        "operationId": "mail",
        "requestBody": {
          "content": {
            "application/json": {
              "schema": {
                "description": "Nodemailer message options",
                "example": {
                  "from": "sender@server.com",
                  "to": "receiver@sender.com",
                  "subject": "Message title",
                  "text": "Message content"
                }
              }
            }
          },
          "required": true
        },
        "responses": {
          "200": {
            "description": "Success",
            "content": {
              "application/json": {
                "schema": {
```

```
                "properties": {
                  "status": {
                    "type": "string",
                    "example": "success"
                  }
                }
              }
            }
          }
        },
        "401": {
          "$ref": "#/components/responses/UnauthorizedError"
        }
      },
      "security": [
        {
          "userId": [],
          "authToken": []
        }
      ],
      "x-codegen-request-body-name": "body"
    }
  },
  "/resources": {
    "get": {
      "tags": [
        "resources"
      ],
      "summary": "Get files info",
      "description": "Get files info.",
      "operationId": "resources_get",
      "parameters": [
        {
          "$ref": "#/components/parameters/resources_fs"
        },
        {
          "$ref": "#/components/parameters/resources_type"
        }
      ],
      "responses": {
        "200": {
          "description": "Success",
          "content": {
            "application/json": {
              "schema": {
                "type": "array",
                "description": "Corresponding files",
                "items": {
                  "$ref": "#/components/schemas/File"
                }
              }
            }
          }
        },
        "401": {
          "$ref": "#/components/responses/UnauthorizedError"
        }
      },
      "security": [
```

```
          {
            "userId": [],
            "authToken": []
          }
        ]
      },
      "post": {
        "tags": [
          "resources"
        ],
        "summary": "Upload a file",
        "description": "Upload a new file. The file must be sent via
*multipart/form-data*.",
        "operationId": "resources_post",
        "parameters": [
          {
            "$ref": "#/components/parameters/resources_fs"
          },
          {
            "$ref": "#/components/parameters/resources_type"
          }
        ],
        "requestBody": {
          "content": {
            "multipart/form-data": {
              "schema": {
                "type": "object",
                "properties": {
                  "file": {
                    "type": "string",
                    "format": "binary"
                  }
                }
              }
            }
          }
        },
        "responses": {
          "201": {
            "description": "Success",
            "content": {
              "application/json": {
                "schema": {
                  "properties": {
                    "_id": {
                      "type": "string",
                      "example": "5df0a89f8a60e139366143e4"
                    },
                    "filename": {
                      "type": "string",
                      "example": "file_1"
                    }
                  }
                }
              }
            }
          },
          "401": {
            "$ref": "#/components/responses/UnauthorizedError"
```

```
          }
        },
        "security": [
          {
            "userId": [],
            "authToken": []
          }
        ],
        "x-codegen-request-body-name": "body"
      }
    },
    "/resources/{id}": {
      "get": {
        "tags": [
          "resources"
        ],
        "summary": "Get a single file",
        "description": "Get a single file.",
        "operationId": "resources_id_get",
        "parameters": [
          {
            "$ref": "#/components/parameters/resources_id"
          },
          {
            "$ref": "#/components/parameters/resources_fs"
          },
          {
            "$ref": "#/components/parameters/resources_key"
          }
        ],
        "responses": {
          "200": {
            "description": "Success",
            "content": {
              "application/json": {
                "schema": {
                  "type": "object",
                  "description": "File that may be downloaded, shown in the
browser, etc.",
                  "example": "File that may be downloaded, shown in the browser,
etc."
                }
              }
            }
          },
          "401": {
            "$ref": "#/components/responses/UnauthorizedError"
          }
        },
        "security": [
          {
            "userId": [],
            "authToken": []
          }
        ]
      },
      "delete": {
        "tags": [
          "resources"
```

```
      ],
      "summary": "Delete a single file",
      "description": "Delete a single file.",
      "operationId": "resources_id_del",
      "parameters": [
        {
          "$ref": "#/components/parameters/resources_id"
        },
        {
          "$ref": "#/components/parameters/resources_fs"
        },
        {
          "$ref": "#/components/parameters/resources_key"
        }
      ],
      "responses": {
        "200": {
          "description": "Success",
          "content": {
            "application/json": {
              "schema": {
                "properties": {
                  "_id": {
                    "type": "string",
                    "example": "5df0a89f8a60e139366143e4"
                  }
                }
              }
            }
          }
        },
        "401": {
          "$ref": "#/components/responses/UnauthorizedError"
        }
      },
      "security": [
        {
          "userId": [],
          "authToken": []
        }
      ]
    }
  },
  "/resources/{id}/info": {
    "get": {
      "tags": [
        "resources"
      ],
      "summary": "Get a file and its metadata",
      "description": "Get a single file, including its metadata.",
      "operationId": "resources_id_info",
      "parameters": [
        {
          "$ref": "#/components/parameters/resources_id"
        },
        {
          "$ref": "#/components/parameters/resources_fs"
        },
        {
```

```
            "$ref": "#/components/parameters/resources_key"
          }
        ],
        "responses": {
          "200": {
            "description": "Success",
            "content": {
              "application/json": {
                "schema": {
                  "$ref": "#/components/schemas/File"
                }
              }
            }
          },
          "401": {
            "$ref": "#/components/responses/UnauthorizedError"
          }
        },
        "security": [
          {
            "userId": [],
            "authToken": []
          }
        ]
      }
    },
    "/resources/{id}/metadata": {
      "patch": {
        "tags": [
          "resources"
        ],
        "summary": "Modify the metadata of a file",
        "description": "Modify the metadata of a file. The request body must
contain the metadata fields to be updated.",
        "operationId": "resources_id_metadata",
        "parameters": [
          {
            "$ref": "#/components/parameters/resources_id"
          },
          {
            "$ref": "#/components/parameters/resources_fs"
          },
          {
            "$ref": "#/components/parameters/resources_key"
          }
        ],
        "requestBody": {
          "content": {
            "application/json": {
              "schema": {
                "type": "object",
                "description": "Metadata fields to be updated",
                "example": [
                  {
                    "creationTime": "2019-11-10T07:00:00.000Z"
                  }
                ]
              }
            }
```

```
          }
        },
        "responses": {
          "200": {
            "description": "Success",
            "content": {
              "application/json": {
                "schema": {
                  "properties": {
                    "key": {
                      "type": "object",
                      "description": "Single key object, either *_id* or
*filename* depending on the provided value of *key* query param",
                      "example": {
                        "_id": "5df0a89f8a60e139366143e4"
                      }
                    },
                    "update": {
                      "type": "object",
                      "description": "Database update operation data"
                    }
                  }
                }
              }
            }
          },
          "401": {
            "$ref": "#/components/responses/UnauthorizedError"
          }
        },
        "security": [
          {
            "userId": [],
            "authToken": []
          }
        ],
        "x-codegen-request-body-name": "body"
      }
    }
  },
  "components": {
    "schemas": {
      "TwitterData": {
        "type": "object",
        "description": "Twitter developer credentials and text to be tweet",
        "properties": {
          "consumer_key": {
            "type": "string",
            "example": "twitter_consumer_key"
          },
          "consumer_secret": {
            "type": "string",
            "example": "twitter_consumer_secret"
          },
          "access_token_key": {
            "type": "string",
            "example": "twitter_access_token_key"
          },
          "access_token_secret": {
```

```
          "type": "string",
          "example": "twitter_access_token_secret"
        },
        "status": {
          "type": "string",
          "example": "Hello Twitter"
        }
      }
    },
    "File": {
      "type": "object",
      "properties": {
        "_id": {
          "type": "string",
          "example": "5df0a89f8a60e139366143e4"
        },
        "length": {
          "type": "integer",
          "format": "int64",
          "example": 10711
        },
        "chunkSize": {
          "type": "integer",
          "format": "int64",
          "example": 261120
        },
        "uploadDate": {
          "type": "string",
          "format": "date-time",
          "example": "2019-11-10T07:00:00.000Z"
        },
        "md5": {
          "type": "string",
          "example": "8414f8a9ff2b74588c754b26c408dc81"
        },
        "filename": {
          "type": "string",
          "example": "file_1"
        },
        "metadata": {
          "type": "object",
          "properties": {
            "type": {
              "type": "string",
              "example": "image/png"
            }
          }
        }
      }
    }
  },
  "parameters": {
    "layer": {
      "in": "path",
      "name": "layer",
      "required": true,
      "schema": {
        "type": "string",
        "example": "layer"
```

```
        },
        "description": "Layer ID"
      },
      "collection": {
        "in": "path",
        "name": "collection",
        "required": true,
        "schema": {
          "type": "string",
          "example": "collection"
        },
        "description": "Collection ID"
      },
      "doc_id": {
        "in": "path",
        "name": "id",
        "required": true,
        "schema": {
          "type": "string",
          "example": "id"
        },
        "description": "Document ID"
      },
      "doc_field": {
        "in": "path",
        "name": "field",
        "required": true,
        "schema": {
          "type": "string",
          "example": "field"
        },
        "description": "Field to be returned"
      },
      "config_item": {
        "in": "path",
        "name": "item",
        "required": true,
        "schema": {
          "type": "string",
          "example": "collection"
        },
        "description": "Collection or serie name"
      },
      "resources_fs": {
        "in": "query",
        "name": "fs",
        "required": true,
        "schema": {
          "type": "string",
          "enum": [
            "fs"
          ],
          "example": "fs"
        },
        "description": "Files storage bucket. Initially it only exists one called
*fs*"
      },
      "resources_type": {
        "in": "query",
```

```
        "name": "type",
        "schema": {
          "type": "string",
          "example": "image/png"
        },
        "description": "Files media type"
      },
      "resources_id": {
        "in": "path",
        "name": "id",
        "required": true,
        "schema": {
          "type": "string",
          "example": "file_1"
        },
        "description": "ID or filename of the requested file"
      },
      "resources_key": {
        "in": "query",
        "name": "key",
        "required": true,
        "schema": {
          "type": "string",
          "enum": [
            "id",
            "filename"
          ],
          "example": "filename"
        },
        "description": "Field in which to compare the given value of *id* path
param"
      }
    },
    "securitySchemes": {
      "userId": {
        "type": "apiKey",
        "name": "X-User-ID",
        "description": "User ID or App ID",
        "in": "header"
      },
      "authToken": {
        "type": "apiKey",
        "name": "X-Auth-Token",
        "description": "Authentication token or Key ID",
        "in": "header"
      }
    },
    "responses": {
      "UnauthorizedError": {
        "description": "Unauthorized",
        "content": {
          "application/json": {
            "schema": {
              "properties": {
                "status": {
                  "type": "string",
                  "example": "fail"
                },
                "message": {
```

```
                    "type": "string",
                    "example": "Unauthorized"
                }
            }
        }
    }
  }
 }
}
}
```