
# Report on COLLECTiEF Edge Node

**Project acronym:** COLLECTiEF

**Project title:** Collective Intelligence for Energy Flexibility

**Call:** H2020-LC-SC3-EE-2020-2

# Disclaimer

COLLECTiEF project has received research funding from European Union's H2020 research and innovation programme under Grant Agreement No 101033683. The contents and achievements of this deliverable reflect only the view of the partners in this consortium and the European Commission Agency is not responsible for any use that may be made of the information it contains.



| | |
|---|---|
| **Project no.** | 101033683 |
| **Project acronym:** | COLLECTiEF |
| **Project title:** | Collective Intelligence for Energy Flexibility |
| **Call:** | H2020-LC-SC3-2018-2019-2020 |
| **Start date of project:** | 01.06.2021 |
| **Duration:** | 48 months |
| **Deliverable title:** | Report on COLLECTiEF Edge Node |
| **Deliverable No.:** | D3.2 |
| **Document Version:** | 5.1 |
| **Due date of deliverable:** | 31.05.2023 |
| **Actual date of submission**: | 31.05.2023 |
| **Deliverable Lead Partner**: | Partner No. 4, ENERGY@WORK SOCIETA' COOPERATIVA A R.L. |
| **Work Package**: | 3 |
| **No of Pages**: | 62 |
| **Keywords**: | Field communication, Thermal optimization, Flexibility management, Human Building Interface and Visualization, Collective Intelligence, Data processing, Data storage, Complexity Management, Edge Computing, Real time processing, Fault tolerance, Scalability, Interoperability |



This project has received funding from the European Union's H2020 research and innovation programme under Grant Agreement No 101033683

| Name | Organization |
|---|---|
| Giuseppe Mastandrea | E@W |
| Marco Antonio Insabato | |
| Luigi D'Oriano | |
| Giuseppe Rocco Rana | |
| Amedeo Ingrosso | CETMA |
| Jens Brage | NODA |
| Muhammad-Salman Shahid | CSTB – G2ELab |
| Salvatore Carlucci | CyI |
| Panayotis Papadopoulos | |
| Ehsan Naghiaei | VIRTUAL |
| Alfredo Astori | LSI - LASTEM |
| Vahid Nik | ULUND |

**Dissemination level**

| | |
|---|---|
| PU | Public |



This project has received funding from the European Union's H2020 research and innovation programme under Grant Agreement No 101033683

**History**

| Version | Date | Reason | Revised by |
|---|---|---|---|
| 1.0 | 03.04.2023 | Deliverable structure draft | Giuseppe Mastandrea, E@W |
| 1.1 | 04.04.2023 | Methodology defined | Giuseppe Mastandrea, E@W |
| 2.0 | 05.05.2023 | First Version | Marco Antonio Insabato, E@W, Giuseppe Mastandrea, E@W |
| 2.1 | 09.05.2023 | Internal Review and second version | Giuseppe Mastandrea, E@W, Luigi D'Oriano, E@W, Giuseppe Rocco Rana, E@W |
| 3.0 | 18.05.2023 | Partners Contribution | Jens Brage, NODA, Ehsan Naghiaei, VIRTUAL, Salvatore Carlucci, CyI, Panayiotis Papadopoulos, CyI, Muhammad-Salman Shahid, CSTB-G2ELab, Alfredo Astori, LSI-LASTEM, Vahid Nik, ULUND |
| 3.1 | 24.05.2023 | Partners Contributions integration & 3rd version ready for Peer Review | Giuseppe Mastandrea E@W, Marco Antonio Insabato E@W, Luigi D'Oriano, E@W |
| 4.0 | 26.05.2023 | Document review | Greta Tresoldi LSI-LASTEM, Alfredo Astori LSI-LASTEM, Marco Rivolta, LSI-LASTEM |
| 4.1 | 26.05.2023 | Updated version | Giuseppe Mastandrea, E@W |
| 5.0 | 28.05.2023 | Final Version | Mohammadreza Aghaei NTNU, Giuseppe Mastandrea, E@W |
| 5.1 | 30.05.2023 | Document approval | Mohammadreza Aghaei, NTNU |



This project has received funding from the European Union's H2020 research and innovation programme under Grant Agreement No 101033683

# Executive Summary

The deliverable D3.2 related to Task 3.2 and entitled "Report on COLLECTiEF Edge Node", describes the first version of the Edge Node and the integration of the COLLECTiEF platform with the field level devices.

The aim of this deliverable, which relies on the description of the first stage of the development of the Edge Node, is one of the crucial steps for the effective success of the integration of the project solution on the different pilot sites.

To this end, starting from the aim of the COLLECTiEF project and considering as a basis both, the work already conducted in WP2 and the architecture defined in D3.1, the work done to develop the Edge Node able to operate on the different pilot sites and to communicate with the external components, specifically, the Cluster Node and the central Database, has been described in this document.

Particularly, the overall approach and methodologies followed for the development of all the connectors with field devices, for the management of the local database, for the coordination of the algorithms for energy flexibility management and thermal comfort optimization and the communication with the upper layers, have been reported in this document.

Thanks to this, it was possible to develop the first version of the Edge Node (namely the BRiG device) and to start the testing phase before installing the device in the G2Elab for small-scale demonstration and in the different pilot sites for the overall validation of the COLLECTiEF system.



This project has received funding from the European Union's H2020 research and innovation programme under Grant Agreement No 101033683

# Table of Contents





This project has received funding from the European Union's H2020 research and innovation programme under Grant Agreement No 101033683

This project has received funding from the European Union's H2020 research and innovation programme under Grant Agreement No 101033683

# List of Acronyms

| | |
|---|---|
| A | Alarms |
| AC | Air Conditioning |
| API | Application Programming Interface |
| BMS | Building Management System |
| BRiG | Border Router + iGateway (Edge Node) |
| CI | Collective Intelligence |
| CIRL | Collective Intelligence Reinforcement learning |
| CO2 | Carbon Dioxide |
| COLLECTiEF | Collective Intelligence for Energy Flexibility |
| CPU | Central Processing Unit |
| CSV | Comma-separated values |
| D | Deliverable |
| DB | DataBase |
| DSM | Demand-Side Management |
| EN | Edge Node |
| G2Elab | Green'ER Building |
| HTTP | Hypertext Transfer Protocol |
| ID | Identification |
| JSON | JavaScript Object Notation |
| K | Kelvin |
| LoRaWAN | Long Range Wide Area Network |
| M | Meter |
| MQTT(S) | Message Queue Telemetry Transport (Secure) |
| ORM | Object-Relational Mapping |
| PIN | Personal Identification Number |
| POE | Post-Occupancy Evaluation |
| REST | REpresentational State Transfer |
| RL | Reinforcement learning |
| S | Schedule |
| SQL | Structured Query Language |
| SRI | Smart Readiness Indicator |
| T | Temperature |
| TDY | Typical Downscaled Year |
| W | Watt |
| Wi-FI | Wireless Fidelity |
| WP | Work Package |



This project has received funding from the European Union's H2020 research and innovation programme under Grant Agreement No 101033683

# List of Figures





This project has received funding from the European Union's H2020 research and innovation programme under Grant Agreement No 101033683

# List of Tables





This project has received funding from the European Union's H2020 research and innovation programme under Grant Agreement No 101033683

# 1 Introduction

The purpose of this deliverable, as the technical output of the project, is to present the first version of the Edge Node and integration with field devices of the COLLECTiEF distributed Cluster-Edge platform. The deliverable describes the steps and actions performed until the M24 to develop the first version of the COLLECTiEF Edge Node and can be considered a key element for the whole COLLECTiEF system as it represents the element capable of interfacing all the components of the COLLECTiEF system with the field devices across the small-scale real environment in G2Elab and the other three project pilot sites that represent different operational environment at large-scale.

Throughout the document, the main specifications and developments of the COLLECTiEF Edge Node are described in the scope of addressing the COLLECTiEF objectives and its innovation potential.

## 1.1 Scope and objectives of the deliverable and relevance in the COLLECTiEF framework

In this deliverable, the first version of the COLLECTiEF Edge Node, as well as the integration with field devices of the whole COLLECTiEF platform, will be described. This document provides a concrete description of all the developments of all the development activities conducted so far with reference to the role of the Edge Node in the whole COLLECTiEF system architecture defined in D3.1.

## 1.2 Structure of the deliverable

D3.2 “Report on COLLECTiEF Edge Node” consists of eight chapters, in which the first version of COLLECTiEF Edge Node has been described as follows:

- **Chapter 1** presents the general description of the scope and objectives of the deliverable.
- **Chapter 2** describes the definition of the scheme of the Edge Node with reference to the different project pilots by providing a high-level description of all the components which constitute the Edge Node and a description of how they are deployed on the considered physical device, the Raspberry PI 4.
- **Chapter 3** presents all the developments that have been carried out to ensure communication with the field devices of the whole COLLECTiEF system.
- **Chapter 4** describes the Broker that has been set up to enable communication inside the Edge Node and toward external components such as Cluster Node and Central DB.
- **Chapter 5** presents the Database designed and used at the Edge Level to manage all the entities associated with the field data and all the data that will run through the COLLECTiEF system.
- **Chapter 6** describes the algorithms for the Collective Intelligence Demand Side Management and Building Thermal Optimization in consideration of the user comfort that will reside on the Edge Node.
- **Chapter 7** presents the local user Dashboard that will be deployed on the Edge Node.
- **Chapter 8** provides the conclusions of the overall work.



This project has received funding from the European Union's H2020 research and innovation programme under Grant Agreement No 101033683

# 2 Edge Node Architectural scheme definition

This chapter presents a comprehensive functional schema for the Edge Node, represented by the BRiG device constituted by the Border Router and iGateway deployed on Raspberry PI4, within the Edge-to-Cluster COLLECTiEF architectural scheme for energy management in buildings. The primary objective of the Edge Node is to efficiently gather, store, and process data from the building, enabling the implementation of algorithms for local flexibility management and enhancing user thermal comfort. Additionally, the Edge Node serves as a host for the graphical interface, facilitating user interaction, and ensuring seamless communication with the upper layers, specifically the Cluster Node and the COLLECTiEF central DB server.

The Edge Node plays a crucial role in the overall COLLECTiEF system, acting as a key intermediary between the building and the higher-level functionalities implemented in the Cluster Node. By harnessing its capabilities, the Edge Node effectively contributes to optimizing energy consumption and enhancing overall building performance.

One of the key functionalities of the Edge Node is data collection. It serves as a data aggregator, gathering information from various sensors and devices deployed throughout the building. This data encompasses vital parameters related to energy consumption, environmental conditions, and other relevant metrics. By collecting and distributing this information both towards the internal components and the upper layers, the Edge Node enables comprehensive monitoring and analysis, forming the foundation for the CI-based decision-making process. The Edge Node also assumes responsibility for both short-term data storage that happens locally and long-term data storage through the interaction with the central DB Server, ensuring secure and efficient retention of the collected information.

Particularly, the iGateway component, through the internal MQTT Broker and the Cluster Handler Client, has the functionality of connecting together the Edge Node with a number of sources at the application level through MQTT communication protocol. As shown in Figure 1 this happens by interfacing the Cluster Handler Client and consequently, the MQTT broker with the field devices through the Hub Core that collects data from the field by using the different communication protocols used to connect with the devices (MQTT, REST API, WEB scraping[1]). Something similar happens for the Sphensors for which there is their own gateway component which is able to connect them through MQTT over Threads physical protocol to communicate with the MQTT Broker.

iGateway performs the function of middleware and connector to the rest of the COLLECTiEF architecture, as well as ensures that the algorithms receive the proper data on request, and pushes it to the other components. This allows the implementation of the local algorithms for the local flexibility management and thermal comfort optimization by coordinating their implementation with the Cluster Node and the possibility to show, to the local user through the GUI, historical and trends analysis. Hence, by leveraging advanced data processing techniques, the Edge Node can generate real-time insights, enabling prompt response to changing building conditions and improving overall energy efficiency.

Finally, the Edge Node acts as a host for the local graphical interface, providing users with a user-friendly platform to interact with the field. Through this interface, building occupants can access relevant information, control settings, and monitor their energy consumption patterns. This empowers users to actively engage in energy-saving practices and enhance their thermal comfort within the building environment. To this end, the work conducted in this task, Task T3.2, has started

[1] https://it.wikipedia.org/wiki/Web_scraping



This project has received funding from the European Union's H2020 research and innovation programme under Grant Agreement No 101033683

from the general COLLECTiEF Edge-to-Cluster system architecture defined in D3.1 [1] to define the functional schematic reported in Figure 1. The reported functional schematic has acted as a basis for all the development activities conducted so far in T3.2 and it will be continuously updated during the next 6 months in which the current version of the Edge Node will be improved to implement the Edge Node consolidated prototype expected to be released at M30 of the project.

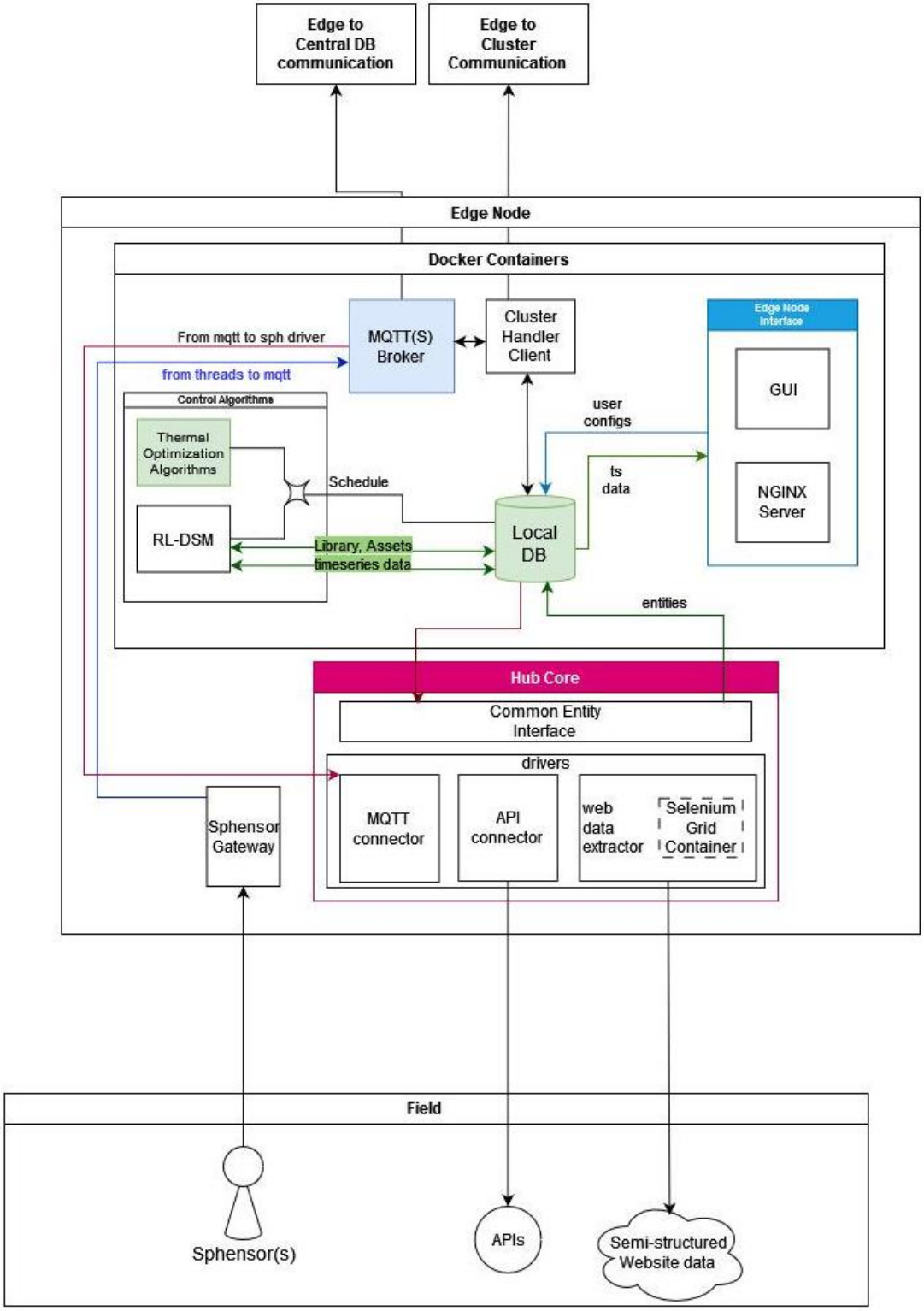


**Figure 1 Functional schematic of the Edge Node**



This project has received funding from the European Union's H2020 research and innovation programme under Grant Agreement No 101033683

## 2.1 Edge Node functional description

The heterogeneity of data and available controls for each of the cases considered in the different COLLECTiEF project pilot sites, has required severe work in understanding and categorizing the incoming data to organize it coherently within the general COLLECTiEF architecture. The incoming data were then organized in a coherent structure and specific method calls have been developed to distribute data amongst the other components of the COLLECTiEF architecture (i.e., Cluster Node, Other Edge Nodes, Central database) without them having to know data access or control methods. De facto, the Edge Node with some of its components acts as a data middleware between the data sources and the interfaces, the algorithms and the control.

In this paragraph, each pilot site will be described and illustrated in terms of data sources available while the software implementation of the data acquisition is described in detail in the subsequent paragraph (2.2) and in chapter 3 as well as the data structure in chapter 5.

Particularly, in this chapter, for each of the pilot sites, the COLLECTiEF system deployment will be reported with a high-level schematic, with reference to the specific communication infrastructure set up for each pilot site. The legend reported in Figure 2 shows the representation of the different external components and of the flows of data:

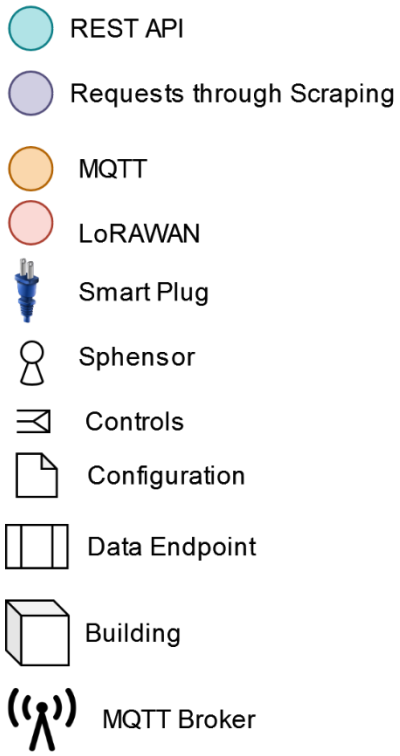


**Figure 2 Schematic functional representation of each pilot: Followed Legend**

### 2.1.1 Grenoble, Green'ER Building (small-scale demonstration site), France

The Green'ER Building is the most developed in terms of sensor and actuation, among all the pilots, since it is included to be ready to perform, in a pre-pilot environment, the testing of the different components of the COLLECTiEF architecture.

The APIs were ready from an early stage, and they are mostly related to the Sginterop platform, that operates at the Green'ER Building (G2Elab), and allows to interact with the field for gathering and controlling through the following data sources:



This project has received funding from the European Union's H2020 research and innovation programme under Grant Agreement No 101033683

- **SGInterop Data Structure:** groups the data in a table that contains each sensor of the room independently. Such a structure is well defined in the relevant assets by the name of the columns, which becomes a timeseries of its own instead of being part of the original CSV file used in the column. The available data through the sensor are:
  - **Indoor temperature**
  - **Heating energy**
  - **Cooling energy**
  - **Water flow**
  - **Inlet water temperature** (available in 7 out of 8 rooms)
  - **Outlet water temperature** (available in 7 out of 8 rooms)
  - **CO2 concentration** (available in 7 out of 8 rooms)

  While, the controllable parameters from Sginterop are:
  - **Thermostat setpoint**
  - **AC setpoint**
  - **CO2 setpoint**
- **Sphensors**: Other than acting as a fundamental data source, they may act as redundancy for parameters such as the $CO_2$ concentration, in case of malfunctioning of the SGInterop platform.

Thanks to this pilot, it has been possible to plan a strategy of configuration-driven development easily reusable in the other pilots. In Figure 3 the schematic functional representation of the communication infrastructure based on the deployment of the Edge Nodes in the Grenoble's G2ELab pilot site is reported.

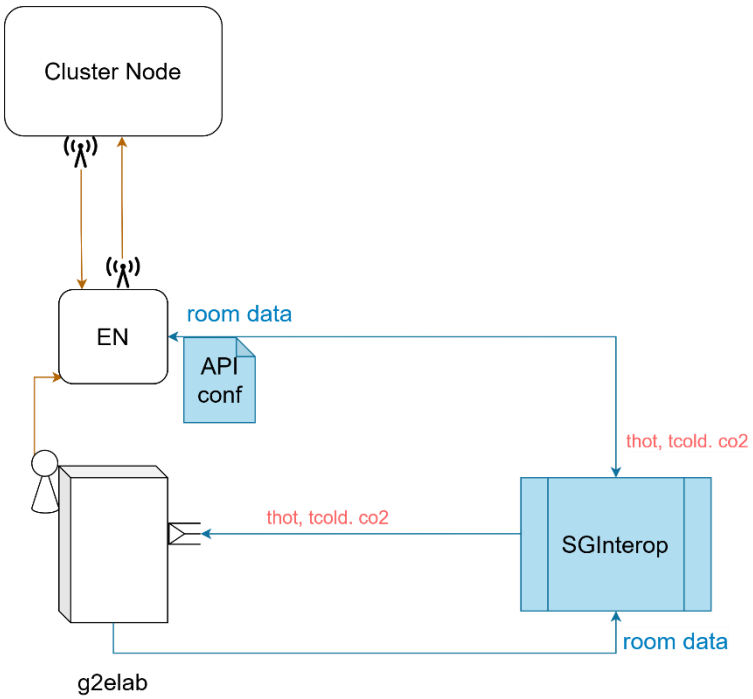


**Figure 3 Schematic functional representation of Grenoble's G2ELab pilot site**



This project has received funding from the European Union's H2020 research and innovation programme under Grant Agreement No 101033683

### 2.1.2 Milan, Residential Buildings, Italy

The Milan site is composed of three residential buildings, covering up an area of 3706 $m^2$. Each apartment is equipped by a set of sensors and controls, namely:

- **Smart valves**: The smart valves control the thermal radiator setpoints and activations. They're connected through LoRaWAN to the APIs made available by the local utility company A2A;
- **Smart plugs**: The smart plugs control some of the available appliances, as well as calculate their energy consumption. Just like the smart plugs, they are accessible the APIs made available by the local utility company A2A, allowing for binary actuation;
- **Heat Cost Allocators**: The heat cost allocators contain the data about how the thermal energy is allocated to each room. Normally not available through internet protocols, their data has to be accessed through the portal of the manufacturer, ISTA, and gathered through automatic data gathering mechanisms based on data scraping techniques;
- **Sphensors**: The Sphensors are the most common appliance in all sites, used to gather indoor environmental measurements, they are already accessible through the Sphensor Gateway component thanks to the specific Sphensor Hub Core developed by LSI LASTEM.

In Figure 4 the schematic functional representation of the communication infrastructure based on the deployment of the Edge Nodes in the Italian pilot site is reported.

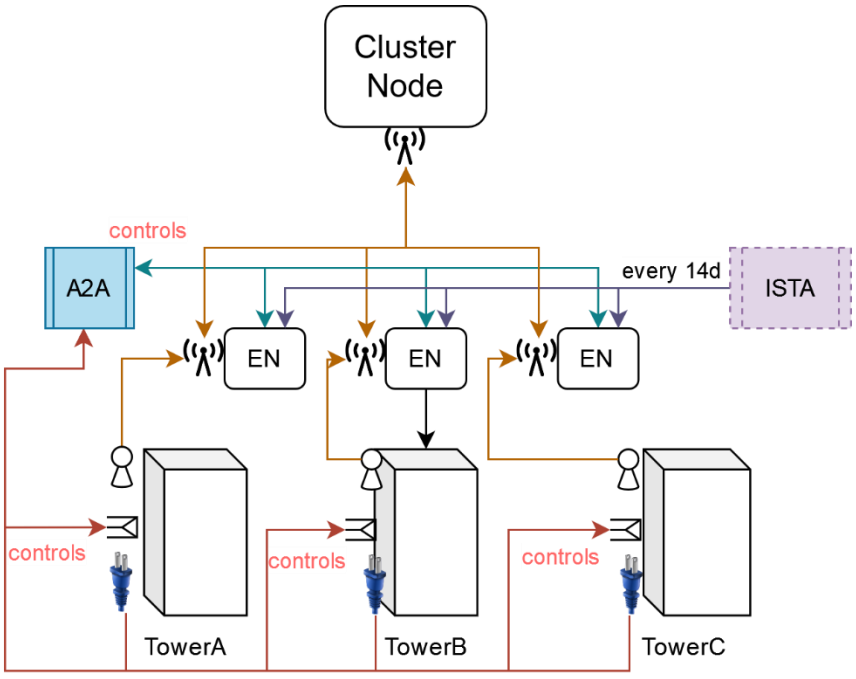


**Figure 4 Schematic functional representation of Italian pilot site**

### 2.1.3 Ålesund, Public Buildings, Norway

The Ålesund site is characterized by a more uniform setup in terms of sensors data retrieval and control, while also being extended in terms of acquired data through other systems.

In particular, the interaction with the field for the gathering of the consumption data takes place mainly through the BMS managed by the EM Systemer partner with whose APIs the Edge NodeEdge Node communicates.



This project has received funding from the European Union's H2020 research and innovation programme under Grant Agreement No 101033683

The devices with which the Edge NodeEdge Node communicates to interact with the different building at the Norwegian pilot site are listed below:

- **EM-Systemer BMS**: The entire system is controlled by a building management system that allows the calling of a number of sensors and controllers according to their typological “zone” (not to be confused with the physical concept of zones). The API shares its current status for each sensor, as well as allows controls through specific value. Such controls will be specified in Chapter 3. Aside from these data classifications, the spatial disposition is only available through the web portal, which has been studied in order to determine the spatial disposition of the algorithms. The BMS has its own methods to control the temperature setpoints, as well as allowing the users to refuse remote control in some cases. This feature requires to be considered in the algorithms within the Edge Node.
- **Sphensor data:** Sphensor devices are the most common appliance in all sites, used to gather indoor environmental measurements, they are already accessible through the Sphensor Gateway component thanks to the specific Sphensor Hub Core developed by LSI LASTEM.
- **Shelly Smart Plugs**: used in order to monitor and control the energy consumption and loads of some of the appliances. The appliances to be connected to the plugs have to be carefully selected according to local regulation in terms of food safety and comfort, for example considering refrigerator for food preservation, or appliances that have a frequent manual usage and interaction by the inhabitants. The Shelly smart plugs make use of both the HTTP REST protocol and the MQTT protocol over-Wi-Fi physical protocol in order to share their data as well as to switch them on or off.

In Figure 5 the schematic functional representation of the communication infrastructure based on the deployment of the Edge Nodes in the Norwegian pilot site is reported. Particularly, the communication with the BMS APIs is expected to be configured to ensure that only the needed data is requested.



This project has received funding from the European Union's H2020 research and innovation programme under Grant Agreement No 101033683

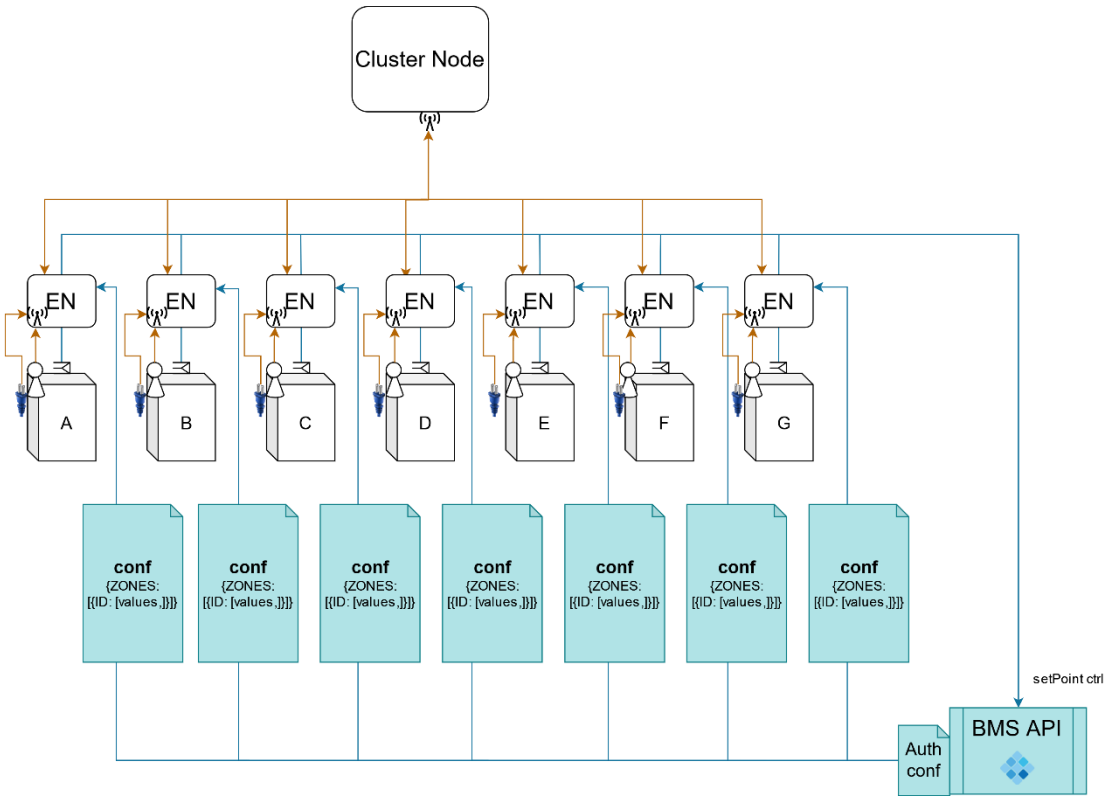


**Figure 5 Schematic functional representation of Norwegian pilot site**

### 2.1.4 Nicosia, University Buildings, Cyprus

The main data sources for this pilot are the following:

- **Ecobee** [2] smart thermostats: they have yet to be installed, their development has been possible through a test device and the API documentation.
- **Sensibo** [3] AC units: they have yet to be installed, the driver was developed through the test device and the API documentation (Sensibo).
- **Power meters**: available by Web scraping the manufacturer website, named *nrgportal*, in practice only for the Graduate School building. The data has a frequency of 15 minutes, and considers the data of a number of rooms, evaluating consumption for items such as
  - Fan Coil Units for each room
  - Heat Pumps
  - Lighting
  - Power by floor
  - Power lights for new building wing
- **Sphensors** are available here as well

Even if, as of today, the Ecobee and Sensibo devices are not installed yet, the driver wrappers for the API have been developed based on the test site functionalities. Details will be described in the chapter 3.

In Figure 6 the schematic functional representation of the communication infrastructure based on the deployment of the Edge Nodes in the Cypriot pilot site is reported.



This project has received funding from the European Union's H2020 research and innovation programme under Grant Agreement No 101033683

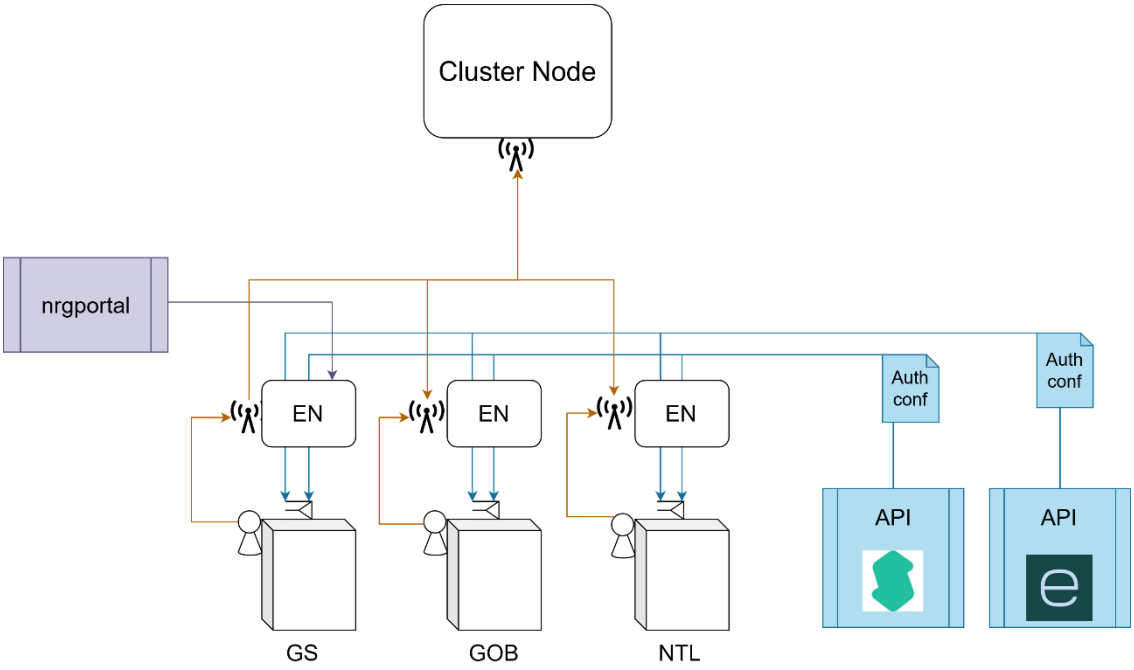


**Figure 6 Schematic functional representation of Cypriot pilot site**

## 2.2 SW Modules High Level description

### 2.2.1 Sphensor Gateway

The Sphensor Gateway (sg) module implements the standard operation of the software operating in the standard LSI LASTEM product. The specificities due to the use of this module within BRiG relate to the management of configuration and operation diagnostics, so that these two functions are aligned and functionally compatible with the corresponding functions of the other modules operating within the device.

#### 2.2.1.1 MQTT topic structure

The topic structure is adapted from the initial internal LSI LASTEM structure into the COLLECTiEF structure. The gateway, therefore, acts as an adapter that goes from the original LSI LASTEM topic structure to the COLLECTiEF architecture structure.

### 2.2.2 Hub Core

The internal functions of Hub Core are summarized in the following points:

- Main manager of MQTT messages published by the broker, specifically:
  - System configuration commands.
  - Field equipment management commands for measuring and switching signals.
  - Service commands.
  - Passive reception of incoming messages from Sphensor equipment and generated by the sg module with the native format of this product family; reconversion of MQTT messages from the native Sphensor format to that defined in this specification and related to the COLLECTiEF project.
  - Functional analysis of the various modules running within BRiG, generating statistical data on their operation, recording diagnostic messages in the system log, and publishing statistics to the MQTT broker.

The function of reconverting messages containing Sphensor data is subject to the filtering of these messages based on the master information contained in the internal database. Unregistered



This project has received funding from the European Union's H2020 research and innovation programme under Grant Agreement No 101033683

Sphensor devices are therefore ignored, except for any log entries in the system (partial, not continuous) evidencing events of receiving messages from Sphensor that are not registered in the system.

### 2.2.3 Communication Drivers

All the data sources are handled through Python scripts called drivers, that act as adapters for the entities either of the Hub Core or of other components not handled by the Hub Core. The role of the adapter is not only handled for receiving data but also for sending commands, so that any component can be handled with the same software interface (common API), regardless of the communication technology, endpoint, or authorization required to access it. This eases the pressure on modifying the architecture for each sensor or actuator component.

### 2.2.4 External Communication Modules

As previously described, the objective of the Edge Node is to receive data from heterogenous sources and store it, forward it to other components, as well as present different interfaces that adapt to each case. For the purposes of this project, it may include other Edge Nodes, Cluster Nodes and auxiliary components for data sources that are not handled directly by the Hub Core.

### 2.2.5 Database

The database in question is a MySQL derived database, precisely MariaDB, that contains specific tables related to each component in order to provide an effective separation between the resources shared among different modules. The relational model is not followed completely due to the semi structured data format that different data sources as well as components entail. Using a NoSQL strategy is not ideal for dealing with timeseries data, so it was limited to configurations, algorithm definition as well as settings.

### 2.2.6 Control Algorithms

The control algorithms are developed according to the modes set by the project, specifically according to different policies towards comfort, energy saving or flexibility in extreme conditions. The challenge regarding the Edge Node is not just be ensuring the exchange but also keeping track of the mode of transmission and handling the scheduling according to the case.

### 2.2.7 Edge Node Interface for User Communication

The local interface runs on the Edge Node allowing seamless interaction between users and the system. It provides a user-friendly and responsive platform that enables efficient control and access to functionalities, directly on the device. This interface empowers users to effortlessly navigate and manipulate the system, utilizing the computational power of the BRiG device to deliver a smooth and intuitive experience on data visualization and with field devices interaction.

## 2.3 Deployment aspects

### 2.3.1 Local configuration environment for runtime application

The configuration environment for runtime application depends on the setup of the building the local Edge Node is monitoring and controlling in terms of data sources available. In order to ensure that the data connection to the system is active, there has to be a common understandable format for each interface connection in order to handle each case regardless of the type of assets, behavior or system. Because of this, concepts of config-driven programming have been implemented in the general architecture, so that the code can be dynamically called on the basis of what the locally available systems are.



This project has received funding from the European Union's H2020 research and innovation programme under Grant Agreement No 101033683

Namely, the configuration files contain information about the following:

- Identifications of the available sensors, actuators and their specifications in terms of precision, range and time constraints.
- Specific configuration of each sensor/actuator and data source for parsing and getting proper authorizations for the platforms and the sensor access.
- Spatial organization of those sensors in relation to the monitored areas and which parameters are effective for what.
- Dynamic constructed methods to obtain aggregated demand, indoor temperature or other components transparently with respect to external components.

These initial configurations are applied at first with the use of JSON files, that contain the relevant information. After the initial configuration of the JSON file is complete, the BRiG will import such configuration in the database. Methods for loading and dumping such configurations are also set up to allow interoperability, change of configurations as well as improvement of scalability.

#### 2.3.1.1 Assets

The assets configuration is tasked, organizing the way the system handles the configuration structure for each driver. Specifically, it handles the initial asset configuration in terms of nomenclature, spatial organization within the building as well as zone definition.

This configuration has been defined so that the algorithms can work in a generalized format rather than requiring a complete code remake for each pilot's algorithms.
All data formats follow a precise schematic, where the **_comment** serves the purpose of describing the pilot in general. The pilot's name is the descriptive human name while the uuid is used for identification purposes within the COLLECTiEF architecture.
The overall JSON file structure organizes the pilot into buildings, defined by its **brigId** (although that can be debated). In each building, the brigId is repeated within the field in case of redundancy or in case of changes.
The **buildingUuid** is used for recognition between Cluster Node and Edge Node communication.
The **zones** field is where the rooms are actually described. Each zone element contains a number of subfields. Each zone is defined by a unique ID, according to the installation pilot zones, or yet another **uuid**. The description discriminates between sensors and actuators to be used in the control algorithms according to the respective schedule. The description for the sensors and the actuators is made by a key-value description, derived from the **ent_id,** and the driver's name that associates the way to reach the measurements to the identifiers for each sensor. This description is called by the algorithms in order to easily query the necessary timeseries regardless their names. The **ent_id** is the entity identifier, while the driver is the script to retrieve the data. Anything else can be adopted for the purposes of identification. Next, within each sensor, there are a number of tags that describe the kind of measures to be retrieved from the database. The actions available for the actuators are described in the default description, which can be a floating-point value, if so, the minimum and maximum setpoints are added. The default field itself can also be an object that contains all the default values, for instruments with multiple settings. Other features can be described in options, that describe the fields and the permitted values, that can depend on what the API allows. The final goal of this system is to generalize data access as much as possible with respect to the identification. The components don't have to know names, measurements, and access methods, since the configuration files contains this information for the algorithm establishment. The methods can be generalized as much as possible as long as the naming standards remain consistent between pilots.



This project has received funding from the European Union's H2020 research and innovation programme under Grant Agreement No 101033683

**Table 1 assets.json for handling the algorithm monitoring and control**

```
{
  "{{brigId}}": {
    "_comment": "Comment",
    "brigId": "{{Brig serial}}",
    "buildingName": "{{Pilot name}}",
    "buildingUuid": "{{uuid}}",
    "zones": {
      "{{zoneId}}": {
        "zoneUuid": "{{dataset}}",
        "zoneName":"{{name}}",
        "zoneDesc":"{{description}}",
        "sensors": {
          "{{ent_id}}-{{driver}}": ["{{tag}}-{{channel}}"]
        },
        "actuators": {
          "{{ent_id}}-{{driver}}": {
            "default": "{{default settings}}",
            "settings": {
              "values": {
                "min": "{{min}}",
                "max": "{{max}}"
              },
              "features": {
                "option_1": "{{value}}",
                "option_2": "{{value}}"
              }
            }
          }
        }
      }
    }
  }
}
```

In case of slow performance of the JSON file elaboration, further optimizations will be performed in order to generalize the setup.

The General schematic of the JSON settings prototype is reported in Figure 7.



This project has received funding from the European Union's H2020 research and innovation programme under Grant Agreement No 101033683

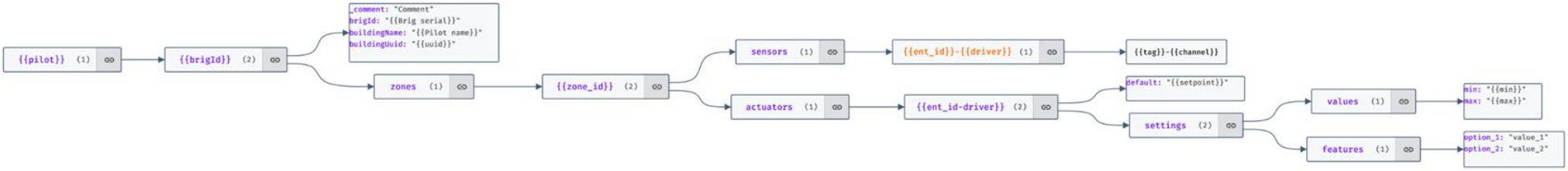


**Figure 7 General schematic of the JSON settings prototype**



This project has received funding from the European Union's H2020 research and innovation programme under Grant Agreement No 101033683

#### 2.3.1.2 Library

The library data handles the memory of the DSM algorithm, which will be properly described in the relevant database entry. The original library is held in a file that contains the signal and actuator setpoints to be used in the algorithm. After library data is imported into the Edge Node, either from external component or from the value, it is indexed according to zone of interest, season and hour of the day. Such a standardization will also be applied for other algorithms, in order to properly select the value of interest.

### 2.3.2 Module containerization

Docker is an open-source platform that enables software developers to package their applications and their dependencies into containers. Containers are lightweight, standalone, and portable units that encapsulate everything needed to run an application, including the code, runtime, system tools, libraries, and settings. Docker provides a consistent environment for running applications, regardless of the underlying operating system or infrastructure.

One important concept in Docker is volumes. Volumes in Docker provide a way to persist and share data between containers and the host machine. A volume is a directory within a container or on the host machine that is specially designated to store data. By using volumes, data can be shared and accessed by multiple containers, allowing for better separation of concerns and more flexible management of data.

Volumes have several advantages in Docker. They provide a persistent storage solution, allowing data to survive container restarts and even container deletion. Volumes also enable data sharing between containers, making it easier to implement microservices architectures where different containers need to communicate and share data. Additionally, volumes can be used to store configuration files, logs, or any other type of data that needs to be accessed or shared by containers.

Docker's utilization for software containerization brings several benefits. Firstly, it simplifies the deployment process by eliminating the need to worry about dependencies and environment compatibility. Applications packaged in Docker containers can be easily reproduced and deployed on different machines, reducing compatibility issues and ensuring consistency across different environments.

Secondly, Docker enables efficient resource utilization. Containers are isolated from each other and share the host machine's operating system kernel, which allows for running multiple containers on the same host without the need for dedicated virtual machines. This improves resource efficiency and reduces overhead.

Moreover, Docker facilitates scalability and portability. Containers can be easily scaled horizontally by running multiple instances of the same containerized application, allowing for efficient handling of increased workload. Furthermore, Docker containers can be seamlessly migrated between different environments, such as development, testing, and production, without the need for extensive reconfiguration or modifications.

Overall, Docker provides a powerful platform for software containerization, offering developers an efficient and flexible way to package, deploy, and manage applications while ensuring consistency, scalability, and portability across different environments.

For the module containerization in the COLLECTiEF Edge Node, Docker has been utilized.

Indeed, the majority of the components of the BRiG device at the Edge side is associated with a Docker volume, related to files that are passed to the container from the host machine. Each of these



This project has received funding from the European Union's H2020 research and innovation programme under Grant Agreement No 101033683

volumes is put into a number of containers, as well as, making use of auxiliary containers in order to avoid any issues with writing and reading the database values.

The main containers are described as follows. Other containers may be deployed on a site-to-site basis.

1. Containers for the individual MQTT brokers: Each Edge Node and each Cluster Node will have a deployed broker with specific configurations that can be adapted depending on the setup. Every container has a configuration file attached as a volume that describes the needed options, as well as potential files for authentication and file handling. In case of dynamic security use, such settings are to be handled within the containers.
2. RL – DSM and Thermal Comfort algorithms: the thermal comfort algorithms have been containerized in order to handle a variable case to case scenario. For connecting with the database through an ORM, the DSM has been modified for that specific purpose and configuration files are added as a data volume.
3. Scraper Grid Unit: the scripts to handle scraping are dealt with as a driver, with the sole difference being that in order to send the data they make use of the previous MQTT modules, that saves and outputs the overall data within the database. It is a grid unit because SELENIUM [4] containers may be one or more on the grid and handle multiple Chrome Instances.
4. Incompatible drivers: in case of fully developed drivers such as those from G2ELab, the drivers are wholesale containerized and the data is extracted and sent through MQTT in order for the iGateway to save them and reuse them.
5. The iGateway component handles messages as a container, either forwarding them to the Cluster Node for the database functions as well as handle the connection to the central NTNU database for forwarding. Between the Edge Node and the Cluster Node it will act as an adapter as well as assigns identification for each entity to the external architecture. For this purpose, translation tables, asset management and such will handle the connections, the chronology of the exchanged timeseries and so on through the same database as the other components.



This project has received funding from the European Union's H2020 research and innovation programme under Grant Agreement No 101033683

# 3 Field level communication

In this chapter, we will explore the various components and mechanisms involved in field-level communication within COLLECTiEF system providing more details regarding the components introduced in the previous section.

## 3.1 Sphensor Gateway

This module maintains standard operating characteristics in order to meet the following needs:

- Allow the use of the LSI LASTEM programs already built and available for configuration of Sphensor devices (sensors, repeaters);
- Maintain compatibility with the data logging systems currently in operation within the COLLECTiEF project in order to continue the collection of baseline metrics even after the implementation of the new MQTT message management formats, thus without having to modify the programs already made.

## 3.2 Hub Core

The Hub Core is one of the few components that remains external with respect to the docker containerization, with each component exchanging data with the local database through the use of an ORM [5]. The ORM will not only perform connections to the database through object representation but also perform common writing and reading procedures for other components, making the database models something that crosses between each component of the database, allowing the components to navigate through the database tables to retrieve the data of interest.

Most of the components require a configuration in order to be accessed, namely the MQTT broker's credentials, the database credentials, the topics to subscribe to in order to retrieve the data, and finally any API keys that will be required in order to access the data.

It connects to each data source by defining them as entities: each entity is characterized by being a Measure Entity, Control Entity, or both.

Measure entities are tasked with retrieving measurements, setting the unit of the measurements as well as saving new ones with a polling mode. The Hub Core messages are split into a head section and a payload section. The header always has the same fields, just like in the rest of the BRiG.

Once the measured entity is defined, the data retrieval mechanism makes use of a scheduler that handles all the polling of each entity, regardless if it is a control or a reading, by adding them to a message pool. Depending on the time of the message, the scheduler handles the message differently. In case of malformed messages, these can be dismissed. These schedulers are run for both Sphensor entities as well as external entities or other messages that act independently from the rest of the system.

The Hub Core has a number of tables that are described in the relevant paragraph and are used in order to access the data accordingly. These tables are also accessed by other components, however only the hc_* (DB table that deals with the Hub Core, see section 5) can handle writing directly, by using the MQTT broker as the common interface for all messages.

It also performs logging functionalities in case of disconnection, bad packet format or packet loss. The overall structure of the Hub Core is comprised of general configuration for the database, the MQTT broker, as well as the current machine setup.



This project has received funding from the European Union's H2020 research and innovation programme under Grant Agreement No 101033683

### 3.2.1 MQTT Connector

Most of the operations that the Hub Core performs require some form of communication with the other components and to orchestrate their commands and their functions. With this in mind, the Hub Core deals with exchanging messages following the typical methods of subscription, publishing, as well as handling messages. The MQTT topics are structured as follows:

*collectief/brig_id/ent/ent_id/class/trig*

where:

- **collectief**: determines whether the message belongs to the functions and devices inherent in the COLLECTiEF project;
- **brig_id**: defines the identifier of the BRiG device generating or receiving the message; more than for programs within it, this information is useful for external devices that, having to be connected to different BRiGs, can classify incoming messages according to this identifier. The identifier is uniquely defined by the user when configuring the BRiG device.
- **ent**: indicates the type of entity to which the message refers; BRiG manages, via MQTT messages, both for system (internal) entities, present in single instance and managed directly, and for non-system (external) entities, used to manage external apparatus or information systems; the latter entities are managed via appropriate drivers. The identity type name can apply to:
    - **brig**: message related to BRiG in its functional entirety.
    - **brig_mb**: system identity; message related to the mb functional module of BRiG.
    - **brig_hc**: system identity; message related to BRiG functional module hc.
    - **brig_sg**: system identity; message related to function module sg of BRiG.
    - **brig_ig**: system identity; message related to BRiG function module ig.
    - **brig_db**: system identity; message related to BRiG function module db.
    - **brig_ng**: system identity; message related to the ng function module of BRiG.
    - **field_xxx**: external entity related to a physical apparatus installed in the field; message is related to a measurement apparatus and/or an on/off or proportional actuation apparatus, managed by BRiG via a communication system (media and/or communication protocol) other than Thread; managed directly by BRiG's hc function module. The entity type is specified by the suffix xxx, which coincides with the code of the BRiG internal driver dedicated to message management (e.g., *sph* for Sphensor sensors, or *shly* for Shelly sensors).
    - **serv**: entity external to the system; message related to a measurement system and/or an on/off or proportional actuation system connected via network protocol to an information service running in a remote network server; managed directly by BRiG's hc and ig function modules.
- **ent_id**: identifier of the specific entity to which the message refers. Depending on the type of entity, it is either meaningfully valorized with a unique identifier local to BRiG and related to the entity itself or, for internal entities, valorized with the same value as ent when the message refers to unique entities in the system since, in this case, the brig_id field contained in the same topic already provides unique identification. Finally, consider also the possibility of setting the value of ent_id to any, with the purpose of indicating any entity managed by the system and related to the entity type specified by ent.
- **class**: indicates the class of the message. Its applicability depends on the section of the opi cent_id. In general, it may apply:



This project has received funding from the European Union's H2020 research and innovation programme under Grant Agreement No 101033683

  - **registry**: messages inherent to the registry management of multiple entities managed by BRiG (device registration); to this category belong the measurement and actuation tools that belong to the specific BRiG.
  - **config**: configuration parameters specific to the specified entity.
  - **diag**: message inherent to the general diagnostic information managed by the specified entity.
  - **cmd**: command (output setting, system restart, etc.) sent to the specified entity.
  - **date**: generic indicative of data carried by the message, used when the terms given by the previous points are not semantically suitable.
- **trig**: indicates how the message was generated; may apply:
  - **req**: generated for the purpose of obtaining a certain (asynchronous) response.
  - **ans**: generated in response to a message with trig = req.
  - **event**: generated spontaneously, unprompted, as it is produced on the basis at a certain event or following specific timing.

### 3.2.2 API Connector

API connector in terms of the Edge Node COLLECTiEF architecture is a schematic abstraction of most drivers that are considered in the pilot sites of this project (i.e., SGInterop, Ecobee, Sensibo Sky). In fact, most of the work that the Edge Node does is interfacing the general architecture with external data sources, working as a middleware for the rest of the architecture. The reason for using API connections at the application layer, rather than relying on lower-level protocols, stems from the necessity of using multiple heterogenous data sources with a different protocol stack from the transport layer and below, relying on the advantages that the REST APIs offer in terms of accessibility to the endpoints. Any sort of downside that is implied by using APIs may be the one concerning extra latency in terms of data reception. Such an issue may not be as preponderant due to the frequency that data arrives with, usually being at around fifteen minutes, which covers even slow reception or connections; in that case, it's better to deal with resilience with respect to missing data through interpolation, as well as acknowledge that the domain of IoT does not expect data connections to be perfectly reliable and fast at all times.

With this introduction out of the way, what is meant by API connector is but a special case of the common entity interface, where the entity in question has methods to control, poll for data as well as detect the status of the sensors, as well as rely on the general Hub Core in order to be queued with the other entities, whatever they may be.

Any API however, has a number of parameters that are required in order to be accessible by the Hub Core component of the Edge Nodes, that may be keys or credentials. Such credentials are stored in the database as entities. Entities are then queried and turned into objects that are self-contained in terms of methods, code, and credentials. Mostly relying on the request's python library, it is sufficient.

Such interfaces also need methods to select the needed data and, thus, have a configuration for selecting which data series can be saved and which cannot be saved, this also being part of the configuration setup. For the most part, the API connectors are sufficient for most purposes and have to rely on the polymorphism of the custom entity interface rather than being built ex-novo.

In general, in these cases, data sources are associated with an endpoint. A description of how the data is recovered through these mechanisms from the different devices present in the pilot sites that operate in this way is reported below.



This project has received funding from the European Union's H2020 research and innovation programme under Grant Agreement No 101033683

#### 3.2.2.1 Norway's enportal BMS

Norway's EM Systemer BMS is a legacy system that handles all the pilot buildings in Norway in terms of monitoring, control and scheduling of the heating houses. Being the most developed, it has the following basic data taxonomy, starting from the API calls, following this structure

```
https://emportal.no/api/2/Service/[Function]/[Param1]/…/[ParamN]
```

There are a number of functions that perform data retrieval as well as control depending on how the API call is performed. **[Function]** usually refers to a number of functions provided by the API; namely:

- **GetMany:** used to get all the values for a zone, with the following call and parameters: GetMany/ZoneType/Recursion/OnlyLiveValues
  - **ZoneType**: Zone type (between S, M, T and A)
  - **Recursion** (between -1,0,1), depending on the value it shows more detailed information
  - **OnlyLiveValues** (bool, either true or false), applicable if the Recursion is set at -1, in order to further reduce the JSON value
- **GetOne:** used to get the value by a single sensor by ID, with the following call and parameters:
  GetOne/ZoneType/ID/Recursion/OnlyLiveValues
  - **ZoneType**: Zone type (between S, M, T and A)
  - **ID**: ID of the sensor, maximum value depends on what sensors are available
  - **Recursion**
  - **OnlyLiveValues**
- **SetOneFloatResource:** used to set the value of a parameter, in case the zone

The possible zone types are:

- (**T**)emperature: temperature data of the room, as well as current setpoints available for those rooms
- (**M**)eter: meter data, which gives either power (2 s frequency) or energy readings (1 m frequency)
- (**S**)chedule: here are stored variables for scheduling the setpoints according to profiles, as well as the timing
- (**A**)larms: control of the alarms for many facilities, windows, and all sensors that deal with binary values (on/off)

For each zone, the data structure obtained varies depending on the recursion value. For future data retrieval, most of these fields will be removed in order to only get what is strictly necessary for the algorithms to operate. Finally, it is important to define the configuration data that is necessary for deploying the Edge Node solution when using this API.

The configuration will be a JSON formatted string that contains the following fields:

- Header: the following fields will be saved in the configuration for access on the building:
  - api_key_org: organization key
  - Api_key: key for the building
  - Content-Type: always **application/json**
- Sensors:



This project has received funding from the European Union's H2020 research and innovation programme under Grant Agreement No 101033683

- A mapping of the IDs to each zone and room, as well as the relevant measurements to save in the database
- The driver will be handled as a single entity containing a large number of data, determined by the kind of incoming data series. Any spatial disposition will be handled by the dsm_assets table explained in the section 5.

**Table 2 Example value received from a "M" zone request**


```
{
  "ID": 1,
  "ZoneLetter": "M",
  "Description1": "Description1",
  "Description2": "Description2",
  "MeasureType": "kWh",
  "MesureTypePerHour": "kW",
  "Consumption_Last2min": 0,
  "Consumption_CurrentHour": 0,
  "Consumption_LastHour": 0,
  "Current_MaxValue": 9999
},
```


**Table 3 "T" zone data**


```
{
  "ID": 1,
  "ZoneLetter": "T",
  "Description1": "Description1",
  "Description2": "Description2",
  "MaxNumberOfZones": 1500,
  "MaxValue": 30,
  "MinValue": -20,
  "MaxAllowed": 9999,
  "MinAllowed": -10000,
  "DaySetPoint": 10,
  "DeltaTemperature": 0,
  "LowSetPoint": 17,
  "ClosedSetPoint": 11,
```




This project has received funding from the European Union's H2020 research and innovation programme under Grant Agreement No 101033683

```
"PMin": 20,
"Type": 2,
"Acceleration": 0,
"AccelerationStr": "",
"Unit": "°C",
"InndorRef": 0,
"OverrideStatus": "",
"BackColorString": "FFFFFF",
"ForeColorString": "000000",
"ActualValue": 8.0,
"SetValue": 0,
"Gain": 0,
"OutdoorTemperature": 8.0001112234155,
"RefZoneLetter": "",
"RefZoneID": 0,
"HasEffectCalculation": 0,
"HasEffectRegulation": false,
"IsEffectRegulationOn": false,
"SetpointLogging": {
  "Name": "T0001 Setpoint",
  "Hours": 1,
  "Minutes": 0,
  "IsActive": false,
  "Type": "SetPoint"
},
"ActualValueLogging": {
  "Name": "T0001.AddedValue",
  "Hours": 1,
  "Minutes": 0,
  "IsActive": true,
  "Type": "ActualValue"
},
```




This project has received funding from the European Union's H2020 research and innovation programme under Grant Agreement No 101033683

```
    "GainLogging": {
      "Name": "T0001.Impact",
      "Hours": 1,
      "Minutes": 0,
      "IsActive": false,
      "Type": "Gain"
    }
  }
```


**Table 4 “A” zone data**


```
{
    "ID": 1,
    "ZoneLetter": "A",
    "Description1": "Description1",
    "Description2": "Description2",
    "AlarmActive": false
},
```


After this in-depth description of the API calls of this system, the next point to raise is how the relevant data will be obtained and categorized for each room. Aside from the API, enportal BMS has also provided us with the room disposition of these sensors through the web portal, which has allowed us to organize the incoming data spatially for the algorithms to actually work. Namely, especially T zone and A zone data, as well as dealing with aggregate data coming from the M zone for energy consumption. This allows to have a layer of transparency between this original data source and the rest of the architecture, reducing the information load to the strictly necessary data.

#### 3.2.2.2 Ecobee Thermostats

The Ecobee AC units are accessed initially through a manual procedure that will require the user help in order to allow the user commands to work (Figure 8). The initial setup only contains an API key that needs to be sent with the following method (as explained in the [6]). After the API key has sent an authorization request, the user will receive a PIN that he will need to input in the relevant app section.



This project has received funding from the European Union's H2020 research and innovation programme under Grant Agreement No 101033683

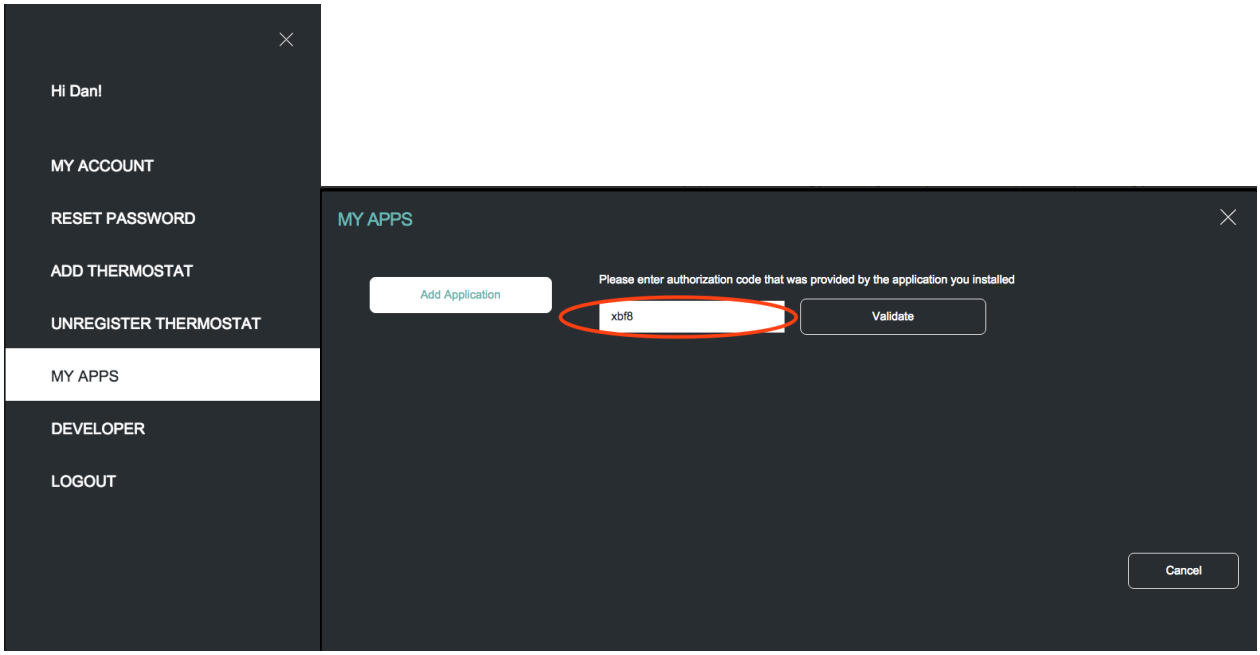


**Figure 8 Ecobee app section for entering login credentials**

Once the application has been authorized, the configuration will be modified with the received **access tokens** and **refresh tokens**. The former allows access to the central Ecobee database for 60 minutes, while the latter allows access for up to one year. Once the access has been granted, these tokens have to be periodically updated in order to keep operating. In this sense, the driver will need to perform, in addition to their data polling, a periodical polling of new access token every 60 minutes, as well as a long-term polling. The refresh token request must be performed every 60 minutes in order for the access to continue. In case of disconnection, the refresh token remains valid for up to one year and can be reused whenever needed to get an access token.

The frequency of API usage limit is up to once every three minutes, with the max number of contemporary HTTP requests being three.

The API endpoints to be used in this case are

**[POST]** https://api.ecobee.com/token

This API endpoint is used in order to renew the refresh and access tokens. The header requires the following payload in order to grant a new set of tokens:

{'grant_type': 'refresh_token', 'code': <<refresh_token>>, 'client_id': <<api_key>>}

In the configuration of the driver, the refresh_token and api_key values are sufficient in order to generate a new access token to be used. It will be valid for an hour until a new set of tokens will be required.

**[GET] https://api.ecobee.com/1/thermostat?json=<<selection>>**

The API requests thermostat data according to a JSON selection parameters, which can be left empty in order to get the full data. The field requires a selection payload that can accept a set of parameters to specify how to receive the response. In our case the parameter is as follows:



This project has received funding from the European Union's H2020 research and innovation programme under Grant Agreement No 101033683

Table 5 JSON selection parameters for data retrieve from Ecobee devices

```
{
    "selection":{
        "selectionType":"registered",
        "selectionMatch":"",
        "includeRuntime": true,
        "includeEquipmentStatus": true,
        "includeWeather":true
    }
 }
```

Following the Selection object reference, this payload means that all data arrives from the registered thermostats for a given area, assuming that those thermostats are selected. The selectionMatch parameter refers to a match based on the selectionType, with the empty string being a wildcard. The includeRuntime parameter, instead, is a Boolean parameter that includes the extended thermostat runtime object, that contains the thermostat identifications as well as any revision that might have happened over time.

The typical response payload, excluding the status and the weather fields, is:

Table 6 Ecobee typical response payload

```
{
  "page": {"page": 1, "totalPages": 1, "pageSize": 1, "total": 1},
  "thermostatList": [
    {
      "identifier": "413721917137",
      "name": "B09Z06",
      "thermostatRev": "230516141554",
      "isRegistered": true,
      "modelNumber": "nikeSmart",
      "brand": "ecobee",
      "features": "Home,HomeKit",
      "lastModified": "2023-05-16 14:15:54",
      "thermostatTime": "2023-05-17 15:58:00",
      "utcTime": "2023-05-17 12:58:00",
      "runtime": {
        "runtimeRev": "230517125228",
        "connected": true,
        "firstConnected": "2022-10-07 12:17:41",
        "connectDateTime": "2023-05-17 01:52:34",
        "disconnectDateTime": "2023-05-17 01:35:57",
        "lastModified": "2023-05-17 12:52:28",
        "lastStatusModified": "2023-05-17 12:52:28",
        "runtimeDate": "2023-05-17",
        "runtimeInterval": 153,
        "actualTemperature": 801,
        "actualHumidity": 40,
        "rawTemperature": 801,
```



This project has received funding from the European Union's H2020 research and innovation programme under Grant Agreement No 101033683

```
        "showIconMode": 0,
        "desiredHeat": 662,
        "desiredCool": 806,
        "desiredHumidity": 36,
        "desiredDehumidity": 60,
        "desiredFanMode": "auto",
        "actualVOC": -5002,
        "actualCO2": -5002,
        "actualAQAccuracy": 0,
        "actualAQScore": -5002,
        "desiredHeatRange": [450, 790],
        "desiredCoolRange": [650, 920]
      }
    }
  ],
  "status": {"code": 0, "message": ""}
}
```


After the data has been received, some parameters have to be processed, in order to obtain the actual measurements and save them in the database. Specifically, the parameters “actualTemperature”, “rawTemperature”, “desiredHeat” and “desiredCool” have to be rescaled and translated following this relation:

$$T = \frac{(x - 320) \cdot 5}{90}$$

Where x is the temperature in tenths of Fahrenheit degrees and T is the output temperature in Celsius degrees.

**[GET] https://api.ecobee.com/1/thermostatSummary?json=<<selection>>**

This API request is for frequent polling, in order to detect any revision in terms of data for the individual thermostat. This method handles if there has been any revision for the given thermostats. For the most part, the use of this command has not been completely clarified until there will be a need to reduce the polling time to considerably less than 15 minutes.

#### 3.2.2.3 Sensibo AC Units

The Sensibo AC units have a simpler authorization format that only requires an api_key parameter in order for the driver to be authorized to retrieve data. Whenever a Sensibo client driver is initialized, it has to send a device request, whose content is then put in the cfg field of the entity that describes the sensors and their components. During the initialization of the Sensibo client, the data about the devices is retrieved with a specific request. After that, the data for each device can be retrieved and each device can be controlled, based on request of the algorithm, through the relative commands:

**[GET] https://home.sensibo.com/api/v2/users/me/pods**

The following endpoint serves the purpose of retrieving the pods by ID. Considering that its values are mostly fixed, the driver will call this element rarely and, instead, occasionally update the device with a bi-monthly schedule or when the user changes or removes an AC unit.



This project has received funding from the European Union's H2020 research and innovation programme under Grant Agreement No 101033683

From this all the other requests become available:

**[GET] https://home.sensibo.com/api/v2/pods/<<device_id>>/acStates**

Get the current status of the AC unit in terms of settings, it can be used to determine in the algorithms how to change the behavior.

**Table 7 GET response data payload example for Sensibo AC device**


```
{
  "timestamp": {"time": "2023-05-17T14:42:22.073471Z", "secondsAgo": 0},
  "on": false,
  "mode": "cool",
  "targetTemperature": 26,
  "temperatureUnit": "C",
  "fanLevel": "auto",
  "swing": "stopped",
  "horizontalSwing": "stopped",
  "light": "on"
}
```


The following method may allow setpoints in the algorithms to be reused through the next request.

**[POST] https://home.sensibo.com/api/v2/pods/<<device_id>>/acStates**

This request serves the purpose of changing the state of the device for any of the parameters. In case of complete actuation, this may be necessary and useful, otherwise, the next PATCH request can be employed.

**Table 8 Example of POST for device status change**


```
{
    "acState": {
      "on": false,
      "mode": "string",
      "fanLevel": "string",
      "targetTemperature": 0,
      "temperatureUnit": "string",
      "swing": "string"
    }
  }
```


The following system only serves the purpose of specifically changing one of the current values instead of sending a post request. Both POST request and PATCH request have been implemented to serve the Edge Node driver in order to allow a high compatibility and flexibility with the APIs.

[PATCH] https://home.sensibo.com/api/v2/pods/<<device_id>>/<<property_to_change**>>**

The following GET method allows to find the latest measurement obtained from the output:

**[GET] https://home.sensibo.com/api/v2/pods/<<device_id>>/measurements**

The values of these measurements have the following format.



This project has received funding from the European Union's H2020 research and innovation programme under Grant Agreement No 101033683

Table 9 Example of current value retrieved via specific GET from Sensibo AC


```
{
  "time": {"time": "2023-05-17T14:41:39.924057Z", "secondsAgo": 42},
  "temperature": 26.5,
  "humidity": 40.8,
  "feelsLike": 26.5,
  "rssi": -55
}
```


These measurements mostly overlap with the Ecobee sensors, however with a lower precision. In order to allow the drivers to be as flexible as possible for the system, these measurements will still be retrievable by the driver, in case the algorithm development for Cyprus requires it.

Historical Measurements is another method to retrieve historical data in bulk, for up to five days before the request.

**[GET] https://home.sensibo.com/api/v2/pods/<<device_id>>/historicalMeasurements**

The message, follows this layout:

Table 10 Example of historical values retrieved via specific GET from Sensibo AC


```
{
  "status": "success",
  "result": {
    "temperature": [
      { "time": "2023-05-16T00:01:08Z", "value": 21.9 },
      { "time": "2023-05-16T00:02:38Z", "value": 21.9 },
      { "time": "2023-05-16T00:04:08Z", "value": 21.9 },
      { "time": "2023-05-16T00:05:38Z", "value": 21.9 },
      …
    ],
    "humidity": [
      { "time": "2023-05-16T00:01:08Z", "value": 65.7 },
      { "time": "2023-05-16T00:02:38Z", "value": 65.8 },
      { "time": "2023-05-16T00:04:08Z", "value": 65.8 },
      { "time": "2023-05-16T00:05:38Z", "value": 65.8 },
      …
    ]
  }
}
```


This request is an alternative to performing polling measurements, leaving out, however, some reinforcement learning functionalities. For the purposes of easily gathering data for training, this may be sufficient during the testing phase.



This project has received funding from the European Union's H2020 research and innovation programme under Grant Agreement No 101033683

#### 3.2.2.4 CityEye A2A Platform

The A2A platform is the platform that collects all the data from the smart plugs and smart valves selected for the Italian pilot, through LoRaWAN connection; after that, the old setup that relied on MQTT over Wi-Fi connections became untenable due to connection problems to reach the different apartments from the common areas of the buildings. New devices in the Italian Pilot have been acquired and used within the A2A's CitiEye platform. These devices replace the missing smart plugs and smart thermostat radiator valves, using LoRaWAN for the data. The two devices are:

- Vicki Smart Thermostatic Radiator Valve, that allows temperature monitoring and remote control by using a LoRaWAN system. It's powered by a battery that can last up to 10 years. In order to save energy, the device is implemented as a Class A device, meaning that, in order for commands to be sent, the control mechanisms must wait for downlink windows.
- Enginko EGK-LW22PLG Smart plug allows controls based on time. Being connected to the home plugs, it does not have to save energy and it is instead a class C device that can send data with a higher frequency. On-off controls are allowed.

These devices are accessible through the use of the CityEye A2A platform, that also includes an API endpoint to access measurements, and control them. Some of the calls to be used in the drivers will be shown and described in order to showcase, like in the previous cases, how they will be accessed by the Edge Node and organized.

[POST] **https://api.cityeye.it/login**

This POST request serves the purpose of signing in with the configured credentials within the database. The login method has the following request body:

**Table 11 Login method request body**

```
{
    "email": "{{email}}",
    "password": "{{password}}"
}
```

The response in JSON format is:

**Table 12 JSON response to login method**

```
{
    "token": "{{token value}}",
    "organizationKeys": [
      "{{key}}",
      …
    ]
  }
```

The token will be used for future requests in the header as a Bearer token.

[GET] **https://api.cityeye.it/devices?pageSize=30&page=1&form=long**



This project has received funding from the European Union's H2020 research and innovation programme under Grant Agreement No 101033683

Get all the devices for a specific organization. This request may be used in order to obtain information about the available devices on the site. The information about devices can then be used for the algorithms on the Edge Node.

The request header is:

**Table 13 Request header to know all the accessible devices**

```
headers = {
    "Authorization": "Bearer {{token}}",
    "Content-Type": "application/json",
    "OrganizationKey": "{{organization}}"
}
```

The response is structured in this way:

**Table 14 Response to "All Devices" request**

```
[
    {
      "id": "{{id}}",
      "name": "{{name}}",
      "organizationKey": "{{key}}",
      "description": "{{description}}",
      "serial": "{{serial}}",
      "position": {
        "longitude": "{{lat}}",
        "latitude": "{{lon}}"
      },
      "address": null,
        …
      "tags": [],
      "sourceType": "{{sourcetype}}",
      "lastMeasurementAt": "YYYY-MM-DDThh:mm:ss.uuuZ",
      "activeSince": null,
      "createdAt": "YYYY-MM-DDThh:mm:ss.uuuZ",
      "public": false,
      "hidden": false,
      "quantities": [
        {
          "measureId": "{{id}}",
          "measureName": "{{temperature}}",
          "unitOfMeasure": "{{unit}}"
        },
        …
      ]
    },
    …
]
```



This project has received funding from the European Union's H2020 research and innovation programme under Grant Agreement No 101033683

The response gives a detailed description of every device for a given zone, according to the organization given in the header, in this case COLLECTiEF.

[GET] **https://api.cityeye.it/devices/<<device_id>>/measures/latest**

Used in order to get all the latest measurements in a given period of time. The id of the device is retrieved from the previous request and is used in order to retrieve the latest measurements. The optional measureName parameter is used in the GET request in order to obtain the selected value. Reducing the number of calls may make the measureName parameter not needed as much as others. Headers are the same for any other authorization. The payload can optionally have the following parameters:

**Table 15 Latest measurement request payload of a specific device**

```
payload = {
    "measureName": "{{name}}"
}
```

The response format is, in this case:

**Table 16 Example of response to latest measurement request of a specific device**

```
{
    "timestamp": "YYYY-MM-DDThh:mm:ss.uuuZ",
    "payload": {
        "humidity": 50.39,
        "motorRange": 496,
        "openWindow": 0,
        "motorPosition": 496,
        "tooLowCurrent": 0,
        "batteryVoltage": 3.4,
        "tooHighCurrent": 0,
        "targetTemperature": 19,
        "measuredTemperature": 20.94,
        "rotaryEncoderStatus": 0,
        "brokenTemperatureSensor": 0
    }
}
```

The following command is used so far to send specific downlinks to the device. The valid actions are available through the next API call.

[POST] **https://api.cityeye.it/devices/<<device_id>>/downlinks**



This project has received funding from the European Union's H2020 research and innovation programme under Grant Agreement No 101033683

A typical payload is as follows:

**Table 17 Typical payload of downlink sending to specific device**

```
{
    "payload": "{{payload}}",
    "fields": {},
    "port": 1,
    "actionName": "{{action}}",
    "downlinkAckByUplink": {{bool}}
}
```

In case of a successful response, the response is a plain device ID.

[GET] **https://api.cityeye.it/devices/<<device_id>>/actions**

Every device allows for a possible set of actions in order to send the data. In the case of the Vicky device what is possible to control is:

**Table 18 Possible controls for Vicky device**

```
{
  "actions": [
    "keepAlive",
    "recalibrateMotor",
    "openWindowDetection",
    "childLock",
    "temperatureRange",
    "forceClose",
    "internalTemperatureControl",
    "internalTempControlTDiff",
    "proportionalTemperatureControl",
    "deviceOperationalMode",
    "deviceTargetTemperature",
    "externalTemperatureRead",
    "joinRetryPeriod",
    "uplinkMessageType",
    "motorPositionTargetTemp",
    "motorPosition",
    "getValue",
    "watchDogParameters",
    "primaryOperationalMode",
    "resetDevice"
  ]
}
```

There are also other further APIs used for the communication between these devices and the BRiG such as the "Downlink action fields" and "Send downlink using action fields", used for sending commands from the BRiG to the A2A server.



This project has received funding from the European Union's H2020 research and innovation programme under Grant Agreement No 101033683

#### 3.2.2.5 SGInterop

The SGInterop driver has been extensively developed by G2ELab researchers and is mostly used within the dataset. The retrieved data comes from a number of sensors that detect different metrics, as well as diagnostics and such within the file. After a number of measures, it has been concluded that the best approach for this driver is to wholesale containerize it and send the latest data directly to the MQTT so that the writer can be handled separately from SGInterop System.

Incompatibility in this way is acceptable even if the drawback is higher waiting times.

A further connector has been developed in order to handle the case of “missing” data, which is not an instance of the malfunctioning sensor, but rather it simply requires interpolation from the latest datapoint.

### 3.2.3 Web Data Extractor connector

A Web Data Extractor connector is a powerful tool leveraging web scraping techniques to automatically extract and retrieve relevant data from websites, enabling efficient and streamlined data collection for various applications. Particularly, it utilizes HTTP methods, such as GET and POST, to interact with web servers and retrieve specific webpages or submit data to online forms, facilitating the extraction process. By sending HTTP requests and analyzing the corresponding responses, the connector can effectively navigate websites, access desired content, and extract structured data for further analysis and integration.

Occasionally, the typical HTTP methods become unreliable due to the security measures set up by the platform managers for extracting the data. This issue requires more convoluted approaches that involve simulating user interaction, in order to obtain the correct credentials, as well as download the necessary data emulating a button click on the user interface, instead of using plain http requests, due to the extra parameters required for the systems to function. Also, occasionally, valid session cookies are not provided to valid clients, requiring the use of scrapers that simulate real user access.

#### 3.2.3.1 Nrgportal

Compared to other components, the handling of this component will not be handled by the API due to the peculiarities of the data retrieval of this system. Even if energy consumption data is available from a WebSocket connection, the details of this connection have not been divulged by the installer.

Particularly, to correctly login into the website, relevant session parameters require the retrieval of a session key from a session emulation. In lieu of all these factors, a scraping approach has been chosen, making use of a Selenium Grid for the specific case of Cyprus. The potential weight that this retrieval may cause is outweighed by the usefulness that power meter data give in terms of energy demand and flexibility estimation. In the part of the portal reported in Figure 9, the login is being performed using Selenium, in order to obtain a valid session key from the server that will allow the driver to download the measurement report.



This project has received funding from the European Union's H2020 research and innovation programme under Grant Agreement No 101033683

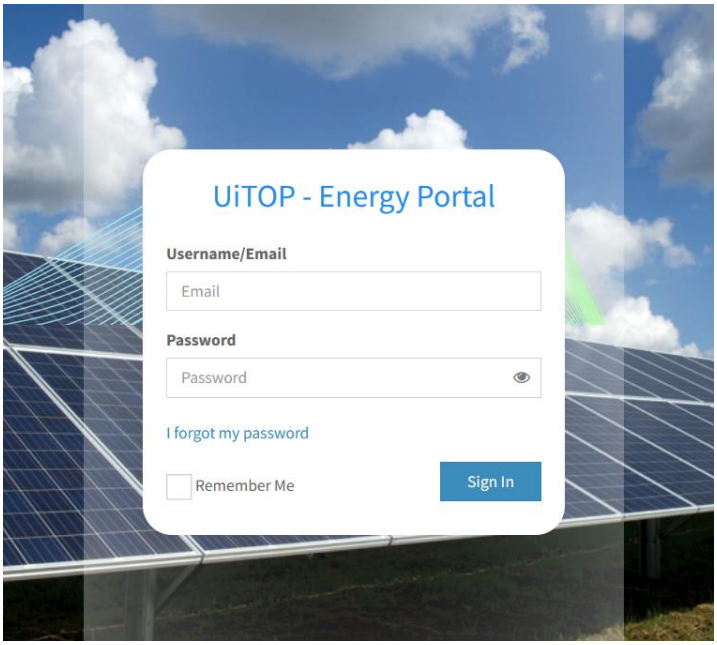


**Figure 9 Nrgportal Login interface**

The procedure for the data retrieval, accessing the main portal page (Figure 10), replaces the button click to download the data with a get request for all the parameters.


**Figure 10 Nrgportal main portal page**

### 3.2.3.2 Heat Cost Allocator, ISTA Portal

The ISTA web portal is a valuable resource that provides daily data related to the energy consumption of each radiator within the building. Despite having a slower data retrieval pace compared to other sources, with each data point being collected every day and retrieved every two weeks, its data points can still prove useful for evaluating thermal energy usage during pilot tests and for calculating energy demand based on requests. However, due to the lower frequency of data collection, some methods such as interpolation or averaging may be required to produce a more comprehensive analysis. Once retrieved, the data is provided in the form of an Excel file, which is



This project has received funding from the European Union's H2020 research and innovation programme under Grant Agreement No 101033683

accessed via a browser bot utilizing Selenium. From there, the file is processed using the Pandas library to extract the desired data—specifically, the heat energy consumed within the selected area.

The TEICOS data points (Figure 11) are used for the heat cost allocator for each radiator, and they are retrieved through the MODBUS communication protocol. It is not possible to reroute this data directly, so the machine for the heat cost allocator is going to be an external component that will relay the data directly from the local data repository to the Edge Node. This point is used to retrieve heat cost allocator data as well as environmental data.

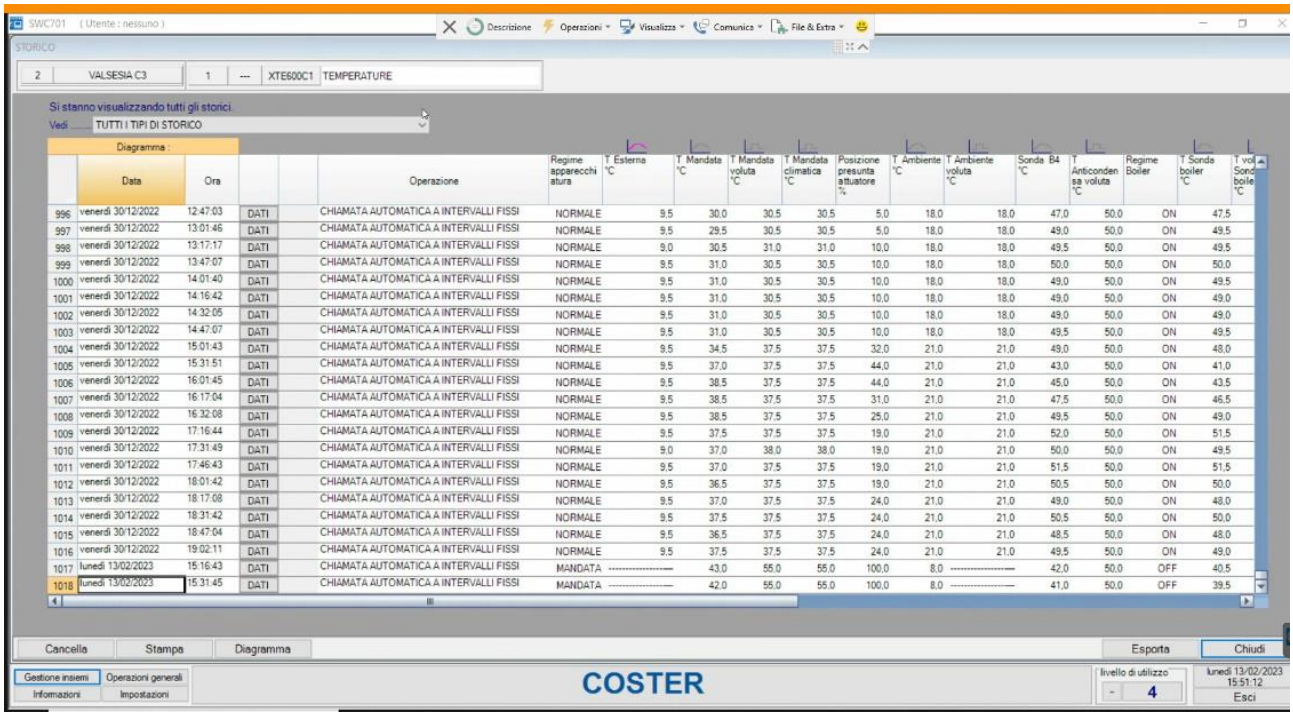


**Figure 11 Program to retrieve the data from the heat cost allocator on the local TEICOS machine**

### 3.2.4 Custom Connector Definition

The COLLECTiEF platform expects the possibility to create new connectors in an easy way. Indeed, in the case new connectors will be added, defining them should be simple and scalable by following an approach similar to what is defined in [7].

Instead of relying upon a single broker, the COLLECTiEF architecture builds redundant brokers for each element. These connectors can be custom-made depending on the case. Currently the work ongoing is for making available interfaces useful to define and add these custom connectors or for the definition of new drivers to be built, starting from the general connector definition.

### 3.2.5 iGateway Component

The iGateway component has the functionality of connecting together the Edge Node with a number of sources at the application level through MQTT, performing the function of middleware and connector to the rest of the COLLECTiEF architecture, as well as ensuring that the algorithms receive the proper data on request as well as push it to the other components. Another functionality is performing additional connections to drivers that are more structured and do not fit clearly within the entity definition due to being externally programmed. The iGateway performs the functions of connecting any external component that isn't related to data sources or such, namely:



This project has received funding from the European Union's H2020 research and innovation programme under Grant Agreement No 101033683

- Cluster Node connection, handling the incoming messages and relaying to the external messages, as well as parsing the incoming blobs
- Central database connection: for the duration of the COLLECTiEF project, there might need to relay the data to the central database for analytical purposes. The database uses a MQTT broker as the main database connection, and simply relaying any incoming messages according to a schematic is sufficient.

### 3.2.6 Common Entity Interface

#### 3.2.6.1 Custom Entity Definition

All the previously described connectors are abstracted into entities that perform the functions of handling the data source behavior and understanding. The object structure of the entity definition is strictly related to the database schema, also considering that SQLAlchemy was adopted as an ORM in order to define the tables and the possible operations for each driver in the specific case of the Hub Core, as well as other data sources. In general, all systems fall into two kinds of entities: sensors and actuators. Sensors have defined methods to retrieve data from the source, while actuators have methods to define and modify a setpoint. Each entity recalls a different driver, which is a Python class defined in accordance with the methods to retrieve that data. Entities can be further categorized according to the kind of source they deal with, mostly being:

- REST API data sources: Requires specific endpoints and requests in order to retrieve the measurements and perform the values
- MQTT Connectors: Often have another protocol from behind the scenes or another retrieval mechanism and uses MQTT messages in order to save the data values.

The general schematic of the Entity Class Hierarchy is reported in Figure 12:



This project has received funding from the European Union's H2020 research and innovation programme under Grant Agreement No 101033683

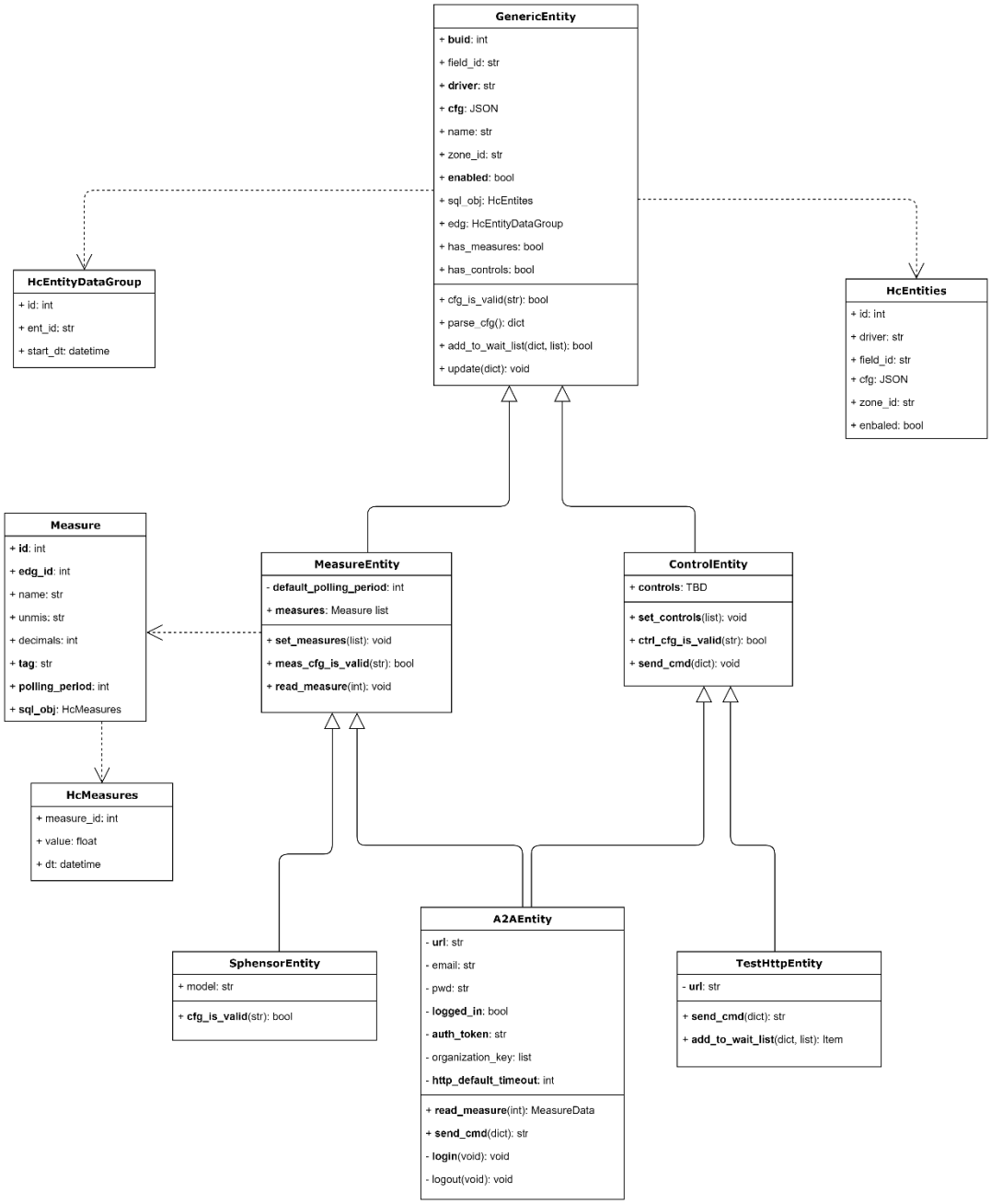


**Figure 12 General Schematic of the Entity Class Hierarchy**



This project has received funding from the European Union's H2020 research and innovation programme under Grant Agreement No 101033683

# 4 BRiG encrypted broker for data communication

## 4.1 Local Broker

This broker is responsible for the communication between the individual components, namely Sphensor Gateway, towards the Hub Core to obtain messages from the Sphensors as well as other messages from the iGateway component, that either cover some missing devices or, in general, require communication to and from the lightweight algorithms in order to work.

The local components, de facto, work as MQTT clients with their own credentials on the 1883 port. The MQTT broker is set up on a docker container that includes, within their volumes, a configuration file, as well as any authentication files used, in order to further enhance data security within the dataset.

## 4.2 Interactions and communications with the upper layers

The messages exchange between components is fundamental to actually fulfil the vision that the COLLECTiEF architecture entails: every component must function seamlessly with each other ensuring that the components work jointly. For this to work, the concept of transparency comes back to mind: the components can and must be able to communicate regardless of the differences in terms of implementation and architecture, and this can be only possible thanks to the use of distributed MQTTS brokers that exchange information using a common understandable topic format.

Aside from a common topic format, the data exchange must be transparent, and mostly agnostic, as far as data exchange is concerned. As a final note, the data exchange must be dealt with mostly as JSON blobs that are handled, breaking a bit with the relational model that is used with the overall system. As a final goal, the field that is managed by the Edge Node is transparent to the upper layers.

### 4.2.1 Edge-to-Cluster communication

The exchange of data happens through blobs whose format is not exact. What matters, in order to identify the direction of the exchange, is to make use of the topic format that the MQTT messages use and have clients for each component interested in that message flow. Using a distributed broker methodology, the topic format and the message direction determine the operations that the data is for.

The topic format for data exchange is as follows:

**{{brig_id}}/{{method}}/{{sensor}}/{{quantity}}**

- **brig_id**: The identification of the BRiG is supplied by the BRiG itself over to the Cluster Node and is used to describe the flow.
- **method**: describes what the data is for in terms of result. The values can be either measured or expected, based on the NODA solutions, or have other names based on the data exchange required for the DSM algorithm, which is **asset** for asset JSONs related to the sensor layout for the algorithm, and library for the signal library related to the algorithm.
- **sensor**: describes the sensor.
- **quantity**: the sensor credentials are mostly transparent to the Cluster Node, and it makes use of the following values in case of the flow exchange:
  - temperature [K]
  - energy_flow [W]
  - energy_cost [dimensionless quantity]
  - energy_mode [dimensionless quantity]



This project has received funding from the European Union's H2020 research and innovation programme under Grant Agreement No 101033683

These measures are not known to the Cluster Node in terms of actual source and the Edge Node simply provides the provenience case by case. The setting of these credentials is done at the Gateway level in order to obtain the measurements of interest from the database.

#### 4.2.1.1 Noda Cluster Node

The Noda Cluster Node solutions contains a framework for modelling a corresponding part of the Edge Node, complete with a communication solution which can be routed through the main MQTT broker. The solution centres around the communication of patches to a virtual global state, with each patch indicating what parts it pertains to. For efficiency, it is possible to publish the patches under a topic indicating the publisher and purpose, but it is not necessary, and the system can function without topics.

But having said that, the Edge-to-Cluster communication involves two messages, one from the Cluster Node to the Edge Node and one from the Edge Node to the Cluster Node. These can be published under the topics for control from the Cluster to Edge and for measurement from the Edge to Cluster.

The payload of the two messages conforms to the same Python data class as described in D3.4 Section 2. However, the contents differ:

- The message from the Cluster Node to the Edge Node contains two pieces of time series data: the control_energy_cost (D2.1 synthetic energy price signal) and the control_energy_mode (CIRL signal 0-5).
- The message from the Edge Node to the Cluster Node contains one piece of time series data: the measure_energy_flow [W], computed from the BRiG energy data by means of interpolating, resampling, and adding the results.

Note that, while it is possible to use the same message format to just pass the BRiG energy data along to the Cluster Node, the NODA Edge Node solution needs this aggregated/total energy data and will have to compute it anyway.

### 4.2.2 Edge-to-Central DB communication

The Edge Nodes by themselves do not have to communicate anything that is not a Cluster Node or a sensor or an actuator. However, for the purposes of this project, this kind of connection is necessary in order to perform a centralized and convenient analysis of the data. The connection to the central database is managed by the iGateway component by making frequent polling calls that retrieve the required meteorological and POE data that are saved respectively in the ig_poe_desc (description of the POE data depending on the site) and the ig_poe_series, that preserve a local copy of the POE data, in order to be used for the thermal comfort algorithms as well as for providing a local and readily available source for the data implementation

The iGateway also periodically retrieves all the available data and will, de-facto, replace the original centralized broker for all transmission purposes, further distributing the communication through multiple brokers.

.



This project has received funding from the European Union's H2020 research and innovation programme under Grant Agreement No 101033683

# 5 BRiG local database for entities management

## 5.1 Local DBMS

The database adopted is a MySQL derived database, MariaDB. This database system is established on the relational model of data management. The data is organized into tables by users, which consist of rows/records and columns/fields. MySQL provides a scalable solution for data storage. It offers various data types, that allows flexible data representation. Structured Query Language (SQL) is the primary language for interacting with the MySQL database. This allows the users to manipulate and query the data in the database.

MySQL has been chosen as the database for its effective performance, scalability, and ease of use.: It can handle large amounts of data effectively and supports techniques like indexing. Moreover, as MySQL is designed to manage simultaneous requests, that makes it suitable for the application of user interfaces.

## 5.2 Local DB schema

The local database schema makes use of several tables, each of which is used and managed by the different components of the BRiG. They are indicated by the prefixes ig_, hc_, dsm_* et cetera. Each prefix defines which Edge Node component will use that table. The tables perform the function of holding, in general, three kinds of data:

- **Timeseries data**: sensor readings as well as actuation setpoints in a given time.
- **Diagnostic data**: rates of missing, unrequested, or incoming messages.
- **Settings**: settings related to sensor or actuator access as well as algorithm scheduling and functionality.

### 5.2.1 Hub Core Tables

#### 5.2.1.1 Registry information

***hc_entities***: enables the registration of internal and external entities managed by BRiG.

| *Field* | *Type and Constraints* | *Description* |
|---|---|---|
| id | INT, AUTO_INCREMENT, PRIMARY KEY | Value generated by the database and unique identifier of the entity in the local scope of the *BRiG*; outside the database module, it is named *buid* |
| driver | VARCHAR(16), NOT NULL | Name of the software driver used to manage the entity |
| field_id | VARCHAR(64), NOT NULL | Identifier of the entity within the field systems (device serial number, measurement code in the BMS, etc.), thus not related to entities within BRiG. This identifier is unique within the driver using it, but not necessarily unique between different drivers. |
| name | VARCHAR(64) | Descriptive name of the entity |
| zone_id | VARCHAR(8), NOT NULL | The area to which the entity belongs within the COLLECTiEF project |



This project has received funding from the European Union's H2020 research and innovation programme under Grant Agreement No 101033683

| | | |
|---|---|---|
| cfg | JSON, NOT NULL | Configuration of entity access parameters; it also includes the sampling rate if entity polling is required |
| enabled | BOOL, NOT NULL, DEFAULT true | Enablement status of the entity within the operations performed by BRiG; it does not necessarily correspond to the actual enablement status of the physical entity external to BRiG |

The following constraint has been added to the table:

- "unique key" applied to columns “field_id" and "driver". This ensures that the combination of field_id+driver is unique within the local BRiG scope.

#### 5.2.1.2 Measurement data

***hc_entity_data_groups***: holds the quantity group descriptors for each defined entity that produces measured data.

| ***Field*** | ***Type and Constraints*** | ***Description*** |
|---|---|---|
| id | INT, AUTO_INCREMENT, PRIMARY KEY | Value generated by the database and unique identifier of the data group generated by an entity |
| entity_id | INT, FOREIGN KEY(hc_entities.id) | Reference to the entity that generated the data set |
| start_dt | DATETIME, NOT NULL | Indicates from which instant the specified measurement grouping is valid. This instant is defined during the entity creation or update operation, according to the parameters defined in the related command |

The following constraints have been added to the table:

- "unique key" applied to columns "entity_id" and "start_dt". This ensures that each entity can only have one “entity data group” associated with a specific datetime.
- "foreign key" linking the "entity_id" column to the "id" column of hc_entities. This associates the entity data group with the entity it refers to. If the value in the parent column is modified, the "entity_id" value in this table will also be updated (on update cascade). If the parent record is deleted, the corresponding records in this table will also be deleted (on delete cascade).



This project has received funding from the European Union's H2020 research and innovation programme under Grant Agreement No 101033683

***hc_measures***: contains the list of measures generated by a single entity.

| ***Field*** | ***Type and Constraints*** | ***Description*** |
|---|---|---|
| id | INT, AUTO_INCREMENT, PRIMARY KEY | Value generated by the database and unique identification of the measure |
| edg_id | INT, FOREIGN KEY(hc_entity_data_groups.id) | Reference to the data group to which the measure belongs |
| tag | VARCHAR(32) | Numeric or alphanumeric code identifying the measurement within the message received from or requested to the entity; this field is used to create a correspondence between the set of data available in the entity and the specific record in *hc_measures* |
| name | VARCHAR(32) | Name of the measured quantity; it may also contain additional information, such as the description of the measurement point (e.g. "north side ceiling") |
| unmis | VARCHAR(8), NOT NULL | Measurement unit |
| decimals | TINYINT(4), NOT NULL | Number of decimal places (typically from 0 to 5) used to correctly represent the measured values to the operator; the value is represented with rounding off the last decimal place specified here with respect to the next one |

The following constraints have been added to the table:

- "unique key" applied to columns "edg_id" and "tag". This ensures that the tag associated with a specific entity data group is unique.
- "foreign key" linking the "edg_id" column to the "id" column of hc_entity_data_groups. This associates the measurements with a specific entity data group. If the value in the parent column is modified, the "edg_id" value in this table will also be updated (on update cascade). If the parent record is deleted, the corresponding records in this table will also be deleted (on delete cascade).



This project has received funding from the European Union's H2020 research and innovation programme under Grant Agreement No 101033683

***hc_measure_data***: contains the values produced over time by a single measurement

| ***Field*** | ***Type and Constraints*** | ***Description*** |
|---|---|---|
| measure_id | INT, PRIMARY KEY, FOREIGN KEY(hc_measures.id) | Reference to the measure that generated the data |
| dt | DATETIME(), PRIMARY KEY | Indicates the time of measurement or, more realistically, of receipt of the measured value by BRiG |
| value | FLOAT(), NOT NULL | Measured value; if the value is a logical state, zero corresponds to the *false* state, while any other value corresponds to *true* |

The following constraint has been added to the table:

- "foreign key" linking the "measure_id" column to the "id" column of hc_measures. This associates the saved values with the measure that generated them. If the value in the parent column is modified, the "measure_id" value in this table will also be updated (on update cascade). If the parent record is deleted, the corresponding records in this table will also be deleted (on delete cascade).

The schema in Figure 13 illustrates how the hc_measure_data table is linked to the hc_entities table.

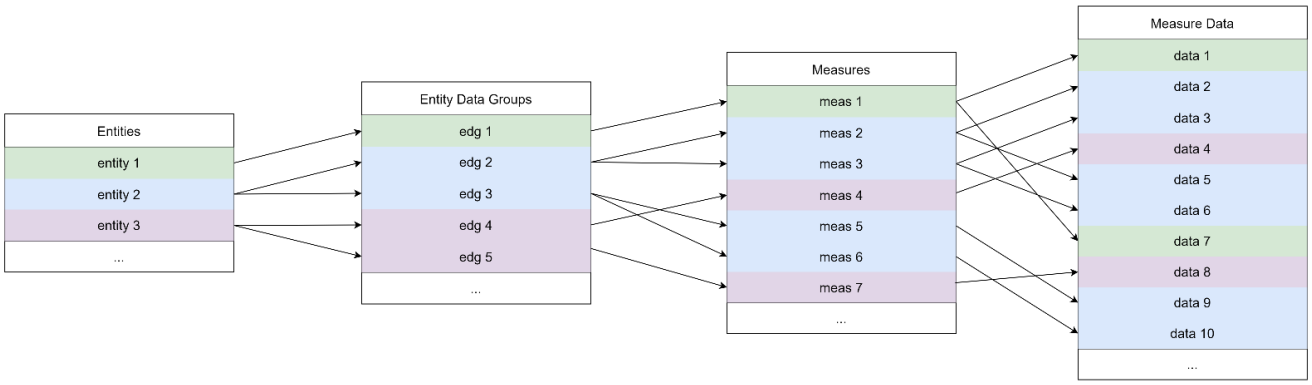


**Figure 13 Relationship between the hc_measure_data table and the hc_entities table**

N.B: When a record in the "hc_entities" table is deleted, all associated entity data groups, measurements, and related data will be permanently deleted as well.



This project has received funding from the European Union's H2020 research and innovation programme under Grant Agreement No 101033683

#### 5.2.1.3 Diagnostics

***hc_system_diagno***: records diagnostic data relating to the operation of the hub core module.

| *Field* | *Type and Constraints* | *Description* |
|---|---|---|
| dt | DATETIME, PRIMARY KEY, DEFAULT CURRENT_TIMESTAMP | Date/time of the diagnostic records; value automatically assigned by the DB during record creation |
| start_dt | DATETIME, NUT NULL | Indicates from which instant the counting of the diagnostic statistics is in progress and determines the period to which all subsequent counts refer |
| db_size | INT, NOT NULL | Database memory size |
| meas_recs | INT, NOT NULL | Number of records in hc_measure_data. This value is only an estimate with an accuracy of less than 1%, calculated from empirical measurements |
| ents | INT, NOT NULL | Number of registered entities |
| en_ents | INT, NOT NULL | Number of enabled entities |
| retr_ents | INT, NOT NULL | Number of entities for which BRiG is awaiting a response following a timed-out operation (no response received) |
| unreach_ents | INT, NOT NULL | Number of entities that have failed all communication attempts and are therefore considered unreachable |
| unreg_ents | INT, NOT NULL | Number of detected unregistered entities |
| sys_errors | INT, NOT NULL | Number of system errors |

***hc_entities_diagno***: records diagnostic data relating to each registered entity.

| *Field* | *Type and Constraints* | *Description* |
|---|---|---|
| dt | DATETIME, PRIMARY KEY, DEFAULT CURRENT_TIMESTAMP | Date/time of the diagnostic data record; value automatically assigned by the DB during record creation |
| start_dt | DATETIME, NOT NULL | Indicates from which instant the counting of the diagnostic statistics is in progress and determines the period to which all the subsequent counts refer |



This project has received funding from the European Union's H2020 research and innovation programme under Grant Agreement No 101033683

| | | |
|---|---|---|
| entity_id | INT, PRIMARY KEY, FOREIGN KEY(hc_entities.id) | Entity id from hc_entities table |
| unsol_msg | INT, NOT NULL | Number of unsolicited messages received from the entity |
| rdns | INT, DEFAULT NULL | Number of measure records created in hc_measure_data table from the entity |
| reqs | INT, NOT NULL | Number of requests sent to the entiy |
| anss | INT, NOT NULL | Number of responses received from an entity |
| status | VARCHAR(3), NOT NULL | Encoding that describes the current functional state of the entity |
| errrors | INT, NOT NULL | The number of messages transmitted by the entity with a non-null error code |

The following constraint has been added to the table:

- "foreign key" linking the "entity_id" column to the "id" column of hc_entities. This associates the entity diagnostic daya with the entity it refers to. If the value in the parent column is modified, the "entity_id" value in this table will also be updated (on update cascade). If the parent record is deleted, the corresponding records in this table will also be deleted (on delete cascade).

### 5.2.2 iGateway Tables

The iGateway follows a similar behavior to the Hub Core when it comes to managing the connections to external components, with the difference that, rather than interfacing direct data sources, it has to interface with other components of the COLLECTiEF architecture in order to behave properly. The iGateway tables are tasked with retaining information from the communications with the Cluster Node, as well as handle the memory for the central communication in terms of diagnostics and error interfacing.

### 5.2.3 Algorithms Tables

#### 5.2.3.1 DSM tables

Algorithm tables follow a semi structured format, with a number of columns being informational data, while the *cfg* files act as configuration JSONs, allowing for flexibility in case of varied behavior.

**dsm_assets**: holds the information for the DSM algorithm, determining which zones contain which sensors and which actuators can control the temperature setpoints and such.



This project has received funding from the European Union's H2020 research and innovation programme under Grant Agreement No 101033683

| *Field* | *Type* | *Description* |
|---|---|---|
| zone_id | VARCHAR(16), PRIMARY KEY | The zone_id parameter is textual, and not definitive. Zone separation is not necessarily physical |
| zone_type | VARCHAR(32) | Describes textually the type of zone, not mandatory |
| zone_category | TINYINT(), NOT NULL | The possible values for the categories are used in the algorithm, they can be either 1, 2 or 3 |
| cfg | JSON, NOT NULL | The JSON data contains the sensors and actuators name, as well as the default values to be put in. Any further setting to be applied such as minimum and maximum setpoints that are allowed:<br><br><<actuator>>: {<br>"default": <<value>>,<br>"settings": {<br>"values": {"min": <<min>>, "max": <<max>>},<br>"features": {<br>…<br>}<br>} |



This project has received funding from the European Union's H2020 research and innovation programme under Grant Agreement No 101033683

**dsm_signal_library**: holds the signal library, according to the setup of NTNU's data. The signal is as follows:

| *Field* | *Type* | *Description* |
|---|---|---|
| zone_id | VARCHAR(16), FOREIGN KEY(dsm_assets.id) | The zone id of the area, it is linked to the assets library in terms of zone id. |
| season | TINYINT(), PRIMARY KEY | Season of the year, values from 1 to 4 |
| signal | TINYINT(), PRIMARY KEY | Flexibility signal value, from 0 to 5 |
| hour | TINYINT(), PRIMARY KEY | Time of the day, in intervals of three hours, from 0 to 8 |
| cfg | JSON, NOT NULL | The configuration follows a pattern with a list called "reward" and another list of the same length called "action" that contains the actions partaken by the system in order to work. |

## 5.3 DB deployment and communication

The initial deployment will start with a simple docker compose that makes use of an entry point script, using a model file for the database schema. This database model will be common for most components and used thanks to docker volumes, so that each component can call upon the same database schema easily.



This project has received funding from the European Union's H2020 research and innovation programme under Grant Agreement No 101033683

# 6 Collective Intelligence BRiG Edge Node Algorithms for demand side management and building thermal network optimization

## 6.1 CI-DSM

DSM refers to the set of means to change the pattern and/or magnitude of energy use, which usually appears as a set of actions and strategies to reduce, increase, or reschedule the demand [8]. We introduced a DSM approach based on Collective intelligence, calling it CI-DSM [9]. CI is a form of universally distributed intelligence, working based on collaborative problem solving and decision making [10]. By analyzing the CI-DSM for extreme climate conditions in Stockholm, we showed that CI-DSM can enhance the flexibility of an energy system and consequently make it more resilient against environmental variations or external shocks [9].

Since building and energy systems are multi-variant systems, reaching an optimum control strategy can become very challenging, especially when the aim is reaching light algorithms that do not need high computational power. In this regard, RL-based methods have shown substantial potential in resolving increasing complexities within the energy domain, considering both supply and demand [11][12]. In CIRL-DSM, RL is used to make decision making at the building level (which are interpreted as agents in the system) while the only information they get from the grid (or their environment) is through flexibility signals (representing the need for flexibility from the energy provider, also represented by the price signal). In CIRL-DSM, the flexibility signal is between 0 and 5, which 0 means there is no need for flexibility and 5 asks for the maximum possible flexibility in the grid.

In relation to the basics of RL, in CIRL-DSM the flexibility signal can be interpreted as the state, informing the agents about the status of the environment. For example, the flexibility signal compares the energy demand at time *t* to the reference energy demand at time *t*. The reference energy demand is the demand during typical weather conditions or TDY. Signal 0 means that the energy demand is less than or equal to the TDY demand while values 1-5 indicate a higher demand; the larger the signal, the higher the demand in comparison to the TDY demand. The self-knowledge of the agent is generated by calculating rewards per time step or values, which in the current version are calculated considering energy demand and indoor comfort. If both the energy demand and indoor discomfort at time t are smaller than the corresponding value for the extreme reference case, the value is equal to 1 and the action will be considered as a suitable adaptation action. The selected actions are added in the library of actions.

The final and optimized library will become the policy of the agent. In other words, the agent's policy is a set of optimal actions which have been selected through an iterative process; actions with the value of 1 that have shown the best performance, energy and comfort wise. The control strategies inside agents (buildings) are interpreted as the actions of the agents, which are also called adaptation actions/measures. To not stick all the time to the policy and try different actions, we defined a randomness factor in the algorithm, which allows the algorithm to pick a random action even if the policy is set. This opens doors to update the policy if, by any chance, a better action is being experienced by the agent (the weakest action will be replaced by the better action in the policy).



This project has received funding from the European Union's H2020 research and innovation programme under Grant Agreement No 101033683

## 6.2 Building Thermal Optimization Algorithms

These algorithms refer mainly to the occupant-centric algorithm that is developed for T2.2 that aims to modulate indoor environmental conditions to improve occupants' comfort in actual operational conditions by increasing their satisfaction and productivity. The concept of the algorithm is presented in more detail in D2.3 - *Fine-tuned and feasible control strategies that address user comfort, cost & energy efficiency, climate change mitigation & adaptation (first version)* and D2.5 - *Verified and working control strategies that maximise the occupant comfort and integration of renewable energy generation, working for current and future climate*. Here are the main functionalities and input/output information of the algorithm:

- ***Main functionalities***: The algorithm aims to improve occupants' comfort in actual operational conditions by increasing their satisfaction and productivity. For this reason, an occupant-centric control algorithm for enabling thermal flexibility by modulating indoor thermal conditions is developed. The backbone of the algorithm is the state-of-the-art thermal comfort models, which are presented analytically in D2.3. The algorithm follows a rule-based approach to activate and deactivate corrections to the online temperature set point to provide: (i) comfort to occupants, (ii) health improvement for occupants, and (iii) energy flexibility to the grid.
- ***Input and output signals***: Through the COLLECTiEF Human-Building interface, the user will have the ability to schedule, within 24 hours per day and 7 days a week, at both building and zone-level, the mode of operation of the COLLECTIEF system; that is "MANUAL", "COMFORT", "HEALTH", and "ENER. FLEX.". "MANUAL" corresponds to the mode in which the users have the option to choose the operation of building equipment (i.e., room air temperature set point, ventilation rate, $CO_2$ concentration level, etc.) according to their preferences. On the other hand, for the modes "COMFORT", "HEALTH", and "ENER. FLEX." the occupant-centric control algorithm will activate to optimize occupant comfort, health and energy flexibility to the grid without creating indoor discomfort conditions.



This project has received funding from the European Union's H2020 research and innovation programme under Grant Agreement No 101033683

# 7 Edge Node interface

## 7.1 Human Building Local Interface

In the Human Building Interface, the development framework used is React.js which is a JavaScript library used to build reusable and dynamic user interfaces. Human building interface consists of many different functions including a login function, which facilitates the building owners/managers to enter the human building interface with their own login identifications.

The building owners/managers can then access the aggregated Dashboard. This dashboard consists of different data visualization methods such as graphs, charts etc.

These graphs and charts represent environmental conditions and analysis for different environmental parameters. The Dashboard is also able to represent the geographical location of the specific building in a map as location reference (Figure 14). Additionally, environmental condition data of the relevant building area such as temperature, relative humidity, wind speed etc., will also be represented in the interface. Information on topics such as energy savings, energy flexibility, climate resilience and combined wellbeing, corresponding to a certain building will be available for the building owners/manager's reference. Information such as, thermal, and visual comfort, temperature levels, air quality data, battery levels of sensors and data transmission efficiency of sensors are other parameters that will be available for the building owners/managers to refer to in the Dashboard.

This aggregated Dashboard will consist of information for multiple apartments of the same building.


**Figure 14 Geographical location details of the building**

The building owner/manager will be able to create locations related to different units or subunits of the building. The building owner/manager will also be able to assign a suitable sensor corresponding to locations created by them.

The creation of user profiles for different users is another available function for the building owner/manager, where they can enable occupants to access their user interfaces.

The building owner/manager will also conduct the SRI calculation through the interface (Figure 15). This feature is available in the Dashboard, in this way the building owner/manager can easily get the SRI calculation results.



This project has received funding from the European Union's H2020 research and innovation programme under Grant Agreement No 101033683

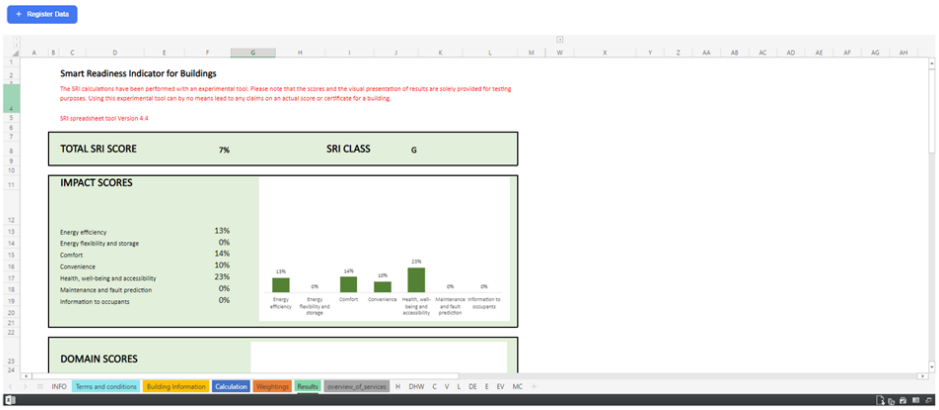


**Figure 15 SRI calculation**

The Aggregated scheduling modules allow building owner/manager to create schedules throughout the day or multiple days of a week. When creating the schedules, different environmental parameters could be selected.

This lets the building owner/manager access to set different modes such as Manual, Resilient, Comfort and Eco modes to effectively manage the consumption of energy. The aggregated notification functions, available in the interface, let the building owner/manager monitor the notification status of multiple apartments in the building.

They can monitor the default notifications as well as create customized notifications as needed. The notifications will be triggered when a defined condition is above/below or does not conform with a certain defined level. The Aggregated feedback function is another option that will be available in the interface, letting the building owner/manager to get feedback from occupants of apartments in the building.

## 7.2 Application Server

For the application backend, Node.JS environment has been used as the backend language framework. Since Node.js follows an event-driven architecture, this supports the creation of applications that efficiently respond to various events. Node.js is best at creating real-time applications such as these interfaces that represent dashboards with real-time data. These are some of the reasons considered when selecting Node.JS environment as the backend language framework.

The whole application will be converted to a docker application that can be run on Raspberry PI board. The docker will provide many benefits, including portability, scalability and easy deployment. Docker containers provides standardized runtime environment, which increases the efficiency and the easy deployment of applications. The Raspberry Pi CPU platform is a versatile choice with a low-cost and energy-efficient computer to host server applications.



This project has received funding from the European Union's H2020 research and innovation programme under Grant Agreement No 101033683

# 8 Conclusions and future works

In this final chapter, we summarize the key findings and outcomes of the "Report on COLLECTiEF Edge Node" (D3.2) and draw conclusions regarding the development and integration of the Edge Node within the COLLECTiEF project.

The primary objective of this deliverable was to describe the first version of the Edge Node and its integration with field-level devices, then establishing a crucial step toward the successful implementation of the project solution across different pilot sites. By building upon the project's goals, previous work conducted in WP2, and the architecture defined in D3.1, we have provided a comprehensive account of the development process of the Edge Node.

Throughout the document, we have detailed the overall approach and methodologies employed to develop the connectors with field devices, manage the local database, coordinate energy flexibility management algorithms and thermal comfort optimization, and facilitate communication with upper layers. These efforts have culminated in the creation of the initial version of the Edge Node, referred to as the BRiG device.

In conclusion, the "Report on COLLECTiEF Edge Node" (D3.2) has successfully outlined the development and integration of the Edge Node within the COLLECTiEF project. It has provided valuable insights into the technical aspects, methodologies, and achievements of this important milestone, that is the development of the first version of the Edge Node.

The first version of the Edge Node, the BRiG device, marks a significant step forward in realizing the project's objectives.

Moving forward, we will continue to refine and optimize the Edge Node to facilitate the effective implementation of the COLLECTiEF system across the pilot sites within the deadlines set by the project.



This project has received funding from the European Union's H2020 research and innovation programme under Grant Agreement No 101033683

# References


[1]. COLLECTiEF D3.1 “Cluster-Edge architectural scheme”

[2]. ecobee. (s.d.). Ecobee API documentation from Ecobee API: https://www.ecobee.com/home/developer/api/documentation/v1/auth/auth-intro.shtml

[3]. Sensibo. (s.d.). Sensibo API documentation. from Sensibo API: https://sensibo.github.io/#post-/pods/-device_id-/acStates

[4]. Selenium, https://www.selenium.dev/

[5]. Object-relational mapping, https://it.wikipedia.org/wiki/Object-relational_mapping

[6]. Ecobee API source https://www.ecobee.com/home/developer/api/examples/ex3.shtml

[7]. Pham, V.-N.; Lee, G.-W.; Nguyen, V.; Huh, E.-N. Efficient Solution for Large-Scale IoT Applications with Proactive Edge-Cloud Publish/Subscribe Brokers Clustering. Sensors 2021, 21, 8232. https://doi.org/10.3390/s21248232

[8]. Lund PD, Lindgren J, Mikkola J, Salpakari J. Review of energy system flexibility measures to enable high levels of variable renewable electricity. Renew Sustain Energy Rev 2015;45:785–807. https://doi.org/10.1016/j.rser.2015.01.057.

[9]. Nik VM, Moazami A. Using collective intelligence to enhance demand flexibility and climate resilience in urban areas. Appl Energy 2021;281:116106. https://doi.org/10.1016/j.apenergy.2020.116106.

[10]. Suran S, Pattanaik V, Draheim D. Frameworks for Collective Intelligence: A Systematic Literature Review. ACM Comput Surv 2020;53:14:1-14:36. https://doi.org/10.1145/3368986.

[11]. Nweye K, Liu B, Stone P, Nagy Z. Real-world challenges for multi-agent reinforcement learning in grid-interactive buildings. ArXiv211206127 Cs Eess 2022.

[12]. Perera ATD, Kamalaruban P. Applications of reinforcement learning in energy systems. Renew Sustain Energy Rev 2021;137:110618. https://doi.org/10.1016/j.rser.2020.110618.




This project has received funding from the European Union's H2020 research and innovation programme under Grant Agreement No 101033683