GAIA - Green Awareness In Action

D2.2 Final Infrastructure Software

Document Ref.	
Document Type	Deliverable
Work package	WP2
Author(s)	Federica Paganelli, Giovanni Cuffaro (CNIT), Nelly Leligou, Katerina Papadopoulou (Synelixis), Mariano Leva, Matteo Zaccagnino, Massimo Mecella (OVER), Orestis Akrivopoulos, Nikos Kanakis (SPARK), Georgios Mylonas, Dimitrios Amaxilatis (CTI)
Contributing Partners	CNIT, SYN, OVER, SPARK, CTI
Dissemination Level	Public
Status	Final version
Version	V1.0
Contractual Due Date	M24 (January 31, 2018)
Actual Delivery Date	February 20, 2018

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement 696029.

Disclaimer

Partner No.	Name	Short Name	Country
1	Computer Technology Institute and Press "Diophantus"	CTI	Greece
2	Söderhamns Kommun	SK	Sweden
3	Eurodocs AB	EDOC	Sweden
4	National Interuniversity Consortium for Telecommunications	CNIT	Italy
5	Synelixis Solutions Ltd	SYN	Greece
6	OVER	OVER	Italy
7	Ellinogermaniki Agogi	EA	Greece
8	Spark Works ITC Ltd.	SPARK	United Kingdom
9	Ovos Media Consulting Gmbh	OVOS	Austria

Document Revision History

Date	Issue	Author/editor/contributor	Summary
22/11/2017	0.1	Federica Paganelli, Giovanni Cuffaro	Document structure
08/01/2018	0.2	Matteo Zaccagnino, Orestis Akrivopoulos, Nikos Kanakis, Federica Paganelli, Giovanni Cuffaro	Added content
20/01/2018	0.3	Matteo Zaccagnino , Orestis Akrivopoulos, Nikos Kanakis, Federica Paganelli, Giovanni Cuffaro	Section 11 added, performance metrics added
10/02/2018	0.4	Federica Paganelli	Introduction and Conclusions sections
14/01/2018	0.5	Massimo Mecella, Federica Paganelli, Nikos Kanakis, Nelly Leligou	Conclusions sections, minor revisions
19/02/2018	1.0	Georgios Mylonas	Final version

Abbreviations

Abbreviation	Expression
AA	Authentication & Authorization
AMQP	Advanced Message Queuing Protocol
API	Application Programming Interface
AS	Authorization Server
CD	Continuous Deployment
CI	Continuous Integration
CRUD	Create Read Update Delete
DB	Database
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
REST	Representational State Transfer
URI	Uniform Resource Identifier
WP	Work Package

Executive summary

This deliverable describes final release of the infrastructure software called *GAIA Service Platform*, which represents essentially the back-end of the GAIA ecosystem. The main goal of the GAIA Service Platform consists in acquiring data from the heterogeneous sensor infrastructures deployed in the schools' premises in Greece, Italy and Sweden, store and process these data to derive meaningful information for energy consumption awareness and reduction (i.e., analytics and energy-saving recommendations), and, finally, provide GAIA end-user applications with a unified access mechanism to such data.

This deliverable describes the final release of the software by describing major additional features and changes with respect to the first release, documented in [GAIA2.1]. Indeed, [GAIA2.1] provides an overview of the logical architecture of the GAIA Service Platform, describing the features offered by each functional blocks, namely: Authentication and Authorization Infrastructure, Acquisition, Storage, Analytics, Recommendations, Building knowledge base. This document reports on updates with respect to the release documented in [GAIA2.1] and provides further insights on deployment details and metrics related to performance and operation of the software.

Table of Contents

1 Introduction

1.1 Scope of deliverable

This deliverable describes the work conducted by the GAIA Consortium in WP2, by reporting the results of Tasks 2.1, 2.2 and 2.3. Emphasis is given on the work conducted since the release of deliverable D2.1 [GAIA2.1]. Therefore, D2.2 complements [GAIA2.1] in documenting the final release of the infrastructure software, while minimizing overlapping content.

1.2 Relation to other deliverables

The goal of WP2 is to support the GAIA mission by providing an intermediate layer (middleware) between sensors' infrastructures and GAIA end-users applications. This middleware is called *GAIA Service Platform*. The main goals of the GAIA Service Platform consist in: i) acquiring data from the heterogeneous sensor infrastructures deployed in the schools' premises in Greece, Italy and Sweden, ii) storing and processing these data to derive meaningful information for energy consumption awareness and reduction (i.e. analytics and energy-saving recommendations), and iii) providing GAIA end-user applications with a unified access mechanism to such data.
This deliverable describes the final release of the software by describing major additional features and changes with respect to the first release, documented in [GAIA2.1]. Indeed, [GAIA2.1] provides an overview of the logical architecture of the GAIA Service Platform, describing the features offered by each functional block, namely: Authentication and Authorization Infrastructure, Acquisition, Storage, Analytics, Recommendations, Building knowledge base.
This document reports on updates with respect to the release documented in [GAIA2.1] and provides further insights on deployment details and metrics related to performance and operation of the software.

1.3 Overall structure of this document

The rest of this document is organized in the following chapters:

- **Section 2 GAIA Service Platform** shows the overall logical architecture of the platform, distinguishing main functional blocks, and introduces novel features added in the second release.
- **Section 3** describes the updated Authentication and Authorization Infrastructure.
- **Sections 4** describes novel components of the Data Acquisition system and reports on performance metrics.
- **Section 5** reports on the performance metrics related to the Data Storage system.
- **Section 6** describes the centralized logging system that has been setup for the effective monitoring of the platform operation.
- **Section 7** describes minor updates on the Building Knowledge Base component.
- **Section 8** provides details on the Analytics module implementation.
- **Section 9** describes main updates of the Recommendation Engine.
- **Section 10** shows additional GAIA processes, leveraging WP2 and WP3 components

cooperation, in addition to processes already described in D2.1.

- **Section 11** provides details on the environment where the GAIA Service Platform has been deployed and discusses related choices.
- **Section 12** concludes the document discussing achieved results and relation to GAIA KPIs [GAIA 1.1].

2 GAIA Service Platform architecture

In this section, we show the overall architecture of the GAIA Service Platform. We refer to D2.1 for the description of the main components of the logical architecture, listed hereafter:

- Authentication and Authorization Infrastructure
- Acquisition
- Storage
- Analytics
- Recommendations
- Building knowledge base

We have updated the figure to take into account updates on the platform, which are fully documented in the following sections. Although some significant changes and updates have been done on the service platform software implementation, only a few are reported in the schema, since matching with the level of abstraction of components in Figure 1.

In the Data Acquisition block, the Resource Registry component has been added, while the Recommendation engine has an additional notification channel (email). The structure of the Analytics block has been revised to better reflect the organization in modules (i.e. Anomaly Detection, Clustering, Statistics).

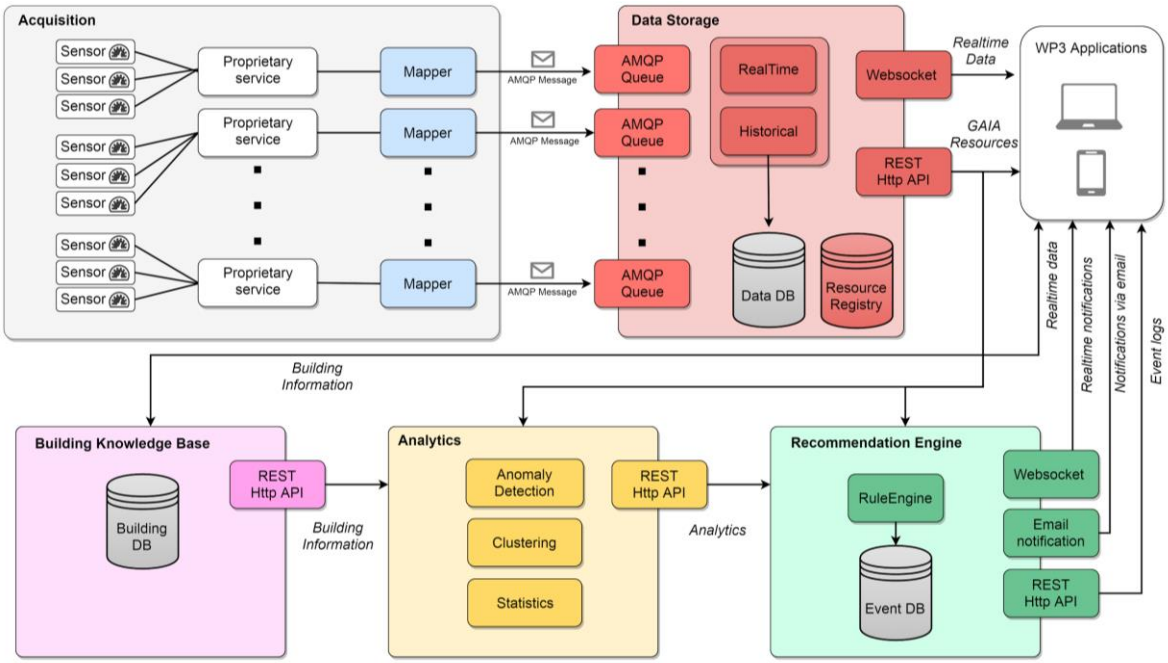

Figure 1 GAIA Service Platform logical architecture

3 Authentication & Authorization Infrastructure

3.1 Requirements

Code	Description	Priority
Aa.6	New users should be able to register in the platform	HIGH
Aa.7	The platform should provide a new user invitation mechanism	HIGH
Aa.8	A platform user should be able to change its password	HIGH
Aa.9	The platform should provide "forgot password" functionality	HIGH

3.2 Design

Apart from the functionalities of the initial prototype of the AA service, the following capabilities have been added.

- In order to register with a new system account a user has to navigate to the registration page of the AA application, which contains a registration form that must be filled with some necessary personal information for identification and security purposes. To ensure proper functionality and avoid malicious users trying to access the system, all new registrations are not processed instantly, but need to be approved by users with appropriately authorized account.

- Apart from the common process of new users registration described above, the platform provides a user invitation mechanism. Specific user roles are able to send an invitation for a new user registration in the platform. An email is delivered to the new candidate user email address with instructions on how to accept and complete its registration to the GAIA platform.

- Any registered user in the platform is able to administer its account details i.e. changing its personal details and preferences or changing its password. Furthermore, in case that a user has forgotten its password, it is possible to request a password change through a process initialized from the login screen. In more details, a user is able to request a password reset after providing its registered email address in the platform. The system afterwards validates the provided email address and delivers an email to the users' address with instructions on how to complete the password change process.

4 Data Acquisition

The Data Acquisition block has been enhanced with a novel component, called Resource Registry, described hereafter.

4.1 Requirements

Code	Description	Priority
Rr.1	The platform should be able to monitor the health of each sensoring resource	HIGH
Rr.2	The platform should be able to detect unhealthy sensoring resources	HIGH
Rr.3	The platform should be able to notify appropriate users for unhealthy sensoring resources	HIGH

4.2 Design

A crucial point on the operation in a system like the GAIA platform where sensing data are collected through several, heterogeneous sensing sources is the ability of the system to monitor the "health" of each sensor. Sensor "health" is described as the operational status of a sensor regarding the ability to collect sensing data and pipe them in the platform.

To fulfil this need the Resource Registry service is introduced in the platform. Every sensor source of the platform is registered in the Resource Registry service and its operation is continuously monitored based on the latest data received.

The metadata of the registered sensors on the Resource Registry service are stored in an hierarchical manner according to the site each resource belongs to allowing for bulk operations on the Resource Registry.

Upon reception of the first measurement from a sensing source, the platform registers this sensing source as a resource in the Resource Registry with a specific time to live (TTL) value according to the type of the sensing data. The TTL depends on the sensor type since the data acquisition interval is defined with respect to the data type each sensor collects. When a new measurement arrives, the TTL entry for the specified sensor is updated in the Resource Registry service marking the sensor as healthy. If a sensor stops or fails to send new measurements for any reason, this will cause the TTL lease of the sensor to expire and the Resource Registry will mark the sensor as unhealthy in its registry. Afterwards, according to the GAIA site, the specified sensor belongs to, the site local managers and the global managers will be notified via email for the sensor unhealthy status.

4.3 Implementation

The Resource Registry service is a distributed reliable key-value store deployed in the Docker Swarm GAIA cluster. The implementation of the service is based on etcd [ETCD] version 2. Etcd is a distributed key value store that provides a reliable way to store data across a cluster of machines. Based on this technology, the Resource Registry provides high availability with data replication

alongside the whole cluster since etcd is deployed in cluster mode in the nodes of the GAIA Docker Swarm cluster. Finally, etcd provides official Docker images of the system therefore the deployment in the GAIA Docker Swarm cluster is straightforward.

4.4 Performance

Some performance metrics on the Data Acquisition service are summarized in Table 1. The metrics presented on Table 1 are collected at the end of the first month of year 2018.

Table 1 Data acquisition performance metrics

Number of API Mappers	**7**
Number of Resources	790
Average messages per sec	35
Message size (Bytes)	110
Average input data per sec (Bytes)	3850 (35 * 110)

5 Data Storage

Regarding the performance of the GAIA Data Storage module, some data metrics are summarized in Table 2 and Table 3 as collected by the end of the first month of year 2018.

A Resource in the context of the GAIA platform denotes any source producing data in the platform i.e. can be a physical sensor or also can represent aggregates/summaries produced by the analytics engine of the Data storage module.

A Data Tuple denotes an actual measurement in the form of a triplet with the unique Resource URI, the timestamp of the measurement and the actual value of the measurement and is stored in JSON format in a NoSQL database.

Finally, the average processing rate metric refers to the rate at which the processing engine of the Data Storage module generates aggregate values (i.e. averages or summaries) from the incoming measurements as depicted from the statistics metrics of the message broker component.

Table 2 Data storage performance

Number of Data Tuples (measurements)	253.231.740
Total size of processed data stored (GB)	115
Average processing rate (Bytes/sec)	103.915

Furthermore, Table 3 provides some metrics on the average response time of the Data Storage API for retrieving processed data for several combinations of time period and results granularity.

Table 3 Data storage API performance

Query time for summary query	247 ms
Query time 1 hour (5 min granularity)	265 ms
Query time 24 hours (5 min granularity)	864 ms
Query time 24 hours (1 hour granularity)	377 ms
Query time 1 month (1 hour granularity)	1023 ms
Query time 1 month (1 day granularity)	293 ms
Query time 1 year (1 month granularity)	234 ms
Query time 1 year (1 day granularity)	824 ms

6 Centralized Logging

6.1 Requirements

Code	Description	Priority
CL.1	The platform should provide centralized logging	HIGH
CL.2	Logs should be parsed and send to a central database in near real-time	HIGH
CL.3	Database capacity to handle near real-time data querying	HIGH
CL.4	Visual representation of data through filtered tables and dashboards	HIGH
CL.5	Predefined Reports for visualising logs per application type	HIGH

6.2 Details

In a microservices platform such GAIA, centralized logging can be very useful when struggling to identify problems from different microservices composing the whole system, as it allows searching through all of the services logs in a single place. It is also useful because it allows identifying issues that span multiple servers by correlating their logs during a specific time frame.

In GAIA we need a combination of decentralized log collectors that are sending information to a centralized parsing service and data storage. ELK stack from Elastic [ELK] fulfils the above requirements and has been utilized in GAIA to provide log events aggregation from the platform microservices into a centralized database for analysis and visualization.

The ELK stack from Elastic consists of:

- **Elasticsearch**
  Elasticsearch is a distributed and scalable full-text search database, enabling systems to store and search large volumes of log events.
- **Logstash**
  Logstash can collect log events from multiple types of sources, transform them to a single format and send them to a number of destinations.
- **Kibana**
  Kibana provides the visualization and analysis of the log events stored in Elasticsearch.

In the GAIA cluster, along with the GAIA services, the dockerized ELK stack has been deployed which collects, analyzes and provides the logs from the Data Storage, the Authentication & Authorization, the Resource Registry and the Message Broker modules.

7 Building Knowledge Base

There has not been any major change to the Building Knowledge Base module since its last release. In the following sections we describe the minor enhancements that have been introduced.

7.1 Requirements

Code	Description	Priority
Kb.1	It must be able to store data about the structure of the building	High
Kb.2	It must be able to store data about the network measurement deployed in the building.	High
Kb.3	It must be able to store specific information on each part of the building (i.e., building itself, floor and room)	Medium
Kb.4	It may be able to store information about the employed heating and cooling systems.	Low
Kb.5	It may be able to store information about the user transportation choices.	Low
Kb.6	It must provide CRUD operation accessible by REST web services	High
Kb.7	It may store information about the person in charge of managing a building.	Low
Kb.8	It may store information about relation between persons and area	Low

7.2 Design

The Entity-Relation model of the Building Knowledge Base is recalled in Figure 2.
The schema is almost completely unchanged, apart from the addition of a new attribute to the SiteInfo entity in order to keep track of the average weekly working hours of a Site.
It is also worth mentioning that the Event entity is now used to store the anomalies detected by the Anomaly Detection Module, allowing reviewing and/or deleting them, as we will discuss in more details in the next chapter.

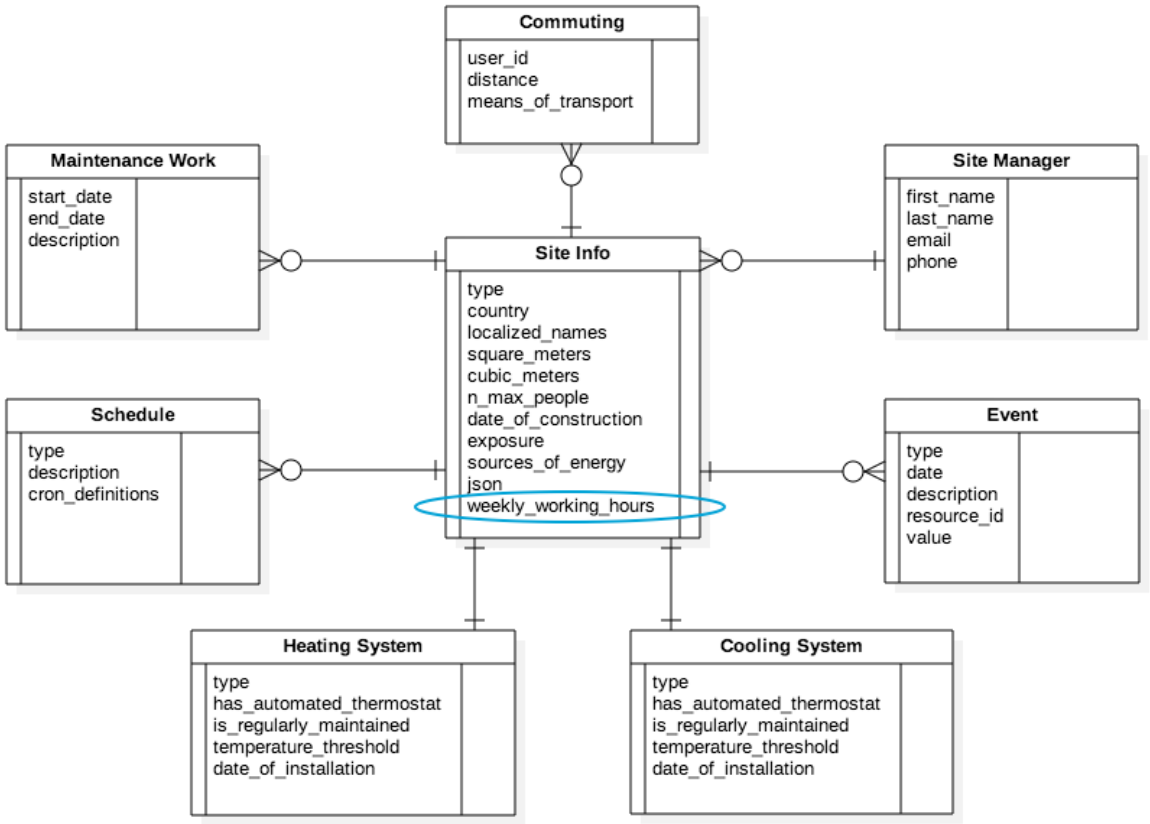

Figure 2 Building Knowledge Base Entity-Relation diagram

8 Analytics module

The Analytics module is made up of three different sub-modules: the Statistics module, the Clustering module and the Anomaly Detection module. While the former has been left unchanged from the last release, the Clustering and the Anomaly Detection have been substantially improved, as described hereafter.

8.1 Requirements

Code	Description	Priority
An.1	It must be able to determine a typical pattern of consumptions per each day of the week	Medium
An.2	It must be able to recognize anomalies if values aggregated from the GAIA platform differ by a certain amount from those ones of the typical pattern	Medium
An.3	It should provide clustering functionalities in order to aggregate typical patterns of multiple resources	Medium
An.4	It may forecast consumption of a given day on the base of typical patterns computed	Low
An.5	It must provide general statistics such as average consumption per area or per resource	High
An.6	It should provide statistics on average at least with the granularity data of “MONTH, DAY, HOUR, QUARTER HOUR”	High
An.7	It must provide REST API implementing common CRUD operation to allow the interface with WP3 applications	High
An.8	It must provide a way to annotate, in a textual form, discovered anomalies	High

8.2 Clustering Module

In the previous release of the Analytics module, buildings were clustered according to their similarity with regards to architectural structure, energy consumptions and climate. These aspects were translated into features that led to a single representation of a building in an n-dimensional space. Thus, two buildings would have been considered similar by the algorithm only if they share a set of similar features across all the above aspects.

Although interesting, we found out that this approach was of little practical use, since it would not provide much useful information in all those cases where two buildings were neither 100% nor 0% similar as it was difficult to tell which the shared features were and which ones were not. Therefore, it was not easy to perform a comparative analysis in all those instances where it would have matter

the most.

To overcome this limitation, the Clustering module has been redesigned and improved. Two separated clustering instances are now available:

- **ENERGY CLUSTERING**: this instance evaluates buildings according solely to their energy consumption (seasonal and yearly averages).
- **STRUCTURE CLUSTERING**: this instance evaluates buildings according to their architectural features and schedules.

For each building, two vector representations (one per clustering instance) are produced by mapping the relevant features to vector components. Vector components are then normalized by using Min-Max Normalization in order to reduce feature bias. Such vectors are then clustered using the K-Means algorithm and the results are stored in the Building Knowledge Base. Currently, the clustering task is scheduled at the start of each month to keep up with changes in the seasonal energy consumption. A clustering task that involved the 19 buildings currently in the system requires about five minutes to be completed on the current architecture. The number of clusters is fixed at 4 and can be changed manually.

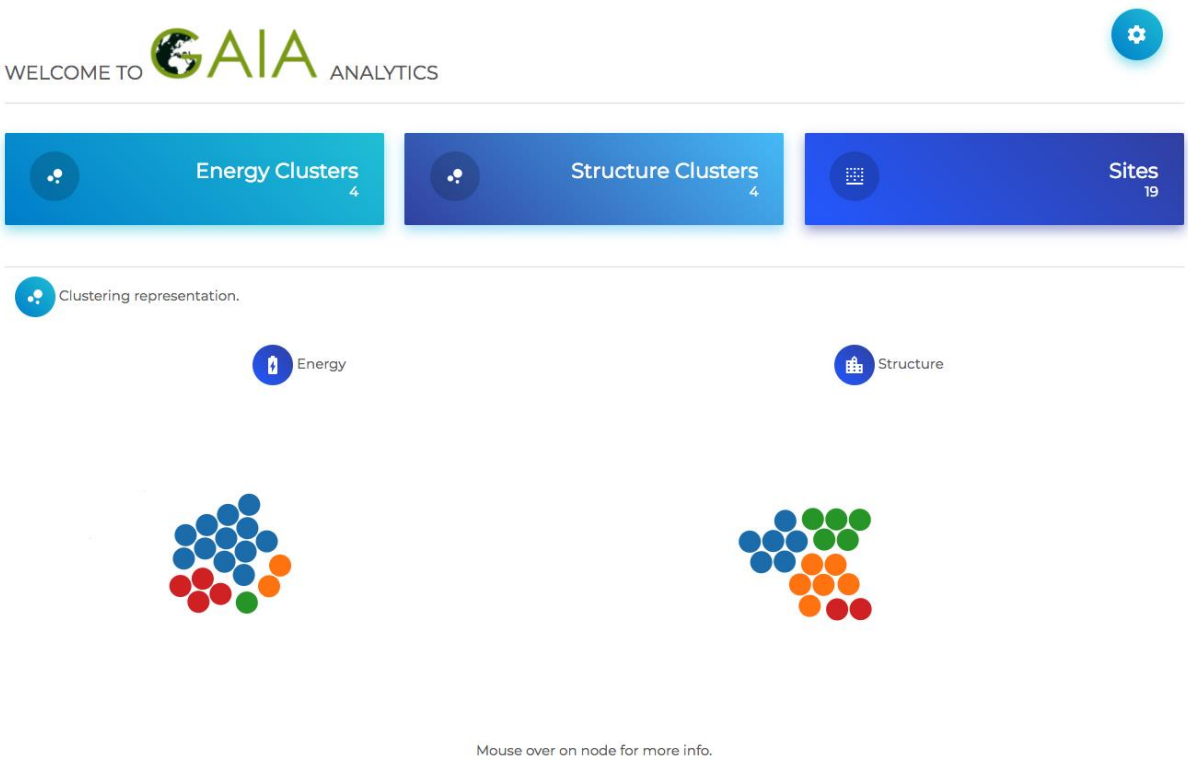

Figure 3 Analytics dashboard - clustering

To be able to inspect the results of the clustering and anomaly detection modules we built a simple UI available at https://www.overtechnologies.com/gaia_dev/. Figure 3 shows the results of the new clustering module. Each circle is a building and each colour identifies a different cluster.

It is important to notice that the energy and the structure clustering do not correspond. That is because it is possible that buildings have similar energy consumption even though they do not share the same structure or schedule and vice versa. Those cases are exactly the ones in which there could be room for improvements and the new clustering module helps finding them.

Let us assume we are interested in understanding if a given building has reasonable energy consumption with regards to how other buildings in the platform are performing. The first step could be finding what are the buildings that share the same structural features with the one we are analysing, i.e. identifying which structure cluster our building is in. We could then look at all the buildings in that cluster and see if they are also inside the same energy cluster. In that case, we can conclude that our building is actually performing as well as other physically similar buildings, otherwise we have a chance to understand why the building is consuming more (or less) energy compared to the others. To further simplify this kind of analysis we also introduced two different similarity scores, one for the energy and one for the structure, computed used the cosine similarity and expressed as a percentage value.

For instance, at the moment of writing, the Gramsci-Keynes School in Prato and the Experimental Junior High School in Patras have a structure similarity score of 10% and a 45% energy similarity score. The school in Prato is, in fact, almost ten times bigger than the one in Patras and can host more than three times the number of students. However, these proportions does not hold when considering their energy consumption. Therefore, it would be interesting to understand whether such a disparity is justified by some other factors or if the Prato's school is indeed more efficient than the one in Patras.

Both the clustering results and similarity scores are available through REST endpoints (the documentation is available at https://analytics.gaia-project.eu/gaia-analytics/swagger-ui.html).

8.3 Anomaly Detection Module

The Anomaly Detection module has been reworked in order to overcome some drawbacks of the previously employed algorithm. The vector quantization technique that was used was in fact lacking three key features:

- **exogenous factors**: consumption patterns were estimated without directly taking into account exogenous factors such as temperature, luminosity or occupancy;
- **online training**: the algorithm had to be re-trained manually in order to include newly acquired data;
- **sensible threshold**: the threshold that determines whether or not a given measurement had to be considered an anomaly was arbitrarily chosen and set as an algorithm parameter.

The new version of the module is now based on the **GAM** (Generalized Additive Model) algorithm. The following formula represents the model that has been used:

$$Y(t) = \varepsilon_0 + \sum_i \quad f_i\big(x_i(t)\big)$$

- $Y(t)$ is the energy consumption at time t;
- $x_1$is the day of year at time t;
- $x_2$is the day of week at time t;
- $x_3$is the hour of day at time t;

- $x_4$is the luminosity at time t;
- $x_5$is the occupancy at time t;
- $x_6$is the temperature at time t

For each building, a GAM is trained over all the available data up to the last 24 hours, thus leading to an estimation of the differentf_i. A grid search is performed in order to select the best combination of parameters for the algorithm.

The trained algorithm is then run over the data of the last 24 hours. Each consumption value that is above the 95% confidence interval estimated by the GAM is considered an anomaly and is stored in the Building Knowledge Base as an Event with type POWER_CONSUMPTION_ANOMALY. Thus, the building manager has the possibility to review the result of the analysis and delete the events that are actually false positives.

Concerning the three issues raised above, this new version of the module offers the following solutions:

- **exogenous factors**: luminosity, occupancy, temperature, time of the day and type of the day are all directly taken into account in the Generalized Additive Model;
- **online training**: the analysis is performed at the end of each day and each time the algorithm is trained over all the available data up to the last 24 hours, excluding all the detected anomalies;
- **sensible threshold**: the 95% confidence interval of the estimated value is chosen as the threshold to determine what an anomaly is and what is not. Although the 95% is still an arbitrarily chosen value, using the confidence interval of the trained algorithm allows to set a threshold whose meaning can be easily related to the shape of the analysed dataset.

The following charts show an example of what are the results of the algorithm, trained over the dataset of the school in Prato, over a span of multiple days and then the detail of a single day in which there are no anomalies (Figure 5) and in which there are (Figure 7).

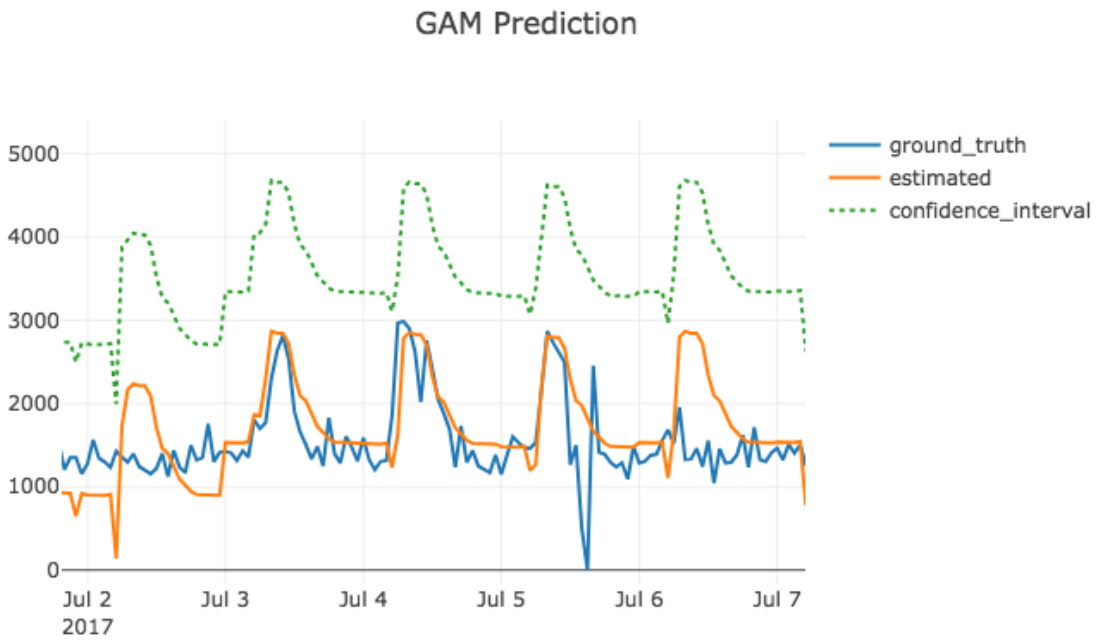

Figure 4 Behaviour of the GAM algorithm - multiple days - no anomaly - Prato

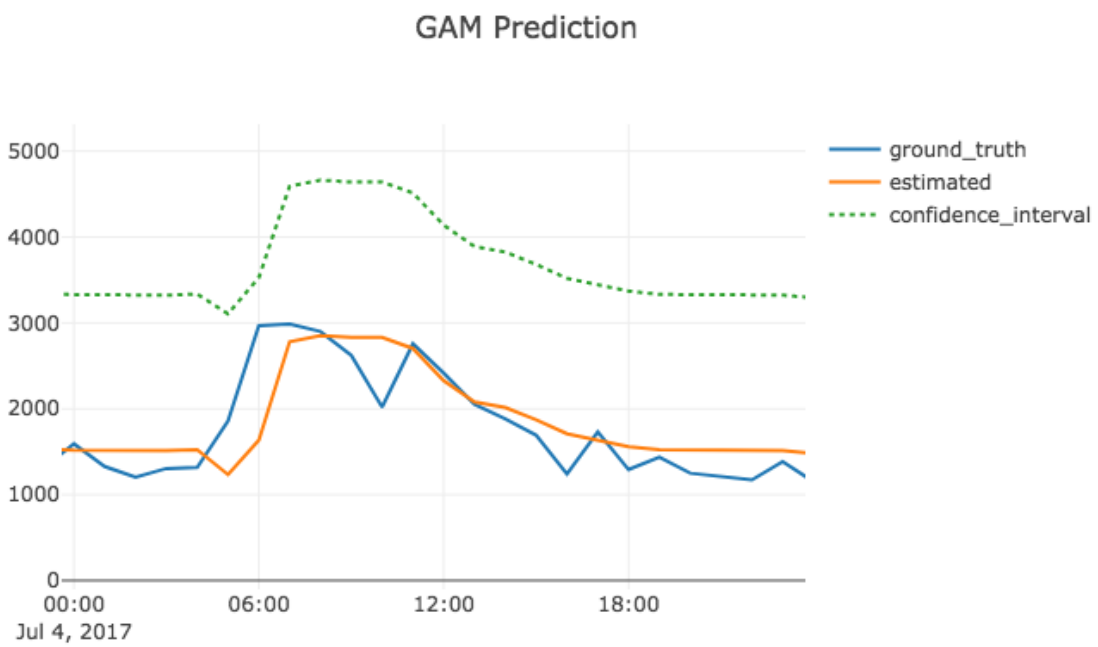

Figure 5 Behaviour of the GAM algorithm - single day - no anomaly - Prato

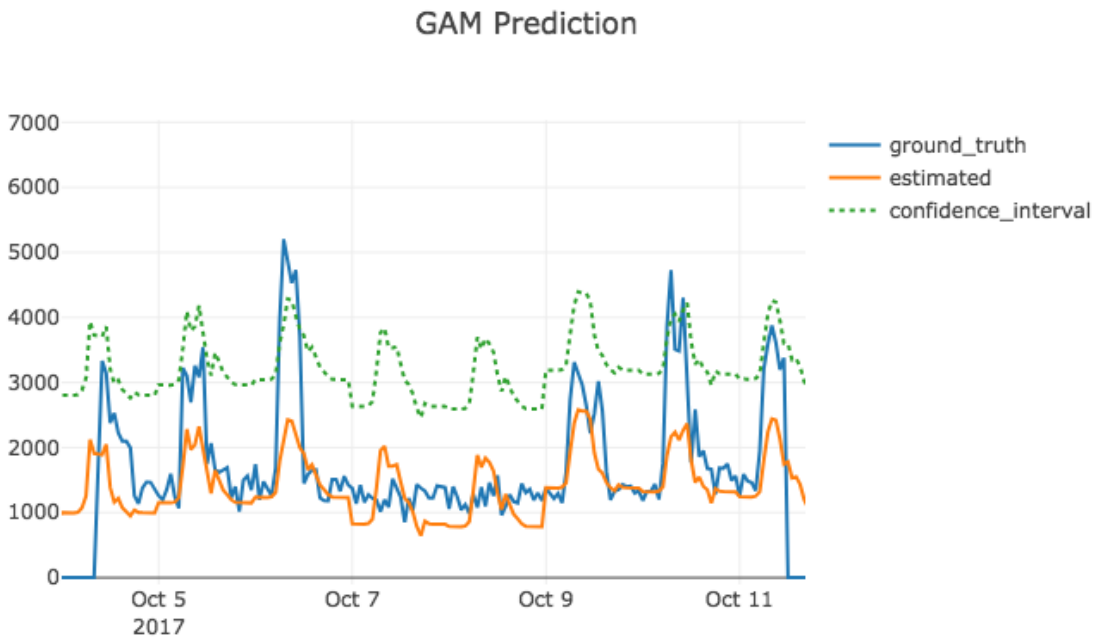

Figure 6 Behaviour of the GAM algorithm - multiple days - anomalies detected - Prato

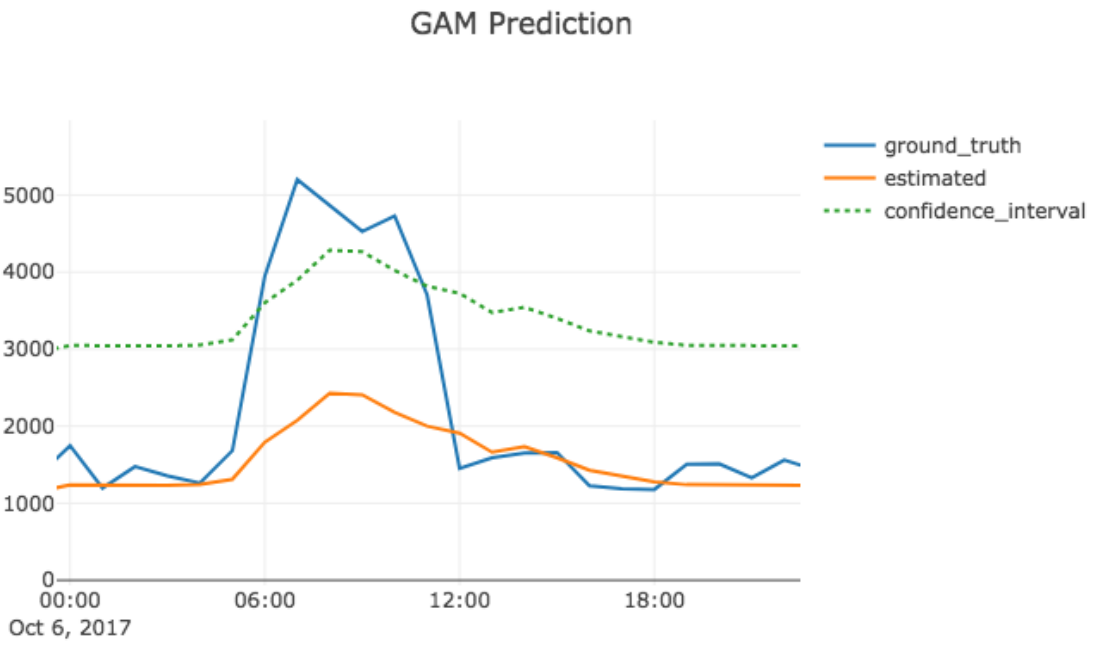

Figure 7 Behaviour of the GAM algorithm - single day - anomalies detected - Prato

Table 4 reports some key indicators of the performances of the GAM algorithm after having being trained over the datasets of the buildings currently available in the platform, namely the Mean Absolute Error (MAE - expressed in Wh), the Mean Squared Error (MSE), the Symmetric Mean Absolute Percentage Error (sMAPE), and the R2 score.

The MAE (mean absolute error) is the average of the absolute difference between the value estimated by the algorithm and the actual one. It uses the same scale as the data, thus in this case it is expressed in Wh.

The MSE (mean squared error) is an indicator of the quality of an estimator. It is always non-negative and values closer to zero are better.

The MAPE (mean absolute percentage error) can be used to express the accuracy of an estimator as a percentage. However, it has some major drawbacks, such as not being defined for zero values and it is highly biased toward over-estimations since it has a lower bound, 0, but no upper bound. The sMAPE (symmetric MAPE) is a slightly modified version of the MAPE which tries to address the second issue by enforcing the error to be between 0-100 but it still does not treat equally over- and under- estimations.

The R2 score (r-squared score) is a value that measure how close the actual data are to the fitted model. It is expressed as a percentage and can be interpreted as the amount of variation that is explained by the model over the total amount of variation in the data.

Table 4 GAM algorithm performance

METRICS	AVG	MAX	MIN	STD
MAE	365.16	707.37	17.61	223.03
MSE	549750.05	2193698.23	992.06	602888.67
sMAPE	0.30	0.49	0.11	0.12
R2	0.39	0.88	0.14	0.18

9 Recommendation Engine

This section describes the additional features that have been developed for the final release of the Recommendation engine, which essentially aims at improving the ease of use of the offered features. The newly added features are listed hereafter and described in the following subsections:

- notification to end-users via email
- assisted rule creation
- dashboard for system administrators

We also report on minor changes in the engine implementation which have an impact on the APIs (i.e. time intervals expressed as user-friendly strings as well as Cron strings and storage and retrieval of rules' default values) and we provide some performance metrics on the actual rule engine implementation.

The documentation of the Recommendation Engine is available on the web [Engine].

9.1 Requirements

Code	Description	Priority
Re.1	It must be able to query the Data Storage block APIs for gathering sensors measurements	High
Re.2	It should be able to query the Analytics block APIs for gathering analytics on sensors measurements	Medium
Re.3	It should be able to query the Building Knowledge Base APIs for gathering information on school buildings	Medium
Re.3	It must generate recommendations at the occurrence of energy efficiency relevant conditions	Medium
Re.4	It must log events that are considered relevant to end-user applications' purposes (e.g., report generation)	High

Re.5	It must push notifications at the occurrence of energy efficiency relevant conditions and recommendations generation	High
Re.6	It must expose an API for accessing events	High
Re.7	It should allow the specification of customized rules per school building/area/sensor (e.g., customized threshold values, customized notification message, etc.)	Medium
Re.8	It should allow to define logical group of rules for simplified management and visualization purposes	Medium
Re.9	It may allow the specification of new rule instances and new group of rules	Low
Re.10	It may support separate notification channels for different schools	Low
Re.11	It should use the GAIA AA service	Low
Re.12	It should send notification via email	Low
Re.13	It should assist the end user in creating new rule instances by offering an appropriate API providing a tentative rule instance filled in with default values (e.g. threshold values) as well as parameters' values customized according to the user query (e.g. sensors' URI depending on the site ID provided by the end user)	Low

9.2 Email notification

By default, the action that is triggered upon condition verification consists in sending a notification to the BMS through websocket and storing it in the event database, as described in D2.1. This feature allows adding an email notification channel to the default action.

A rule instance can be provided with an optional *email* field containing a list (separated by commas) of email addresses. Therefore, in order to add an email notification, this field has to be filled in with the desired email addresses. When the rule is triggered, (i.e. the condition is verified) an email is

sent to all the recipients aside from the default action (notification to the BMS storage in the event database). Email delivery is actually implemented by using the Mailgun service [mailgun]. At present, the Gaia's Mailgun account is active with a basic free profile with a limit of 10000 messages per month. Notification emails are sent by the Mailgun server with the rules@gaia-project.eu sender address.

9.3 Time intervals and schedules

Schedules and Calendars of the buildings/schools and related sub-sites provide relevant information on how the school buildings/sites are used and lived by users and therefore may appear as parameters in rules' conditions. Moreover, configurable time intervals can be handled by the engine in order to manage the periodic checking of rule instances and action triggering. This allows to properly set the engine behaviour for each rule instance in order not to overwhelm the end user with too many notifications related to a condition holding for subsequent time intervals.

Time intervals can be specified in the form of Cron strings or simple, human readable, interval strings. In the table below (Table 5) some examples of both types of strings are shown. In most cases, Cron strings are useful when dealing with periodic conditions (e.g. every Sunday), while interval strings are more suitable when dealing with long, continuous, intervals.

Table 5 Time intervals representation

Cron	* * * ? * SAT-SUN	Each Saturday and Sunday
Cron	* * 12 ? * SUN	Every Sunday at 12:00
Interval	10/05/2018-30/05/2018	
Interval	10/05/2018	From 00:00:00 to 23:59:59
Interval	15/11/2018 13:00:00-15/11/2018 23:00:00	

9.4 Default values store

This feature aims at providing an API for retrieving and modifying the structure and related default values of a rule (e.g., parameters of the rule conditions, default values, default suggestion message, etc.). Moreover, it provides CRUD operations on the default values for the different rule classes and it extends the schema of the classes with some useful information such as a textual description of each field and a Boolean flag telling if the field is mandatory or not. The user may add custom information to each field description for future uses.

A "default entity" returned by this API is made of two main blocks: *fields*, containing the list of the properties of the rule class along with their descriptions, default values (if applicable), and *suggestion*, which contains the default textual suggestion in different languages. The indexes of the suggestion map are the standard (ISO 639-1) language identifier (e.g., it for Italian, en for English, fr for French). Such "default entities" are persisted for later retrieval and manipulation in what we call "default values store" (implemented as an additional database). When a user sends a request for retrieving a default entity for a given rule class, the API will return the values persisted in the default value store.

If a given class has no default value on the store a “Not found” error will be returned when the store is queried, the user can force the system to output a default based on the backup values hard coded as specified in the API documentation.
For some parameters, the concept of default value may be not applicable at the level of rule class. For instance, fields related to sensor identifiers (URIs) do not have a default value, since their value depend on the site where the rule instance will be attached. In that case it is possible to specify the measurement type associated to that URI. This choice has a twofold application: i) it provides end users with information on the type of measurement evaluated in the rule class’ condition when returned in the response message of a default entity retrieval request and ii) it can be used by the system to fill in the field with an appropriate value during a rule instance creation procedure (i.e. the system is guided to look for the right sensors identifiers). As an example, the *temperature_uri* field should have a default value set to *“Temperature”* telling the system to look for a sensor measuring the temperature in the site that the rule is being attached to. If the needed property is not specified the system tries to infer it using the fieldname (e.g., humidity_uri → Relative Humidity). Such procedure is used by the system is the “Assisted rule instance creation” procedure described in the following section.

Hereafter we show the schema of a default entity and an example of a default entity returned for a Simple Threshold rule class.

```
{
    "fields": {
        "field_name": {
            "value": DEFAULT_VALUE,
            "description": "Textual description",
            "required": true|false,
            "custom_attribute_if_needed": VALUE_OF_THE_CUSTOM_ATTRIBUTE
        },
    "suggestion": {
        "A_ISO_LANGUAGE_IDENTIFIER": "Textual suggestion in language A",
        "B_ISO_LANGUAGE_IDENTIFIER": "Textual suggestion in language B",
        "C_ISO_LANGUAGE_IDENTIFIER": "Textual suggestion in language C",
    }
}
```

```
{
    "fields": {
        "temperature_uri": {
            "value": "Temperature",
            "description": "Ciao",
            "required": true,
            "custom_field": "value"
        },
        "humidity_uri": {
            "required": true
        },
        "occupancy_uri": {
            "value": "Occupancy",
            "description": "Occupancy",
            "required": true
```

```
        },
        "threshold": {
            "value": 30,
            "description": "The threshold",
            "required": false
        },
        "name": {
            "value": "Simple Threshold rule",
            "description": "The name",
            "required": true
        }
    },
    "suggestion": {
        "it": "Esempio",
        "en": "Example",
        "el": "Παράδειγμα"
    }
}
```

9.5 Assisted creation of rule instances

The aim of this service is to help the user during rule creation, trying to suggest the most suitable value for each field of the rule. The user has to provide the identifier of the area to which the rule has to be attached and, of course, the class of the rule to instantiate. The engine implements an interactive procedure: upon the user request, the system returns a tentative answer containing a rule instance with as more fields as possible filled in (i.e. default values as well as values depending on the user' query, e.g. sensors' URIs).

The retrieval of the appropriate sensor URIs depends on the site specified in the user query and it is implemented leveraging the default value store feature (see section above). Typically, the system will search for a sensor attached to the given site identifier. However, according to the condition that has to be verified in the given rule class, this is not the unique option that may hold. Indeed, for some rule classes, the sensors to be retrieved should be attached to the school (e.g. a weather station), to an external resource (e.g. a weather service), or to a parent site (e.g., aggregated power consumption). In order to handle these cases, some special prefixes can be used by the developer when creating a rule for the sensor's field name in order to express where the sensor associated to a URI should be placed, as described in Table 6.

Table 6 Convention for sensors' location

`ext_`	the sensor should measure the external value, the system will look for a sensor tagged with "External" in the root
`root_`	the sensor should be attached to the root site (i.e. the school itself)
`parent_`	the sensor should be attached to the parent site

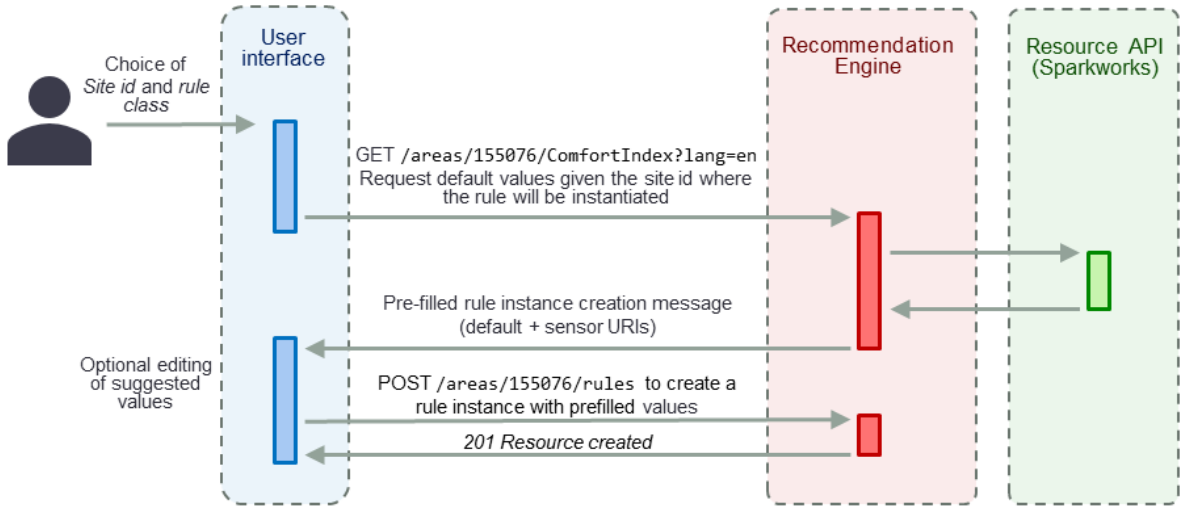

Figure 8 Assisted creation of a Comfort Index rule instance

Figure 8 shows a possible flow for the assisted rule instantiation. The end user interaction with the rule engine is mediated by an application offering a user interface (e.g. the BMS). The end user has to specify the class of the rule to be instantiated and the site the rule should refer to (e.g. a classroom or a floor). The interface uses this information to send the appropriate REST request to the Recommendation Engine. The Engine handles the request by assembling a response containing a rule instance pre-filled with default values (where applicable) and with customized values (i.e. sensor URIs) retrieved by querying the Resource APIs.

The end users inspect the returned response, modify the pre-filled value, as he/she deems appropriate and finally send to the system the rule instance to be added in the system. This last step is handled by the user interface as a POST request sent to the Recommendation Engine with the rule instance with filled values sent in the message body.

9.6 Rule classes available

At present, the rule engine is endowed with the implementation of rule classes. Some rule classes have a generic scope of application (e.g. composite rules, simple threshold rules), while other rules have been purposely implemented for GAIA scenarios. The behaviour of the implemented rules thus is based on hints taken from existing literature and feedback collected from GAIA partners involved in the trials activities.

Composite rules

Rules whose condition is based on the conditions of the children rules.

AnyCompositeRule
The rule is triggered if any of the conditions of the children rules is true simulating a logical “or”.
AllCompositeRule
The rule is triggered if all the conditions of the children rules is true simulating a logical “and”.
RepeatingRule
The rule is triggered if the condition of the child rule is verifies more than N (customizable) times consecutively.

Custom rules*

Rules whose basic behaviour is defined programmatically, different instances have a different set of configurable parameters. These rules have been defined by taking into account main GAIA objectives. This set can be easily extended. Rules can also be added as a result of trial activities.

Don’t waste energy
The basic behaviour is as follows: if the room is not occupied and the active power measurements is greater than a threshold, a notification is delivered to suggest to switch off lights and/or devices.
Exploit natural light
If the lights are ON but luminosity is above a given threshold and the room is occupied, the rule checks if the artificial lighting can be switched off without compromising the user comfort.
Power Factor
The rule logs events when the power factor falls below a threshold. The events are recorded to be later retrieved and analysed by the building manager. This information helps deciding if some corrective actions are needed (e.g. install a capacitor).
Holiday shutdown
The rule generates a message for the building manager delivered before a holiday period to remind him/her to check and switch off devices before the vacation period.
Temperature Forecast
The aim of this scenario is to warn people about the likely sudden decrease of the temperature in the next days, suggesting to wear warmer clothes and suggesting the building manager to decrease the comfort temperature of the heating system by one degree to avoid overconsumption.
Comfort Index
At high temperature and humidity conditions, the comfort factor can be controlled with the concept of Heat Index. The system sends an alert to the building manager to make him aware about the problem.
CO2 Level
The aim of the scenario is to maintain a good level of CO2 inside an area. This is done by monitoring the level and, when it exceeds the comfort level (e.g. 1000 ppm), a notification is triggered.

Template rules

Implement a basic, predefined behaviour that has to be further specified with a set of configurable parameters that better specify the scope of the rule.

Simple Threshold Rule
This rule is fully customizable by the user. It evaluates a simple expression in the form *value operator threshold* (e.g., temperature <= 35.0)
Expression Rule
This rule evaluates a user defined expression which may contain numbers, operators, user defined variables and measurement retrieved from the platform.
Schedule Reminder Rule
The aim of this rule is to remind the user to execute one or more action through notifications and / or emails. For example, it can be used to remind the building manager to reverse the rotation direction of the ceiling fan during season changes.

Table 7 provides an overview of the number of rule classes and instances present in the recommendation engine at December 2017. We also report on the average number of generated events per days.

Table 7 Rules classes and instances currently available

	value/average	standard deviation
Number of rule classes	13	-
Number of instantiated rules	59	-
Number of user defined instances	14	-
Average number of events generated per day	104	31

9.7 Dashboard

The Dashboard is a web application that allows system administrators to easily monitor the status of the engine, navigate the tree of instantiated rules and visualize the rate of detected events.
The main page of the dashboard shows some general information about the status of the recommendation engine such as the list of the latest generated events (top left in Figure 9), a line plot of the number of events generated within the latest seven days (bottom left) and a pie chart which shows the percentage of rule instances for each rule class (right side).

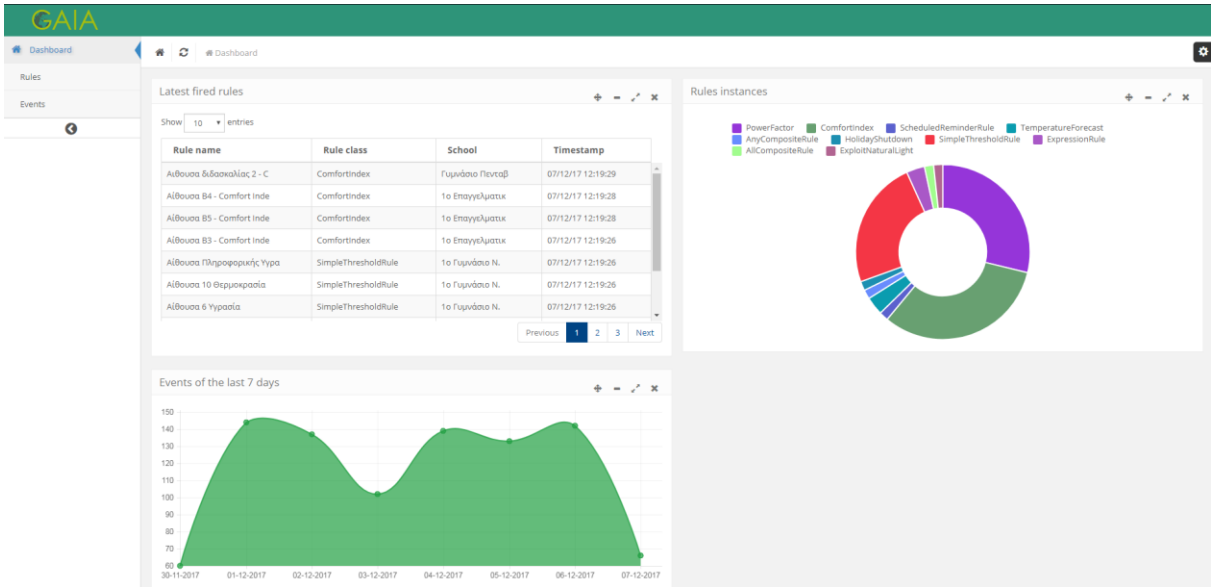

Figure 9 Dashboard main page

Figure 10 shows the rules page of the dashboard where all the buildings managed by the recommendation engine are listed using a tree view. It is possible to navigate the rules instantiated for each area of the building and view a brief summary of each one.

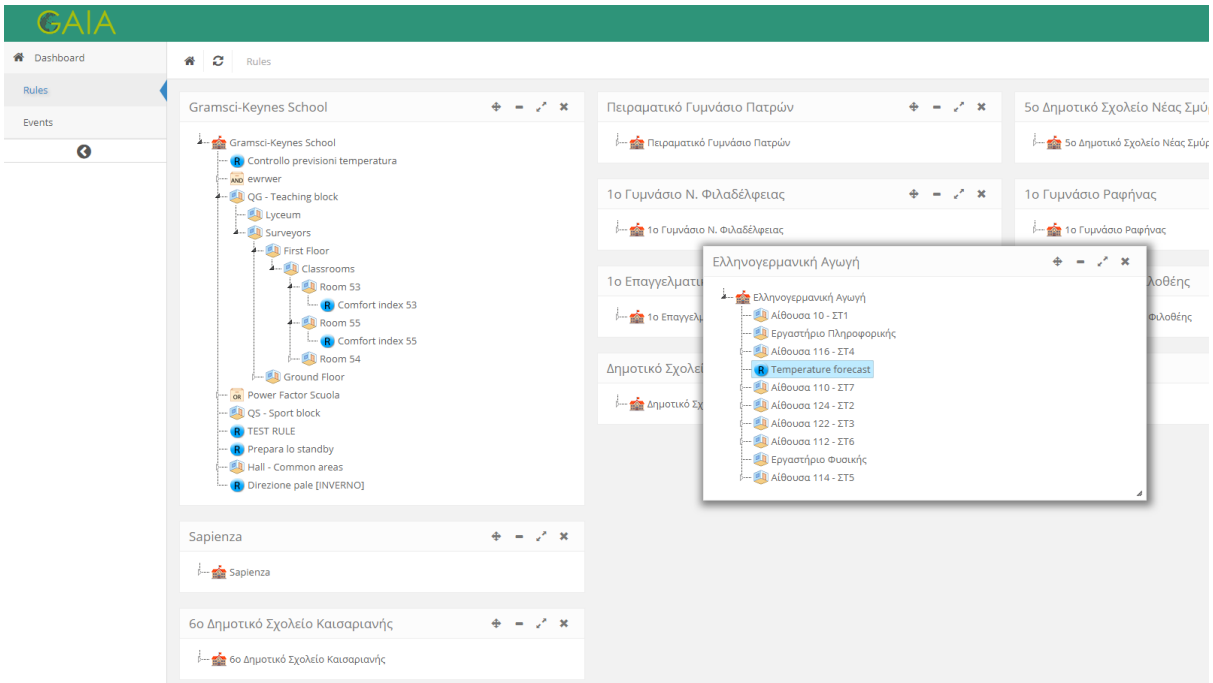

Figure 10 Dashboard Rule navigation

9.8 Performance metrics

In this section, we report on testing activities carried out to evaluate the performance of the recommendation engine.

Testing methodology

The tests aims at measuring the time required by the recommendation engine for performing some reference actions by querying the database and creating a JSON response according to the request without taking into account network latency; for this reason tests have been executed in a single machine, i.e. the client and the recommendation engine run on the same machine.

Hereafter we provide a description of the used metrics and information on how tests have been conducted. Results are reported in Table YY.

Scheduled Iteration time

Each 5 minutes, the recommendation engine performs a check on the registered rules. For scheduled iteration time, we thus mean the time needed to accomplish an iteration.

We provide an average value of this metric. For each iteration, the elapsed time is stored during a period of 24 hours approximately (precisely 271 iterations). The average iteration time is computed over all these iterations. In this test, we take into consideration also the time elapsed when some errors occurred during, for instance, a query to the GAIA platform. The maximum time per iteration during the monitored period is 21.8 seconds while the minimum is 5.4 seconds. The average iteration time is about 8.4 seconds with a standard deviation of about 3.2 seconds. The reason of this variability is that some of the rules use cached values, while other have to retrieve further data from the platform or external web services. So the timing depends on what and how many rules are actually fired according to their settings (for instance rule A is programmed to be fired every 2 hours while rule B every 5 minutes).

Query time for events

We defined four types of event query (see Table 8). For each type of event query, we created a set of 20 event retrieval requests. At every iteration the recommendation engine is queried and the time elapsed is recorded. The parameters of the query are different among the iterations: the start and end of the time range are modified in order not to have queries including cached values (trivial queries).

Rule Suggestion time

We focused on the first part of the assisted rule creation workflow, i.e. the retrieval of a rule instance with values filled in by the engine. We considered 26 requests for different areas mixed between PowerFactor and ComfortIndex rule classes. The availability of default values for the rule fields have been pre-verified in order to prevent misleading responses for performance evaluation.

Rule Creation time

The client sends 30 requests for creating rules belonging to SimpleThresholdRule class and associated to a demo school. The client waits 30 seconds between each request. The average creation time value is reported in Table 8.

Table 8 Performance metrics and test results

	value / average	standard deviation	unit
Average iteration time	8808	3372	ms
Average query time for events (limit 10, filtered by school, 5 days)	379	78	ms
Average query time for events (limit 1000, filtered by school, 1 day)	371	44	ms
Average query time for events (limit 1000, filtered by school, 10 days)	584	88	ms
Average query time for events (latest 100, filtered by rule class)	1149	456	ms
Average suggestion time	690	223	ms
Average rule creation time	632	111	ms

10 Sequence diagrams for main GAIA processes

In [GAIA2.1] we showed some examples of interaction between WP2 components and WP3 applications in relevant GAIA processes, namely:

- Recommendation generation and dissemination.
- Sensors data visualization.
- Participatory Sensing.
- Third-party Application' s access to GAIA Platform services

In Section 10, we showed how the assisted rule creation feature offered by the Recommendation engine could be used by a UI. Hereafter we report on supported processes leveraged by the features offered by the Analytics module.

Hereafter we provide two additional GAIA processes, showing two examples of third party application's use of the services provided by the Analytics module leveraging also the other GAIA platform modules (e.g., Building Knowledge Base and Data Storage).

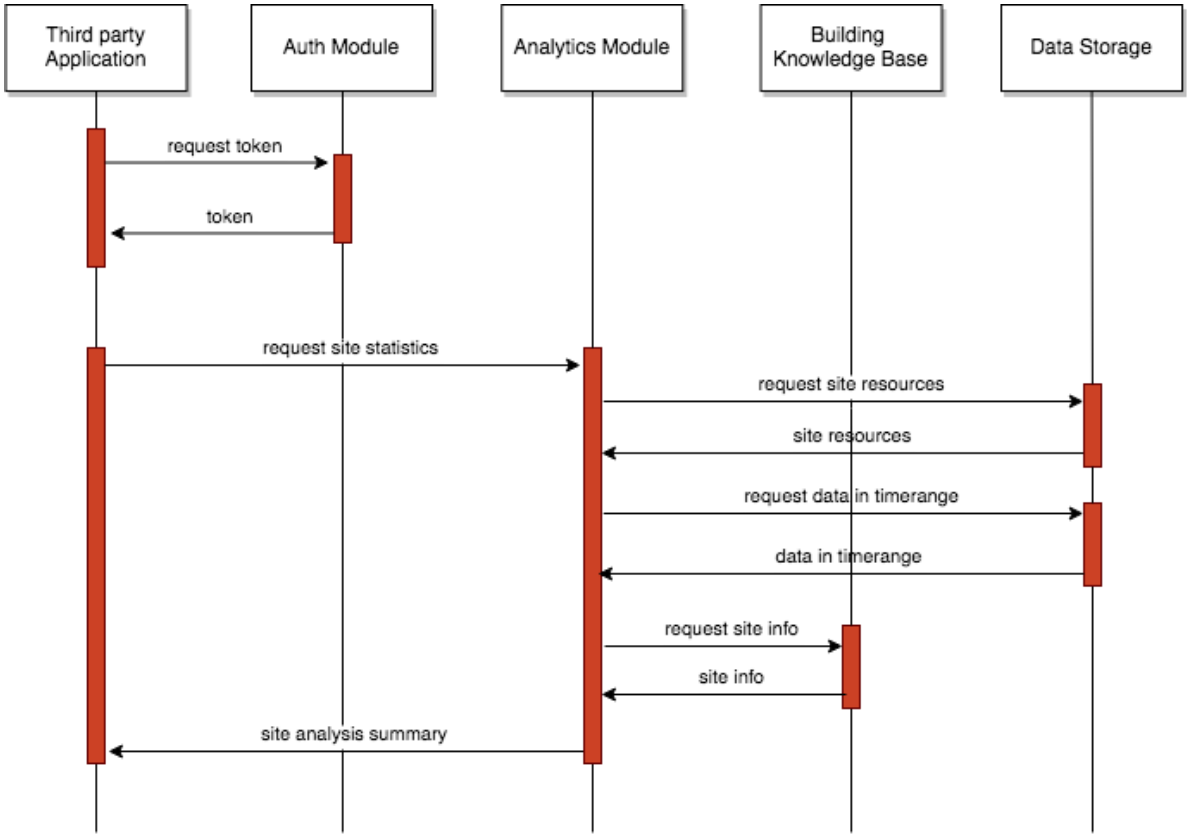

Figure 11 Example of interaction with the Analytics module to retrieve statistics about a site

Figure 11 describes the various interactions between GAIA platform modules when a Third-party Application or a user requests statistics about a site to the Analytics module. The Analytics module retrieves the requested data from the Data Storage and the info about the site from the Building Knowledge Base. It then proceeds to compute various statistics about the site and envelopes them

in an analysis summary object that is returned to the client.

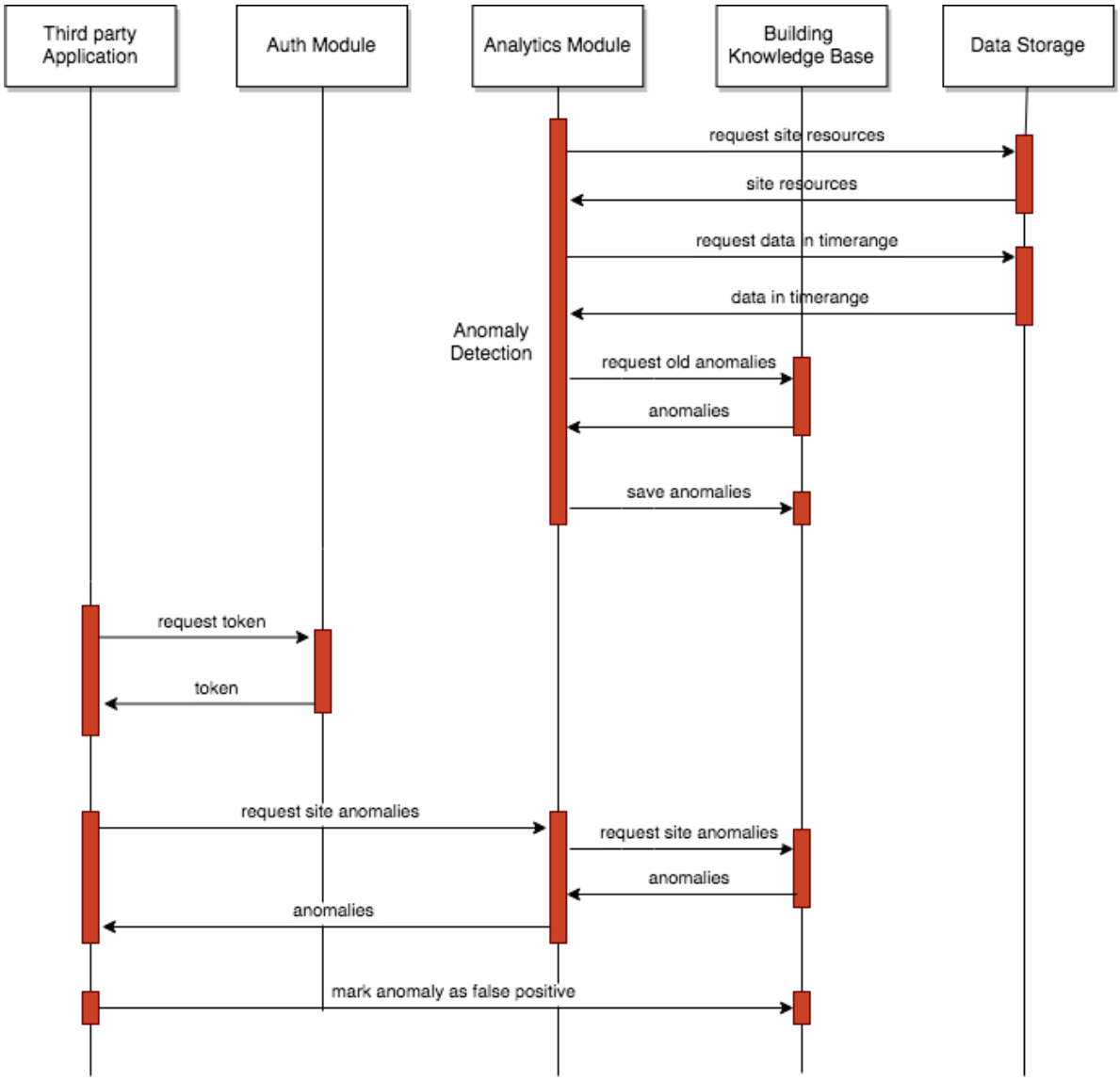

Figure 12 Example of interaction between modules during the Anomaly Detection task and when a Third party requests the detected anomalies for a site

Figure 12 provides an example of the interactions between modules during the anomaly detection procedure and when a third-party application or a user requests the anomalies of a specific site. In the former process, the Analytics module interacts with the Data Storage to build the training set required to train the anomaly detection algorithm. It also gets from the Building Knowledge Base old anomalies in order to filter them out of the analysis. Once the anomalies have been detected, they are stored in the Building Knowledge Base. A client can then retrieve or mark them as false positives either by performing the request to the Analytics module or by accessing directly the Building Knowledge Base.

11 Deployment

In a microservice environment, like the GAIA platform, apart from the effort of developing each microservice application, several other issues arise like tracking down the services dependencies, scaling the application, and updating individual microservices without affecting the entire platform.

To address those issues we introduced Docker [DOCKER] containerization and deployment on a Docker swarm cluster. Software containers are isolated and immutable images providing specific functionality. Containers are able to make a software run reliably and on any environment. Software components are broken into manageable, functional microservices that are packaged individually with all of their dependencies. However, maintaining and deploying many microservices can be a very complex challenge. Each of them can be written in a different programming language or require different application servers to live one or can also use different set of libraries. If each service is packed as a container, all of those problems are eliminated since, containers are self-sufficient bundles containing everything is needed to run in an isolated process and are immutable. Docker is a software tool that manages and orchestrates containers.

Nowadays Docker is the leading software for containerizing a service. Docker provides both community and enterprise edition. In the context of the GAIA project, the community version has been utilized. Docker provides an orchestration tool in Docker Swarm mode. Orchestration refers on automating and simplifying the management of containerized applications at scale. Scaling and updating the GAIA platform containerized microservices is handled seamlessly by Docker swarm within the Docker Swarm cluster.

More specifically, the GAIA platform deployment utilizes Docker swarm since it enables the following features:

- **Integrated, centralized cluster management**: the installed Docker Engine CLI enables the centralized creation, deployment and management of the GAIA cluster.
- **Scaling**: for each GAIA microservice, the number of replicas can be easily declared and scaling up or down a microservice is automatically handled by the Docker swarm.
- **Desired state reconciliation**: the Docker swarm constantly monitors the GAIA cluster state and reconciles any differences between the actual state and the expressed desired state by creating or replacing microservice replicas.
- **Load balancing**: On the GAIA Docker Swarm cluster, a DNS entry is automatically assigned to each running microservice. An internal load balancer, using the round robin algorithm, distributes client requests to all running microservices within the GAIA cluster based on the DNS name of the request.
- **Security**: Each Docker swarm node in the GAIA cluster enables TLS mutual authentication and encryption providing secure communication with all other nodes.
- **Service updates**: With Docker Swarm, GAIA microservice updates are applied to the nodes incrementally. The GAIA cluster is able to control the delay between service deployments to different sets of nodes. Apart from this, if a GAIA microservice deployment fails for any reason, the service is able to rollback to its previous healthy version.

11.1 Continuous Deployment

The development process of the GAIA platform services follows a Continuous Deployment (CD) pipeline. When the developers' code commits are merged to the mainline repository branch the Continuous Integration tools that monitor the project repository is able to detect the changes on the mainline branch and trigger the CD pipeline tasks. The CD pipeline executes different set of tasks to verify that the project source code works as expected. Under the condition that the unit tests, the integration tests and the system tests have been executed successfully the final step of the CD pipeline is executed i.e., the services packaged code is deployed to the production environment. If any of the steps of the CD pipeline fails, the process is aborted and the appropriate persons are notified for the failure. If the CD pipeline has been executed successfully, the source code changes are promoted to a release. The last mile of the CD process refers to automatically deploying each produced release to the production environment.

In the context of GAIA project, a Continuous Integration (CI) server has been installed and configured with a full CD pipeline. Therefore, when new source code reaches the master branch of the project repository the responsible task compiles the source code and executes the relevant unit and integration tests. Upon the successful execution of this task, the results are pipelined to the task that prepares the packaged source code and the relevant Docker images for deployment. Finally, on the deployment task, the containerized software is deployed on the GAIA cluster and functional tests are executed to ensure the proper operation of the platform. In case of any failure of those tests, the newly deployed software is automatically rolled back to its previous healthy state.

11.2 Cost Analysis

The GAIA platform is deployed in a Docker Swarm cluster consisting from 3 nodes. The dedicated hosts for the nodes cluster are provided by Contabo [CONTABO]. The technical specifications and the cost for each node of the cluster are presented in Table 9.

Table 9 Contabo offer

Contabo	vCPU	Ram	Disk	Cost / Year
Node 1	4 Cores	14 GB	1000 GB SSD-boosted	143,88 euro
Node 2	6 Cores	24 GB	600 GB SSD	179,88 euro
Node 3	6 Cores	24 GB	600 GB SSD	179,88 euro
Total	**16 Cores**	**62 GB**	**1000 + 1200 SSD**	**503, 64 euro**

Table 10, Table 11 and Table 12 offer a cost estimation on using similar nodes for the GAIA platform deployment cluster in Amazon AWS, Microsoft Azure Cloud and Google Cloud Platform services as stated by the referenced cloud providers for the year 2018. In the case of AWS, t2.xlarge is the cheapest node with similar characteristics with the Contabo provider while in the case of Google

Cloud 375 GB is the minimum option for the disk storage. Therefore, we have chosen Contabo as the cloud infrastructure provider due to the competitive pricing and the provided infrastructure specifications.

Table 10 Amazon AWS offer

Amazon AWS	vCPU	Ram	Disk	Cost / Year
Node 1 (t2.xlarge)	4 Cores	16 GB	-	1040 euro
Node 2 (t2.xlarge)	4 Cores	16 GB	-	1040 euro
Node 3 (t2.xlarge)	4 Cores	16 GB	-	1040 euro
Total	**12 Cores**	**48 GB**	**-**	**3120 euro**

Table 11 Microsoft Azure offer

Microsoft Azure	vCPU	Ram	Disk	Cost / Year
Node 1 (B4MS)	4 Cores	16 GB	32 GB SSD	1040 euro
Node 2 (B4MS)	4 Cores	16 GB	32 GB SSD	1040 euro
Node 3 (B4MS)	4 Cores	16 GB	32 GB SSD	1040 euro
Total	**12 Cores**	**48 GB**	**96 GB SSD**	**3120 euro**

Table 12 Google Cloud Platform offer

Google Cloud Platform	vCPU	Ram	Disk	Cost / Year
Node 1 (n1-standard-4)	4 Cores	15 GB	375 GB SSD	1408 euro
Node 2 (n1-standard-4)	4 Cores	15 GB	375 GB SSD	1408 euro
Node 3 (n1-standard-4)	4 Cores	15 GB	375 GB SSD	1408 euro
Total	**12 Cores**	**45 GB**	**1125 GB SSD**	**4224 euro**

12 Conclusions

WP2 has successfully developed and deployed a service platform that is accessed by WP3 applications to support the GAIA trials during the school/academic year starting since autumn 2017. The GAIA DoW outlined a social networking game, a serious on-line game, the building manager application and a participatory sensing application. The first release of the components of the GAIA Service Platform has been documented in [GAIA2.1]. The platform design and implementation have been refined in this document, taking into account the feedback from WP3 and WP4 activities and results [GAIA3.1, GAIA3.2, GAIA3.3].

This document (D2.2) complements [GAIA2.1] in documenting the final release of the GAIA service platform and provides some information of performance metrics gathered to date. In this deliverable, we have also reported details on the deployment configuration of the software infrastructure, performance metrics and software artefacts. Hereafter, we conclude the document by discussing the achievement of pertinent KPIs introduced in [GAIA1.1].

In Section 11 we provided details on the deployment of the GAIA core modules on a Docker swarm cluster, which allows tracking down the services dependencies, scaling the application, and updating individual microservices without affecting the entire platform. We also discussed choices regarding cloud hosting. These deployment choices have been driven by the need of assuring a minimum uptime of 99% (KPI TS.1 System uptime). To monitor and measure the availability of the GAIA core platform maintained by SparkWorks we utilized the New Relic (https://uptimerobot.com/) tool.
Figure 13 and Figure 14 show some results concerning system uptime and response time.
Figure 13 Response time and System uptime of Data Storage API

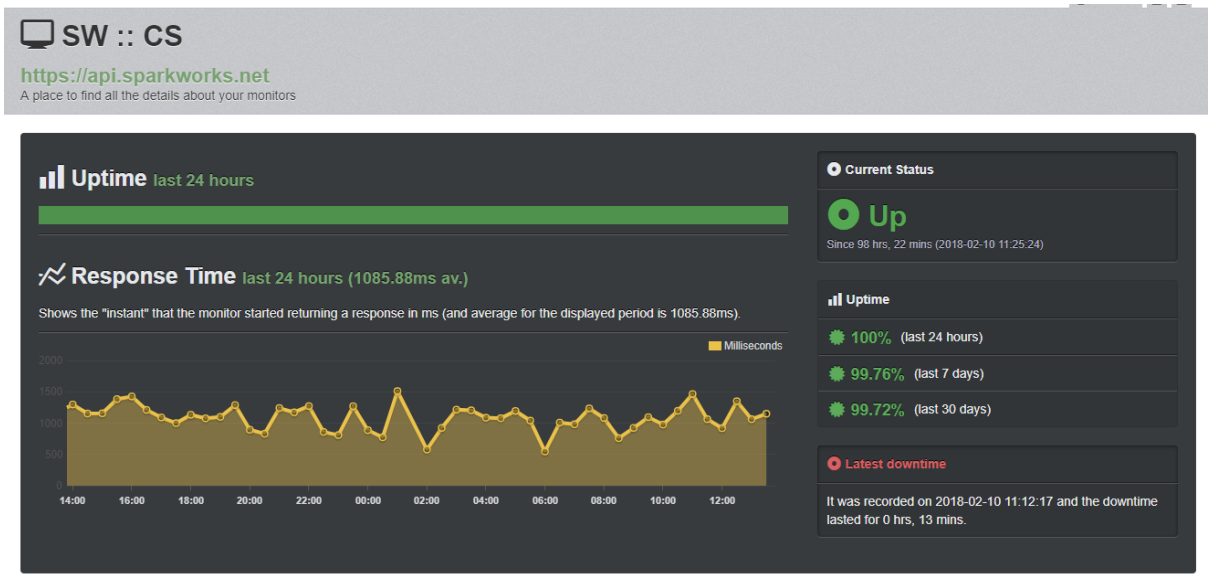

Figure 13 Response time and System uptime of Data Storage API

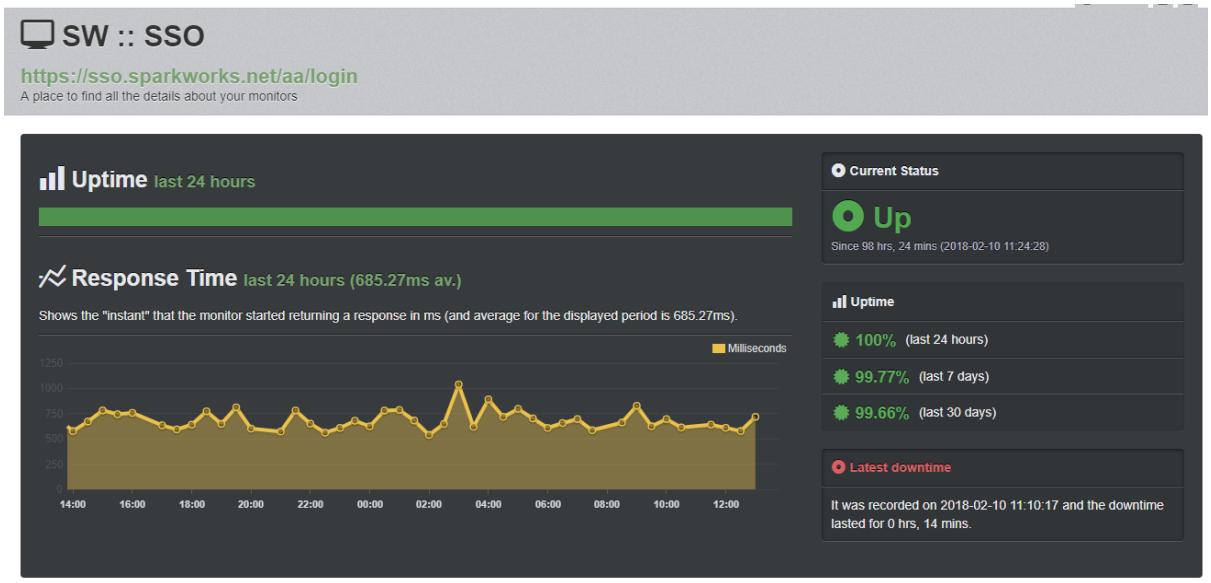

Figure 14 Response time and System uptime of Authentication & Authorization API

Measurement data are processed by the platform as they arrive and provided at the desired level of granularity, i.e., 5_min, quarter, hour, day, month (Data granularity). Details on the stream processing mechanism implemented in GAIA have been provided in D2.1. We also provided for each module a discussion on System performance (TS.3), including average input message rate, processing rate and throughput.

The current pressure on data injection rate is 35 events/sec. However, we have conducted a stress test on the SparkWorks platform where the injection rate was about 7000 events/sec and the average processing rate achieved by the platform was 5833,33 events/sec. The injection rate is only dependent on the broker system (RabbitMQ) which, due to the dockerized nature of our deployment, can be easily scaled horizontally in order to achieve much larger ingestion rates. Those results are presented in the context of a paper accepted and pending publication in the IEEE International Conference on Communications (ICC) 2018 entitled “A Fog Computing-oriented, highly scalable IoT framework for Monitoring Public Educational Buildings”. Apart from the performance evaluation results of our platform operating in our cloud, we have also provided performance evaluation results of the platform deployed and operated on low-powered edge devices, such as Raspberry PI boards, where it achieved average processing rate 15.35 events/sec and on Zotac Atom systems achieving average processing rate 2692.31 events/sec.
The implemented services are available through fully documented open APIs (TS.4), as described in D2.1, where 5 APIs are presented, while latency measurements for a subset of most relevant queries have been provided in this document (TS.5).

At M24, the GAIA service platform integrates data form 3rd-party sensor infrastructures. As documented in D1.1 and D2.1, the infrastructure deployed in the GAIA pilot schools is provided by the following list of technology providers:

- Meazon: devices providing environmental monitoring and consumption measurements in several Greek schools.
- Synelixis: devices acting as weather stations deployed at a number of school buildings.
- Netsens: devices providing environmental monitoring and consumption measurements in Prato.
- OVER: an extensive network of power consumption meters deployed inside a number of building at the Sapienza University of Rome campus.
- Raspberry Pi: a number of new school buildings added to the building fleet of the project are based on this type of devices, utilizing commercial-grade sensing extensions.
- Libelium: a number of devices from this company provide general environmental parameters monitoring at a number of Greek schools.
- Educational lab kit devices: these are Raspberry Pi-based devices with custom extensions that connect to the infrastructure during educational lab kit activities conducted in GAIA's schools.
- Participatory sensing – Smartphones: we have implemented some additions to the system that allow end-users to upload to the system measurements from the integrated sensors of smartphones, such as luminosity.

At M24, the following components are deployed and available to third-party applications access through APIs:

- *Authentication and Authorization Infrastructure*: it provides authentication, authorization and role management services to GAIA software components (WP2 and WP3) for secured access.
- *Acquisition*: it deals with the heterogeneity of the sensors and API exposed by the different proprietary platforms with the aim of making data uniform before sending them to the GAIA infrastructure.
- *Storage*: this module deals with both the storage of the data in a database supporting time-range queries and the computation of real time statistics of the measurements (e.g., average, maximum, minimum).
- *Analytics*: it processes data retrieved from the Data Storage and the Building Knowledge Base and exposes on-demand aggregation- and statistical-derived information through REST APIs.
- *Recommendation engine*: it generates recommendations for promoting energy saving behaviours, based on the analysis of data made available by the Data Storage and Analytics blocks.
- *Building knowledge base*: it stores all the useful information about the school buildings (e.g., maximum number of hosted people, area, volume, schedules, type of heating and cooling systems, etc.).

References

[CONTABO] https://contabo.com

[DOCKER] https://www.docker.com

[ELK] https://www.elastic.co/solutions/logging

[ENGINE] http://gaia-project.eu/index.php/en/rules-engine-documentation/

[ETCD] https://coreos.com/etcd

[GAIA1.1] GAIA Consortium, D1.1 GAIA Design

[GAIA2.1] GAIA Consortium, D2.1 Initial Infrastructure Software

[GAIA3.1] GAIA Consortium, D3.1 Application Prototypes

[GAIA3.2] GAIA Consortium, D3.2 – Applications Initial Release

[GAIA3.3] GAIA Consortium, D3.3 – Applications Final Release

[Mailgun] Mailgun Web site, https://www.mailgun.com

[OAuth2] OAuth 2 Project. https://oauth.net/2/