IoTsens Platform

May 2016

# Introduction to the IoTsens API

# Table of Contents

# 1. Introduction

The aim of this document is to provide an overview to the main services and functionalities offered by the IoTsens platform API both through HTTP REST microservices and from a standalone Java client (SDK)

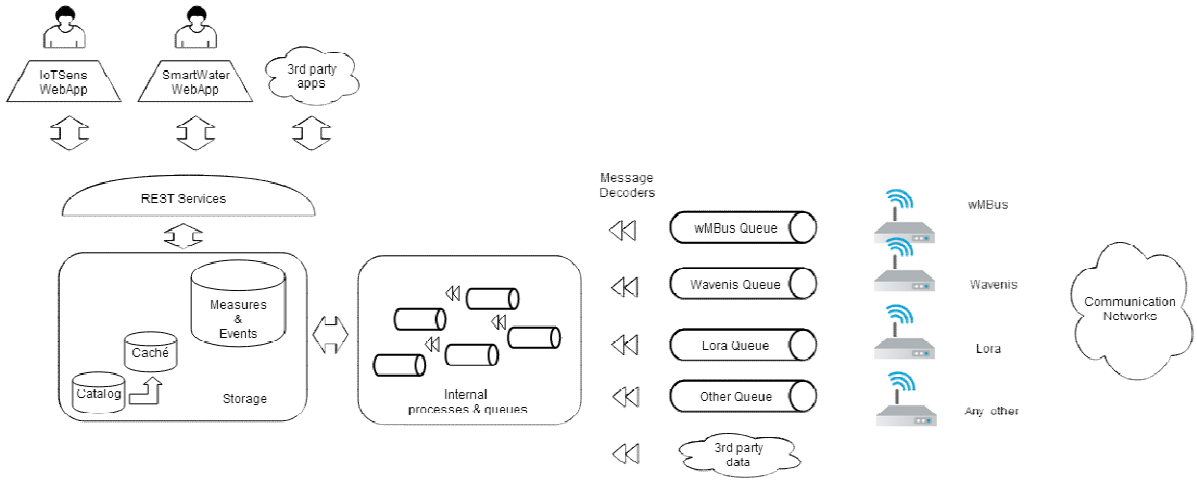
# 2. IoTsens Platform at a glance

The IoTsens platform is built upon a RESTful multi-tier Java architecture, which fully integrates with multiple sensing and communications networks as well as with a wide range of devices (sensors, sensing hubs, communication gateways).

The main components of the platform are:

- A Web Application (https://iotsens.grupogimeno.com ) that allows to use all functionalities of the platform to the users in a convenient and user-friendly way . The platform contains other Web Applications (smarwater, iotindustrial) targeted at specific business sectors.
- A set of RESTful Services (accessible via HTTP) that contains all the functionality offered by the platform.
- Storage component: The platform stores the data to a relational database (MySQL) and a BigData Hadoop (ElasticSearch) storage for its high-performance and clusterization capabilities.
- Internal processes & queues: They are the core pipeline to process all the data received from the sensors.
- Decoders: They decode the data from heterogeneous data formats (sent by the sensors) into a robust homogeneous format to be used by the internal processes & queues
- External Queues (Gateways): They are queues where the sensors publish data to be processed by the platform
- Sensors: Wide range of external devices capable of publishing information to the platform.

As a summary, the following diagram depicts the main components of the platform and their dependencies among them:




## 3. Security

All incoming request to the REST services are validated against unauthorized uses.

The requester calculates a token[1] based on the timestamp of the request, a secret word and a requester code (the secret and the code are known by the requester).

In each request, this calculated token is sent to the server alongside with the timestamp. Upon reception of the request, the server is able to calculate the token from the secret word and requester code (both are stored on the server as well) and the timestamp sent by the client. If the token sent by the requester matches with the one calculated in the server, the requester is considered legitimate to use the platform. There is a time window in which the token is valid; this time window is set to one minute.

Note that this mechanism avoids potential man-in-the-middle attack as the attacker would have to be able to calculate the MD5 without knowing the secret word in less than one minute.

[1] The token is the MD5 function of the timestamp + secret Word + requester Code

Besides the token, the requester must send his or her user name in order to be able to retrieve resources available to him or her.

The following table shows the compulsory header fields related with security:

| | |
|---|---|
| IOT-Authorized-User | IoTsens user |
| GG-Requester-Application | IOTFRONT |
| GG-Request-Signature | MD5(Timestamp + Secret Word + Requester Code) |
| GG-Request-Timestamp | Timestamp |

# 4. Resource conceptual model

The following is a conceptual descripcion of the resource model of the services of the platform covered in this document:

- **Sensor**: The resource **Sensor** models any physical device that is capable of sending/receiving information to/from the platform, for example a thermometer, watermeter…

- **Property:** Sensors have a number of **Properties** which define *static* settings of them, for example the Address where are located, physical size, calibration settings, etc…

- **Variable**: Sensors have a number of **Variables** which define what measure the sensor in time in term of physical or logical units, for example flow, water level, liter_per_minute, etc…

- **Category**: One sensor belongs to one **Category**, which is the "type" or "class" of the sensor, for example water meter, electricity meters, etc…

- **Template**: One sensor could be defined as derived from another one wich acts as template (in terms of variables and properties).

- **Measure**: It is the value of one variable of one sensor at a certain instant (timestamp)

- **RangeMeasure**: Given a set of measures of a sensor in an interval of time, a Range Measure is the result of applying a mathematical operation to all those values.

  - The allowed intervals of time are:

    - SECONDS
    - MINUTES
    - HOURS

- DAYS
- WEEKS
- MONTHS
- YEARS

➢ The allowed operations are:

- MAX
- MIN
- SUM
- COUNT
- AVERAGE
- LAST
- DIFFERENCE_PREVIOUS
- TIME_ON_CONDITION

- **Event**: It is a defined action when some condition happens, for example Sensor Failure, Inactivity events (too much time without receiving information from the sensor), etc…

## 5. Java Client (SDK)

The platform offers a Java Client that acts a Java facade of the Rest services. The main class is `com.iotsens.sdk.IoTSensApiClient` which contains the actual methods to invoke to the REST services. The following is a snippet of how to initialize this class:

```
public static final String APPLICATION_ID = "APPID"; // must be proper application
identifier
public static final String SECRET = "SECRETWORD"; // must be proper secret
public static final String DEFAULT_USER = "USER.NAME"; // must be a proper user


IoTSensApiClient apiClient = aIoTSensClient()
             .withApplication(APPLICATION_ID)
             .withSecret(SECRET)
             .withDefaultUser(DEFAULT_USER)
             .build();

/*
at this point apiClient can invoke all its methods (see details in the examples of
the services)

These methods receives/returns java beans that act as wrappers of the
request/response parameters of the REST services

Some of these Java Beans can be built using a Builder pattern (see details in the
examples of the services) */
```

# 6. REST services

IoTsens platform endpoints are located in http://api.iotsens.com/v1/

Among all the services offered by the platform, only the ones related to Sensors, Events and Measures are covered in this document.

## 3.1 Sensors

### List of Sensors → GET /sensors

Returns a list of sensors. They can be filtered by different criteria, including geographical location, the contents of any of its properties, their category or their template.

Coordinates parameters are used to define a geographic quadrant

| PARAMETERS | | | | |
|---|---|---|---|---|
| **Name** | **Located in** | **Description** | **Required** | **Schema** |
| north | query | North latitude in degrees [-180, 180] | No | number |
| south | query | South latitude in degrees [-180, 180] | No | number |
| east | query | East longitude in degrees [-180, 180] | No | number |
| west | query | West longitude in degrees [-180, 180] | No | number |
| query | query | A query string to filter the sensors according their unique identifiers or the value of any of its properties, for example if query=A00, all sensors or properties having A00 in its name (any position) will match | No | string |
| categoryId | query | The identifier of the category that must have the sensors returned | No | number |
| templateId | query | The identifier of the template that must have the sensors returned | No | number |
| limit | query | The maximum numbers of items returned | No | number |
| offset | query | The number of the first item returned | No | number |

| RESPONSES | | |
|---|---|---|
| **Code** | **Description** | **Schema (see Annex)** |
| 200 | A list of sensors basic info and the total number of items | SensorBasic |

*Example:*

Retrieve all sensors matching the conditions:

- belong to category=6 (watchmeter))
- are located in geographical coordinates (`0.085239, 40.887784, 0.082239, 39.887784)`
- have the word 'A00' in its uniqueId
- belong to templateId=18 (watchmeter)

HTTP:

**GET →**
`http://api.iotsens.com/v1/sensors?categoryId=6&north=40.887784&south=39.887784&east=0.085239&west=0.082239&query=A00&templateId=18`

JSON Response:


```
{
  "success": true,
  "data": [
    {
      "id": 1969,
      "sourceTemplate": {
        "id": 18,
        "templateName": "WatchMeter"
      },
      "category": {
        "id": 6,
        "listingOrder": 170,
        "color": "",
        "name": "Contador suministro"
      },
      "sourceTemplateVersion": 2,
      "enable": true,
      "longitude": 0.083239,
      "templateEntity": false,
      "latitude": 40.219082,
      "uniqueId": "A0020000000601"
    }
  ],
  "total": 1
}
```

Java:

```
/*
   It is assumed that apiClient is properly created

   SensorsRequest is a Java Bean that wraps the request API parameters

   SensorRequest is built using the static builder method:

   com.iotsens.sdk.sensors.SensorsRequestBuilder.aSensorRequest

   this builder takes no parameter


   SensorBasic is a Java Bean that wraps the service response

*/

SensorsRequest sensorsRequest = aSensorRequest()
             .withNorth(40.887784)
             .withWest(0.082239)
             .withSouth(39.887784)
             .withEast(0.085239)
             .withCategoryId(6)
             .withTemplateId(18)
             .withQuery("A000")
             .build();

for (SensorBasic sensorBasic :  apiClient.getSensors(sensorsRequest)) {
      System.out.println("Sensor = " + sensorBasic.toString());
}
```

### Basic data for a list of sensors → POST /sensors/info

Returns the basic data and the properties of a list of sensors (providing the sensorId list as independent POST parameter). This method can be used for external applications which only know about the unique identifier of one sensor.

| PARAMETERS | | | | |
|---|---|---|---|---|
| **Name** | **Located in** | **Description** | **Required** | **Schema** |
| sensorId | formData | A list of sensor unique identifiers | Yes | string |

| RESPONSES | | |
|---|---|---|
| **Code** | **Description** | **Schema (see Annex)** |
| 200 | A list of sensors basic info | SensorBasicWithProperties |

*Example:*

Retrieve basic data (including properties) of sensor with unique id=14F234233 and A0020000000100

HTTP:

**POST** → http://api.iotsens.com/v1/sensors/info

Parameters:

| | |
|---|---|
| sensorId | 14F234224 |
| sensorId | A0020000000100 |

JSON Response


```
{
  "success": true,
  "data": [
    {
      "id": 1483,
      "sourceTemplate": {
        "id": 27,
        "templateName": "Contador wMBus Elster"
      },
      "category": {
        "id": 1,
        "listingOrder": 30,
        "color": "",
        "name": "Contador agua doméstico"
      },
```

```
      "enable": true,
      "longitude": -0.025011,
      "templateEntity": false,
      "latitude": 39.985072,
      "properties": [
        {
          "id": 4347,
          "unit": "",
          "sensorId": "1483",
          "name": "IDENTIFICADOR_RADIO",
          "displayOrder": 10,
          "writeable": true,
          "remote": false,
          "lastValue": "14234233"
        },
        {
          "id": 4348,
          "unit": "",
          "sensorId": "1483",
          "name": "POLIZA",
          "displayOrder": 10,
          "writeable": true,
          "remote": false,
          "lastValue": "1039 113555"
        },
        {
          "id": 4349,
          "unit": "",
          "sensorId": "1483",
          "name": "DIRECCION",
          "displayOrder": 10,
          "writeable": true,
          "remote": false,
          "lastValue": "CALLE VICENTE GIMENO MICHAVILA, 5 - 2 C"
        }
      ],
      "uniqueId": "14F234233"
    },
    {
      "id": 1985,
      "sourceTemplate": {
        "id": 19,
        "templateName": "Contador WatchMeter"
      },
      "category": {
        "id": 6,
        "listingOrder": 170,
        "color": "",
        "name": "Contador suministro"
      },
      "sourceTemplateVersion": 1,
      "enable": true,
      "longitude": -0.034935,
      "templateEntity": false,
      "latitude": 39.987719,
      "properties": [
        {
          "id": 5830,
          "unit": "",
          "sensorId": "1985",
          "name": "FACTOR_K",
          "displayOrder": 10,
          "writeable": true,
```

```
          "remote": false,
          "lastValue": ""
        },
        {
          "id": 5831,
          "unit": "",
          "sensorId": "1985",
          "name": "LITROS_POR_VUELTA_CONTADOR",
          "displayOrder": 10,
          "writeable": true,
          "remote": false,
          "lastValue": ""
        },
        {
          "id": 5832,
          "unit": "",
          "sensorId": "1985",
          "name": "LITROS_INICIALES",
          "displayOrder": 10,
          "writeable": true,
          "remote": false,
          "lastValue": ""
        }
      ],
      "uniqueId": "A0020000000100"
    }
  ]
}
```


**Java:**

```
/*
   It is assumed that apiClient is properly created

   SensorBasicWithProperties is a Java Bean that wraps the service response

*/

List<String> listOfSensorIds = new ArrayList<String>();

listOfSensorIds.add("14F234224");
listOfSensorIds.add("A0020000000100");

for (SensorBasicWithProperties sensorBasicWithProperties :
apiClient.getSensors(listOfSensorIds)) {

      System.out.println("Sensor = " +    sensorBasicWithProperties.toString());
}
```

### Complete data for one sensor → GET /sensors/{sensorId}

Returns all the data about one sensor, including the complete specification of its variables.

| PARAMETERS | | | | |
|---|---|---|---|---|
| **Name** | **Located in** | **Description** | **Required** | **Schema** |
| sensorId | path | The unique identifier of the sensor | Yes | string |

| RESPONSES | | |
|---|---|---|
| **Code** | **Description** | **Schema (see Annex)** |
| 200 | The complete specification of the sensor | SensorWithPropertiesAndVariables |

*Example:*

Retrieve Information (including variables) of sensor id=14F234224

HTTP:

**GET** → `http://api.iotsens.com/v1/sensors/14F234224`

JSON Response


```
{
  "success": true,
  "msg": null,
  "data": {
    "id": 250,
    "uniqueId": "14F234224",
    "variables": [
      {
        "id": 1203,
        "name": "FORWARD_FLOW",
        "type": null,
        "defaultGraphType": "BARS",
        "unit": "m3",
        "needsPolling": false,
        "pollingGap": null,
        "maxInactivitySeconds": null,
        "description": "Volumen consumido",
        "measureConversors": [],
        "eventGenerators": [],
        "derivedFrom": {
          "id": 1932,
          "name": "CURRENT_VALUE",
          "type": null,
          "defaultGraphType": "LINE",
```

```
          "unit": "m3",
          "needsPolling": false,
          "pollingGap": null,
          "maxInactivitySeconds": 172800,
          "description": "Lectura",
          "measureConversors": [],
          "eventGenerators": [],
          "derivedFrom": null,
          "rangeSummaryMinTimeunit": null,
          "summaryOperation": null,
          "displayOrder": 10,
          "roundingDecimals": 2,
          "derived": false,
          "rangesVariable": false,
          "summarizer": {}
        },
        "rangeSummaryMinTimeunit": "DAYS",
        "summaryOperation": "DIFFERENCE_PREVIOUS",
        "displayOrder": 10,
        "roundingDecimals": 2,
        "derived": true,
        "rangesVariable": true,
        "summarizer": {}
      },
….
```


Java:

```
/*
   It is assumed that apiClient is properly created


   SensorWithPropertiesAndVariables is a Java Bean that wraps the service response

*/

SensorWithPropertiesAndVariables sensor = apiClient.getSensor("14F234224");

System.out.println("Sensor complete data = " + sensor);
```

## 3.2 Measures

### Get the measures for the variable by criteria → GET /sensors/{sensorId}/variables/{variableName}/measures

Return the measures for the given variable following some criteria. If the from and until parameters are used, the measures will be included in a range param. If the dayParam is defined, the service will return a list with one measure in the specified day. If no measure is found the specified day, the daysBeforeMargin and daysAfterMargin can be used to increase the date range width. If neither date range nor day are specified, the service will return the oldest measures for the variable. The number of oldest measures that are returned can be constrained using the limit parameter.

NOTE: the parameters from and until take precedence over dayParam, which means that dayParam will be ignored if from and until are populated in the query

| PARAMETERS | | | | |
|---|---|---|---|---|
| **Name** | **Located in** | **Description** | **Required** | **Schema** |
| sensorId | path | The unique identifier of the sensor | Yes | string |
| variableName | path | The name of the variable | Yes | string |
| from | query | The begining of the time period in format yyyyMMdd or yyyyMMddHHmmss | No | string |
| until | query | The end of the time period in format yyyyMMdd or yyyyMMddHHmmss | No | string |
| limit | query | The maximum number of measures to return | No | number |
| dayParam | query | This param is used if only one measure for one day is required (the oldest). The day is specified in this para using the yyyyMMdd format | No | string |
| daysBeforeMargin | query | If the dayParam is defined, this parameter can be used to define the number of days before the specified day in case no measures are found in the specified day | No | number |
| daysAfterMargin | query | If the dayParam is defined, this parameter can be used to define the number of days after the specified day in case no measures are found in the specified day | No | number |

| RESPONSES | | |
|---|---|---|
| **Code** | **Description** | **Schema (see Annex)** |
| 200 | The measures for the variable following the defined criteria | Measure |

*Example 1:*

Retrieve the measures of values of variable TEMP of sensor with unique Id=TEMP01 and from the dates 18/05/2016 until 20/05/2016

**HTTP:**

**GET →**
http://api.iotsens.com/v1/sensors/TEMP01/variables/TEMP/measures?from=20160514&until=20160516

JSON Response


```
{
  "success": true,
  "data": [
    {
      "timestamp": "16/05/2016 23:58:28",
      "variableName": "TEMP",
      "sensorId": "TEMP01",
      "value": "16.01"
    },
    {
      "timestamp": "16/05/2016 23:53:28",
      "variableName": "TEMP",
      "sensorId": "TEMP01",
      "value": "16.04"
    },
    {
      "timestamp": "16/05/2016 23:48:26",
      "variableName": "TEMP",
      "sensorId": "TEMP01",
      "value": "16.12"
    },
….
    {
      "timestamp": "14/05/2016 00:09:53",
      "variableName": "TEMP",
      "sensorId": "TEMP01",
      "value": "16.83"
    },
    {
      "timestamp": "14/05/2016 00:04:50",
      "variableName": "TEMP",
      "sensorId": "TEMP01",
      "value": "16.68"
    }
  ]
}
```

**Java:**

```
/*
   It is assumed that apiClient is properly created

   MeasuresRequest is a Java Bean that wraps the request API parameters

   MeasuresRequest is built using the static builder method:

   com.iotsens.sdk.measures.MeasuresRequestBuilder.aMeasureRequest

  this builder takes the sensorId and the variableName as parameters


  Measure is a Java Bean that wraps the service response

*/

MeasuresRequest request = MeasuresRequestBuilder.aMeasureRequest("TEMP01", "TEMP")
             .withFrom("20160518")
             .withUntil("20160520")
             .build();

List<Measure> measures = apiClient.getMeasures(request);
for (Measure measure: apiClient.getMeasures(request)) {
      System.out.println("Measure = " + measure.toString());
}
```

### *Example 2*

Retrieve the oldest measure of variable TEMP on May, 16thy 2016 of the sensor with unique Id=TEMP01 with range of 2 days (before/after) in case there is not a measure on May, 16thy 2016

HTTP:

**GET →**
http://api.iotsens.com/v1/sensors/TEMP01/variables/TEMP/measures?dayParam=20160516&daysBeforeMargin=2&daysAfterMargin=2

JSON Response


```
{
  "success": true,
  "data": [
    {
      "timestamp": "16/05/2016 23:58:28",
      "variableName": "TEMP",
      "sensorId": "TEMP01",
      "value": "16.01"
    }
  ]
}
```

Java:

```
/*
   It is assumed that apiClient is properly created

   MeasuresRequest is a Java Bean that wraps the request API parameters

   MeasuresRequest is built using the static builder method:

   com.iotsens.sdk.measures.MeasuresRequestBuilder.aMeasureRequest

  this builder takes the sensorId and the variableName as parameters



   Measure is a Java Bean that wraps the service response

*/

MeasuresRequest request = MeasuresRequestBuilder.aMeasureRequest("TEMP01", "TEMP")
             .withDayParam("20160518")
             .withDaysBeforeMargin(2)
             .withDaysAfterMargin(2)
             .build();

for (Measure measure: apiClient.getMeasures(request)) {
      System.out.println("Measure = " + measure.toString());
}
```

### Get the summarized measures for the variable in a time range → GET /sensors/{sensorId}/variables/{variablesName}/rangemeasures

Return the measures for the variable in the defined time period until 3000 at most. The measures represent a value in a defined range of time. All the variables measures in that range of time has been summarized using a given operation ( MAX, MIN, SUM, COUNT, AVERAGE, LAST, TIME_ON_CONDITION , DIFFERENCE_PREVIOUS,)

| PARAMETERS | | | | |
|---|---|---|---|---|
| **Name** | **Located in** | **Description** | **Required** | **Schema** |
| sensorId | path | The unique identifier of the sensor | Yes | String |
| variableName | path | The name of the variable | Yes | String |
| from | query | The begining of the time range in format yyyyMMdd or yyyyMMddHHmmss | No | String |
| until | query | The end of the time range in format yyyyMMdd or yyyyMMddHHmmss | No | String |
| rangeUnit | query | The kind of range to summarize the measures | Yes | String |
| unit | query | The number of range units for summarizing the measures. For instance 3 MONTHS. Allowed values are: SECONDS, MINUTES, HOURS, DAYS, WEEKS, MONTHS, YEARS | No | String |
| summaryOperation | query | The operation to apply to the measures in each time range to produce the summarized measures. Allowed values are: MAX, MIN, SUM, COUNT, AVERAGE, LAST, DIFFERENCE_PREVIOUS, TIME_ON_CONDITION | Yes | String |

| RESPONSES | | |
|---|---|---|
| **Code** | **Description** | **Schema (see Annex)** |
| 200 | The summarized measures for the variable in the defined period | SummarizedMeasure |

*Example:*

Retrieve the range values of variable TEMP of sensor with unique id=TEMP01. The value of the Range is the Minimum value each 30 minutes.

HTTP:

**GET →**

http://api.iotsens.com/v1/sensors/TEMP01/variables/TEMP/rangemeasures?summaryOperation=MIN&rangeUnit=MINUTES&unit=30&from=20160514&until=20160514

JSON Response


```
{
  "success": true,
  "data": [
    {
      "timestamp": "14/05/2016 02:00:00",
      "variableName": "TEMP",
      "summaryTimeUnit": "MINUTES",
      "sensorId": "TEMP01",
      "value": "16.29",
      "rawValue": "16.290000915527344"
    },
    {
      "timestamp": "14/05/2016 02:30:00",
      "variableName": "TEMP",
      "summaryTimeUnit": "MINUTES",
      "sensorId": "TEMP01",
      "value": "16.02",
      "rawValue": "16.020000457763672"
    },
    {
      "timestamp": "14/05/2016 03:30:00",
      "variableName": "TEMP",
      "summaryTimeUnit": "MINUTES",
      "sensorId": "TEMP01",
      "value": "15.68",
      "rawValue": "15.680000305175781"
    },
…..
```

Java:

```
/*
   It is assumed that apiClient is properly created

   RangeMeasuresRequest is a Java Bean that wraps the request API parameters

   MeasuresRequest is built using the static builder method:

   com.iotsens.sdk.measures.RangeMeasureRequestBuilder.aRangeMeasureRequest

   this builder takes as parameters:

       -the sensorId
       -variableName
       -unit
       -summaryOperation


   Measure is a Java Bean that wraps the service response

*/

RangeMeasuresRequest rangeMeasuresRequest =
RangeMeasuresRequestBuilder.aRangeMeasureRequest("TEMP01", "TEMP",
SummaryTimeUnit.MINUTES, SummaryOperationType.MIN)
             .withFrom("20160514")
             .withUntil("20160520")
             .withUnit(30)
             .build();

for (RangeMeasure rangeMeasure: apiClient.getRangeMeasures(rangeMeasuresRequest)) {
      System.out.println("RangeMeasure = " + rangeMeasure.toString());
}
```

### Get the measures for multiple sensors in a time range → POST /measures

Return the measures for multiple sensors in the defined time period.

| PARAMETERS | | | | |
|---|---|---|---|---|
| **Name** | **Located in** | **Description** | **Required** | **Schema** |
| sensorId | formData | A list of unique identifiers of the sensors that generated the events | Yes | String |
| variableName | formData | The name of the variable | Yes | String |
| from | formData | The begining of the time range in format yyyyMMdd or yyyyMMddHHmmss | No | String |
| until | formData | The end of the time range in format yyyyMMdd or yyyyMMddHHmmss | No | String |
| limit | formData | limit of number of measures retrieved | Yes | Number |

| RESPONSES | | |
|---|---|---|
| **Code** | **Description** | **Schema (see Annex)** |
| 200 | The measures for the variable in the defined period | Measure |

*Example:*

Retrieve all measures with unique Id = TEMP01 and TEMP02 and variable Name = TEMP on May 10th 2016

HTTP:

**POST** →

http://api.iotsens.com/v1/measures

Parameters:

| | |
|---|---|
| sensorId | TEMP01 |
| sensorId | TEMP02 |
| variableName | TEMP |
| From | 20160510 |
| Until | 20160510 |
| Limit | 40 |

JSON Response


```
{
  "success": true,
  "data": [
    {
      "timestamp": "10/05/2016 23:58:01",
      "variableName": "TEMP",
      "sensorId": "TEMP01",
      "value": "13.65"
    },
    {
      "timestamp": "10/05/2016 23:58:01",
      "variableName": "TEMP",
      "sensorId": "TEMP02",
      "value": "13.60"
    },
    {
      "timestamp": "10/05/2016 23:53:01",
      "variableName": "TEMP",
      "sensorId": "TEMP01",
      "value": "13.67"
    },
    {
      "timestamp": "10/05/2016 23:53:01",
      "variableName": "TEMP",
      "sensorId": "TEMP02",
      "value": "13.67"
    },
    {
      "timestamp": "10/05/2016 23:48:00",
      "variableName": "TEMP",
      "sensorId": "TEMP01",
      "value": "13.65"
    },
    {
      "timestamp": "10/05/2016 23:48:00",
      "variableName": "TEMP",
      "sensorId": "TEMP02",
      "value": "13.68"
    },
…..
```

Java:

```
/*
   It is assumed that apiClient is properly created

   MultiSensorMeasuresRequest is a Java Bean that wraps the request API parameters

   MeasuresRequest is built using the static builder method:

com.iotsens.sdk.measures.MultiSensorMeasuresRequestBuilder.aMultiSensorMeasureRequest

   This builder takes as parameters:
       -the sensorId
       -variableName
       -limit



   Measure is a Java Bean that wraps the service response

*/

MultiSensorMeasuresRequest multiSensorRequest = aMultiSensorMeasureRequest("TEMP01",
"TEMP", 40)
             .addSensorId("TEMP02")
             .withFrom("20160510")
             .withUntil("20160510")
             .build();

List<Measure> measures = apiClient.getMeasures(request);
for (Measure measure: apiClient.getMeasures(multiSensorRequest)) {
      System.out.println("Measure = " + measure.toString());
}
```

### Get the summarized measures for multiple sensors in a time range → POST /rangemeasures

Return the summarized measures for multiple sensors in the defined time period.

| PARAMETERS | | | | |
|---|---|---|---|---|
| **Name** | **Located in** | **Description** | **Required** | **Schema** |
| sensorId | formData | A list of unique identifiers of the sensors that generated the events | Yes | String |
| variableName | formData | The name of the variable | Yes | String |
| from | formData | The begining of the time range in format yyyyMMdd or yyyyMMddHHmmss | No | String |
| until | formData | The end of the time range in format yyyyMMdd or yyyyMMddHHmmss | No | String |
| rangeUnit | query | The kind of range to summarize the measures | Yes | String |
| unit | query | The number of range units for summarizing the measures. For instance 3 MONTHS | No | Number |
| summaryOperation | query | The operation to apply to the measures in each time range to produce the summarized measures. | Yes | String |
| limit | formData | limit of number of measures retrieved | No | Number |

| RESPONSES | | |
|---|---|---|
| **Code** | **Description** | **Schema (see Annex)** |
| 200 | The summarized measures for the variable in the defined period | SummarizedMeasure |

***Example:***

Retrieve all Ranged measures summarizing each 30 minutes by MAX value of the sensorId with uniqueId 14F234224 and variable Name = CURRENT_VALUE on May 10th 2016.

<u>HTTP:</u>

**POST →**

`http://api.iotsens.com/v1/rangemeasures`

Post Parameters:

| | |
|---|---|
| sensorId | TEMP01 |
| sensorId | TEMP02 |
| variableName | TEMP |
| From | 20160510 |
| Until | 20160510 |
| rangeUnit | MINUTES |
| summaryOperation | MAX |
| Unit | 30 |

JSON Response


```
{
  "success": true,
  "data": [
    {
      "timestamp": "10/05/2016 02:00:00",
      "variableName": "TEMP",
      "summaryTimeUnit": "MINUTES",
      "sensorId": "TEMP01",
      "value": "13.24",
      "rawValue": "13.239999771118164"
    },
    {
      "timestamp": "10/05/2016 02:30:00",
      "variableName": "TEMP",
      "summaryTimeUnit": "MINUTES",
      "sensorId": "TEMP01",
      "value": "12.90",
      "rawValue": "12.899999618530273"
    },
    {
      "timestamp": "10/05/2016 03:00:00",
      "variableName": "TEMP",
      "summaryTimeUnit": "MINUTES",
      "sensorId": "TEMP01",
      "value": "13.20",
      "rawValue": "13.199999809265137"
    },
    {
      "timestamp": "10/05/2016 03:30:00",
      "variableName": "TEMP",
      "summaryTimeUnit": "MINUTES",
      "sensorId": "TEMP01",
      "value": "13.20",
      "rawValue": "13.199999809265137"
    },
….
```

Java:

```
/*
   It is assumed that apiClient is properly created

   RangeMeasuresRequest is a Java Bean that wraps the request API parameters

   RangeMeasuresRequest is built using the static builder method:

   com.iotsens.sdk.measures.RangeMeasureRequestBuilder.aRangeMeasureRequest

   this builder takes as parameters:

       -the sensorId
       -variableName
       -unit
       -summaryOperation



   RangeMeasure is a Java Bean that wraps the service response

*/

RangeMeasuresRequest multiSensorRangeMeasuresRequest =
RangeMeasuresRequestBuilder.aRangeMeasureRequest("TEMP01", "TEMP",
SummaryTimeUnit.MINUTES, SummaryOperationType.MAX)
             .addSensorId("TEMP02")
             .withFrom("20160510")
             .withUntil("20160510")
             .withUnit(30)
             .build();

for (RangeMeasure rangeMeasure: apiClient.getRangeMeasures(rangeMeasuresRequest)) {
      System.out.println("RangeMeasure = " + rangeMeasure.toString());
}
```

## 3.2 Events

### Get events for the sensor → GET /sensors/{sensorId}/events

Search events for the sensor filtering by some properties

| PARAMETERS | | | | |
|---|---|---|---|---|
| **Name** | **Located in** | **Description** | **Required** | **Schema** |
| sensorId | path | The unique identifier of the sensor | Yes | String |
| variableId | query | The variable name to which the event belongs. An empty value will return all variables in sensor | No | String |
| from | query | The activation date of the returned events must be after the from parameter | No | String (date) |
| until | query | The activation date of the returned events must be before the from parameter | No | String (date) |
| name | query | The event name must contain this param | No | String |
| activation | query | The value 'ACTIVE' will return all events in an active state, which includes ACTIVE, PENDING_ACK and ACKNOWLEDGED. The value 'INACTIVE' will return all evente in INACTIVE state. Empty param or 'ALL' value will be ignored. | No | String |
| ackState | query | The state of the returned events must match this param. If empty, all states will be included.<br><br>Could be one of the following:<br><br>• NOT_REQUIRED<br>• PENDING_ACK<br>• ACKNOWLEDGED; | No | String |
| eventType | query | The state of the returned events must match this param. If empty, all states will be included. | No | String |
| offset | query | The number of events that must be ommited in the result, for paginating purposes | No | Number |
| limit | query | The maximum number of events that must be returned, for paginating purposes | No | Number |

| RESPONSES | | |
|---|---|---|
| **Code** | **Description** | **Schema (see Annex)** |
| 200 | The events list for the sensor which satisfy the filtering criteria | Event |

*Example:*

Retrieve the events of the sensor with unique id=METEOLINK002 and matching the following conditions:

- belong to variable "TEMP"
- from February 20th 2016 until May 5th 2016
- acknowledged state = `ACKNOWLEDGED`
- activation = `ACTIVE`
- name=`INACTIVITY_EVENT`
- eventType=`WARNING`

HTTP:

**GET** →
`http://api.iotsens.com/v1/sensors/METEOLINK002/events/?variableName=TEMP&from=20160220&until=20160520&ackState=ACKNOWLEDGED&activation=ACTIVE&name=INACTIVITY_EVENT&eventType=WARNING`

JSON Response:


```
{
  "success": true,
  "data": [
    {
      "timestamp": "20/02/2016 07:22:24",
      "id": 24028,
      "variableName": "TEMP",
      "name": "INACTIVITY_EVENT",
      "sensorId": "METEOLINK002",
      "data": "Se ha superado el tiempo máximo sin recibir medidas (610 segundos) en
variable TEMP",
      "active": true,
      "type": "WARNING",
      "ackState": "ACKNOWLEDGED"
    }
  ]
}
```

Java:

```
/*
   It is assumed that apiClient is properly created

   EventsRequest is a Java Bean that wraps the request API parameters

   EventsRequest is built using the static builder method:

   com.iotsens.sdk.events.EventsRequestBuilder.aEventRequest

   this builder takes no parameters


   Event is a Java Bean that wraps the service response

*/

EventsRequest eventsRequest = EventsRequestBuilder.aEventRequest()
             .addSensorId("METEOLINK002")
             .withVariableName("TEMP")
             .withFrom("20160220")
             .withUntil("20160505")
             .withAckState(AckState.ACKNOWLEDGED)
             .withActivation(ActivationState.ACTIVE)
             .withName("INACTIVITY_EVENT")
             .withEventType(EventType.WARNING)
             .build();

for (Event event: apiClient.getEvents(eventsRequest)) {
      System.out.println(event);
}
```

### Get active events for the sensor → GET /sensors/{sensorId}/events/active

Return the sensor events that are currently active

| PARAMETERS | | | | |
|---|---|---|---|---|
| **Name** | **Located in** | **Description** | **Required** | **Schema** |
| sensorId | path | The unique identifier of the sensor | Yes | String |

| RESPONSES | | |
|---|---|---|
| **Code** | **Description** | **Schema (see Annex)** |
| 200 | The sensor events that are currently active | Event |

***Example:***

Retrieve all the active events for sensor with sensorId=METEOLINK002

HTTP:

**GET** → http://api.iotsens.com/v1/sensors/METEOLINK002/events/active

JSON Response:


```
{
  "success": true,
  "data": [
    {
      "timestamp": "21/02/2016 06:40:59",
      "id": 24325,
      "variableName": "TEMP",
      "name": "INACTIVITY_EVENT",
      "sensorId": "METEOLINK002",
      "data": "Se ha superado el tiempo máximo sin recibir medidas (610 segundos) en
variable TEMP",
      "active": true,
      "type": "INFO",
      "ackState": "NOT_REQUIRED"
    },
    {
      "timestamp": "20/02/2016 07:22:24",
      "id": 24028,
      "variableName": "TEMP",
      "name": "INACTIVITY_EVENT",
      "sensorId": "METEOLINK002",
      "data": "Se ha superado el tiempo máximo sin recibir medidas (610 segundos) en
variable TEMP",
      "active": true,
      "type": "WARNING",
      "ackState": "ACKNOWLEDGED"
    },
```

```
    {
      "timestamp": "20/02/2016 06:17:40",
      "id": 24022,
      "variableName": "TEMP",
      "name": "INACTIVITY_EVENT",
      "sensorId": "METEOLINK002",
      "data": "Se ha superado el tiempo máximo sin recibir medidas (610 segundos) en
variable TEMP",
      "active": true,
      "type": "ALARM",
      "ackState": "PENDING_ACK"
    },
```


Java:

```
/*
   It is assumed that apiClient is properly created

   EventsRequest is a Java Bean that wraps the request API parameters

   EventsRequest is built using the static builder method:

   com.iotsens.sdk.events.EventsRequestBuilder.aEventRequest

   this builder takes no parameters

   Event is a Java Bean that wraps the service response

*/

EventsRequest eventsRequest = EventsRequestBuilder.aEventRequest()
             .addSensorId("METEOLINK002")
             .withActivation(ActivationState.ACTIVE)
             .build();

for (Event event: apiClient.getEvents(eventsRequest)) {
      System.out.println(event);
}
```

### Search events → POST /events

Searches events which match some criteria defined in the parameters, including a list of sensors identifiers to search only the events of those sensors

| PARAMETERS | | | | |
|---|---|---|---|---|
| **Name** | **Located in** | **Description** | **Required** | **Schema** |
| sensorId | formData | A list of unique identifiers of the sensors that generated the searched events | No | string |
| variableId | formData | The variable name to which the event belongs. An empty value will return all variables in sensor | No | string |
| from | formData | The activation date of the returned events must be after the from parameter | No | string (date) |
| until | formData | The activation date of the returned events must be before the from parameter | No | string (date) |
| name | formData | The event name must contain this param | No | string |
| activation | formData | The returned events must be in the selected activation state. Empty param or 'ALL' value will be ignored. | No | string |
| ackstate | formData | The acknowledgement state of the returned events must match this param. If empty, all states will be included. Could be one of the following: NOT_REQUIRED, PENDING_ACK, ACKNOWLEDGED | No | string |
| eventType | formData | The type that must have the returned events. Could be one of the following:<br>• ALARM<br>• WARNING<br>• INFO | No | string |
| offset | formData | The number of events that must be ommited in the result, for paginating purposes | No | number |
| limit | formData | The maximum number of events that must be returned, for paginating purposes | No | number |

| RESPONSES | | |
|---|---|---|
| **Code** | **Description** | **Schema (see Annex)** |
| 200 | The sensor events that satisfy the required criteria | Event |

Searches events which match some criteria defined in the parameters, including a list of sensors identifiers to search only the events of those sensors.

*Example:*

Retrieve all the events from sensors with id 14F234233 and METEOLINK002

HTTP:

**POST →**

`http://api.iotsens.com/v1/events`

Post Parameters:

| | |
|---|---|
| sensorId | METEOLINK002 |
| sensorId | 14F234233 |

JSON Response


```
{
  "success": true,
  "data": [
    {
      "timestamp": "20/02/2016 07:22:24",
      "id": 24028,
      "variableName": "TEMP",
      "name": "INACTIVITY_EVENT",
      "sensorId": "METEOLINK002",
      "data": "Se ha superado el tiempo máximo sin recibir medidas (610 segundos) en
variable TEMP",
      "active": true,
      "type": "WARNING",
      "ackState": "ACKNOWLEDGED"
    }
  ]
}
```


Java:

```
/*
   It is assumed that apiClient is properly created

   EventsRequest is a Java Bean that wraps the request API parameters

   EventsRequest is built using the static builder method:

   com.iotsens.sdk.events.EventsRequestBuilder.aEventRequest

   Event is a Java Bean that wraps the service response
*/

EventsRequest eventsRequest = EventsRequestBuilder.aEventRequest()
             .addSensorId("METEOLINK002")
             .addSensorId("14F234233")
             .build();

for (Event event: apiClient.getEvents(eventsRequest)) {
       System.out.println(event);
}
```

### Number of events in the search → POST /events/count

| PARAMETERS | | | | |
|---|---|---|---|---|
| **Name** | **Located in** | **Description** | **Required** | **Schema** |
| sensorId | formData | A list of unique identifiers of the sensors that generated the searched events | Yes | string |
| sensorVariableId | formData | The variable name to which the event belongs. An empty value will return all variables in sensor | No | string |
| from | formData | The activation date of the returned events must be after the from parameter | No | string (date) |
| until | formData | The activation date of the returned events must be before the from parameter | No | string (date) |
| name | formData | The event name must contain this param | No | string |
| activation | formData | The returned events must be in the selected activation state. Empty param or 'ALL' value will be ignored. | No | string |
| ackstate | formData | The acknowledgement state of the returned events must match this param. If empty, all states will be included. | No | string |
| eventType | formData | The type that must have the returned events | No | string |

| RESPONSES | | |
|---|---|---|
| **Code** | **Description** | **Schema (see Annex)** |
| 200 | The number of sensor events that satisfy the required criteria | number |

Return the number of events which matches some criteria defined in the parameters, including a list of sensors identifiers to search only the events of those sensors. Used for pagination purposes.

*Example:*

Retrieve the number of events belonging to sensors with id=14F234233 and id=METEOLINK002

HTTP:

**POST →**

`http://api.iotsens.com/v1/events/count`

Post Parameters:

| | |
|---|---|
| sensorId | 14F234233 |
| sensorId | METEOLINK002 |

JSON Response

```
{
  "success": true,
  "data": {
    "value": "137"
  }
}….
```

Java:

```
/*
   It is assumed that apiClient is properly created

   EventsRequest is a Java Bean that wraps the request API parameters

   EventsRequest is built using the static builder method:

   com.iotsens.sdk.events.EventsRequestBuilder.aEventRequest

   this builder takes no parameters

   Event is a Java Bean that wraps the service response

*/

EventsRequest eventsRequest = EventsRequestBuilder.aEventRequest()
             .addSensorId("TEMP01")
             .addSensorId("TEMP02")
             .withVariableName("TEMP")
             .withFrom("20160518")
             .build();

List<Event> events = apiClient.getEvents(eventsRequest);

for (Event event : events) {
      System.out.println("event = " + event);
}
Integer count = apiClient.countEvents(eventsRequest);
System.out.println("count = " + count);
```

## 7. Errors

Broadly speaking, if a service can not finish successfully due to an error or exception, that fact is reflected on the response with the attribute `success: false` and the corresponding exception className . Optionally, a message text (msg) could be provided.

Example:

```
{
  "success": false,
  "msg": "message",
  "data": {
    "exceptionClassName": "com.grupogimeno.maya.iot.exceptions.ObjectNotFoundException"
  },
  "total": 0
}
```

The exception class Name could be one of the following:

- `com.grupogimeno.maya.iot.exceptions.ObjectNotFoundException ->` When the expected message to be retrieved or used does not exist on the system.

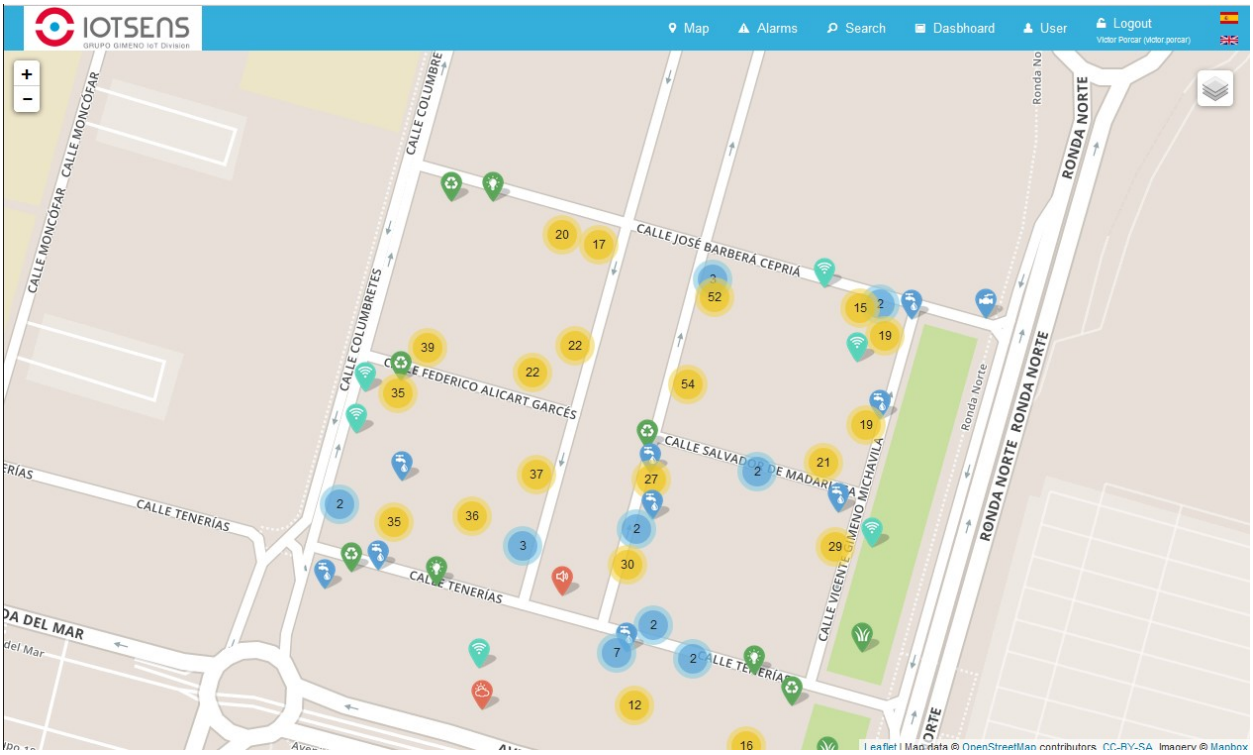
- `com.grupogimeno.maya.exceptions.UserCanNotRequestException ->` When the user is not allowed to access to the requested resource.

- `com.grupogimeno.maya.exceptions.SensorExistsException ->` When the requested or searched sensor does not exist

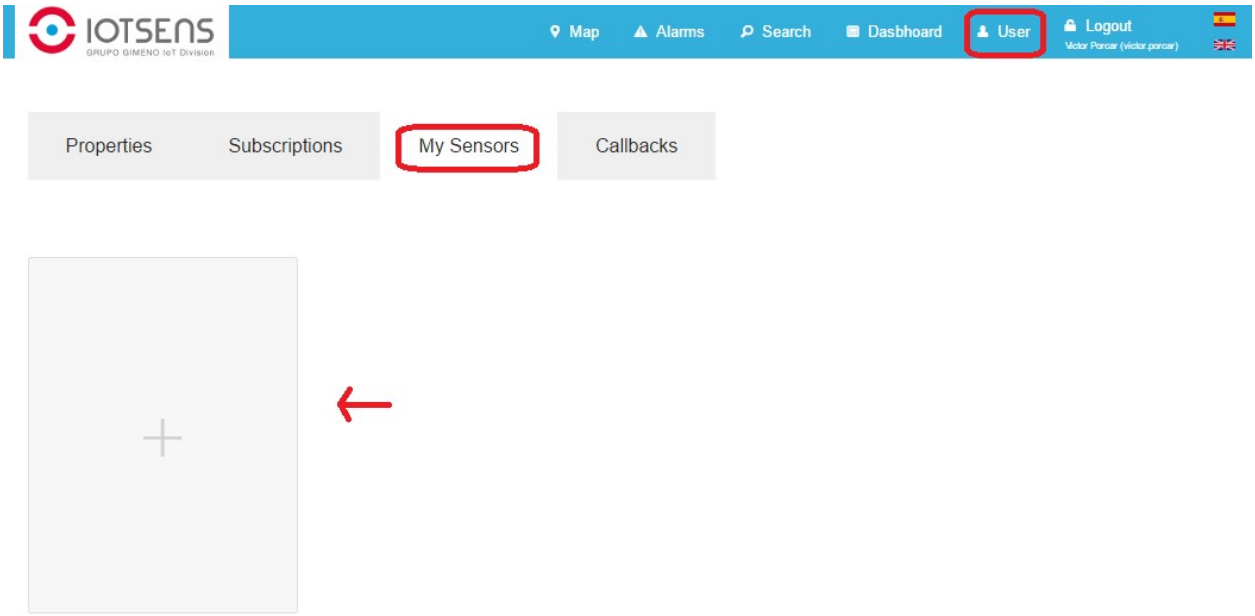
# 8. Use cases

## Create a sensor using the Web application

Log in the http://iotsens.grupogimeno.com with your credentials, it will appear the main page of the Web Application




Assuming your user has enable the profile to create sensors, it is possible to “create” a new one. To do that, go to the menu option “User” → “My Sensors” and push the button with [+] symbol




-Then, click in any point of the map an fill the Id, Category, Latitude and Longitude fields, then push "Add" button

# New Sensor

Info | Variables | Prop.

Id*

MY_SENSOR_TEST

Category*

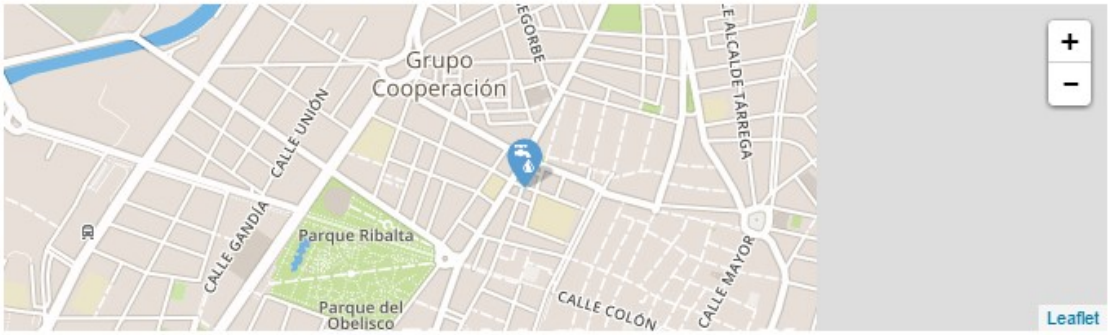
Contador agua doméstico

Latitude*

39,989622

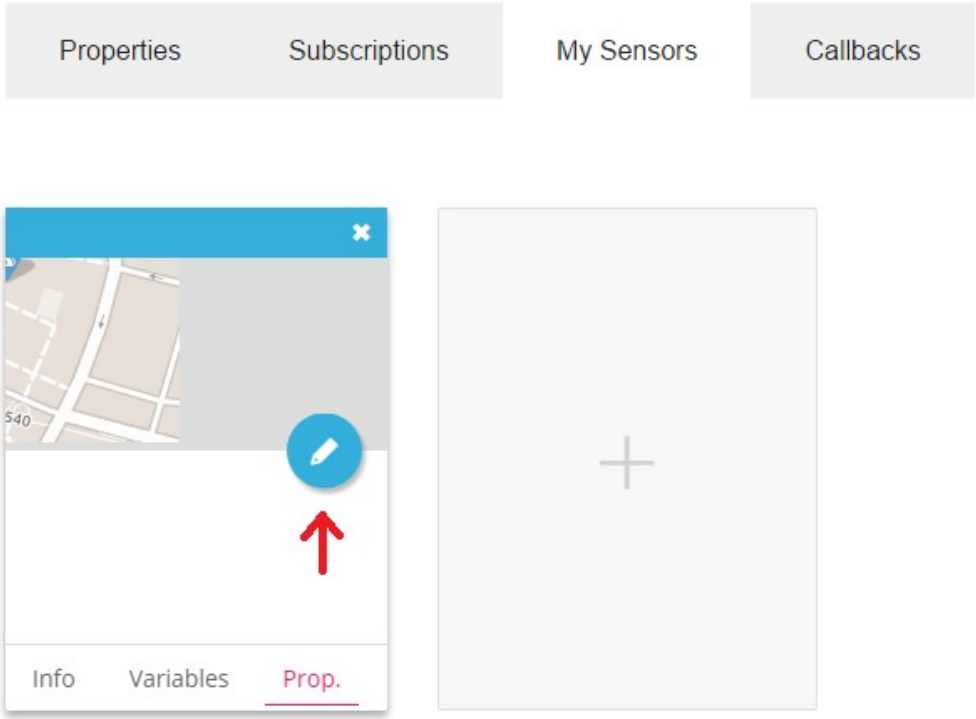
Longitude*

-0,040641

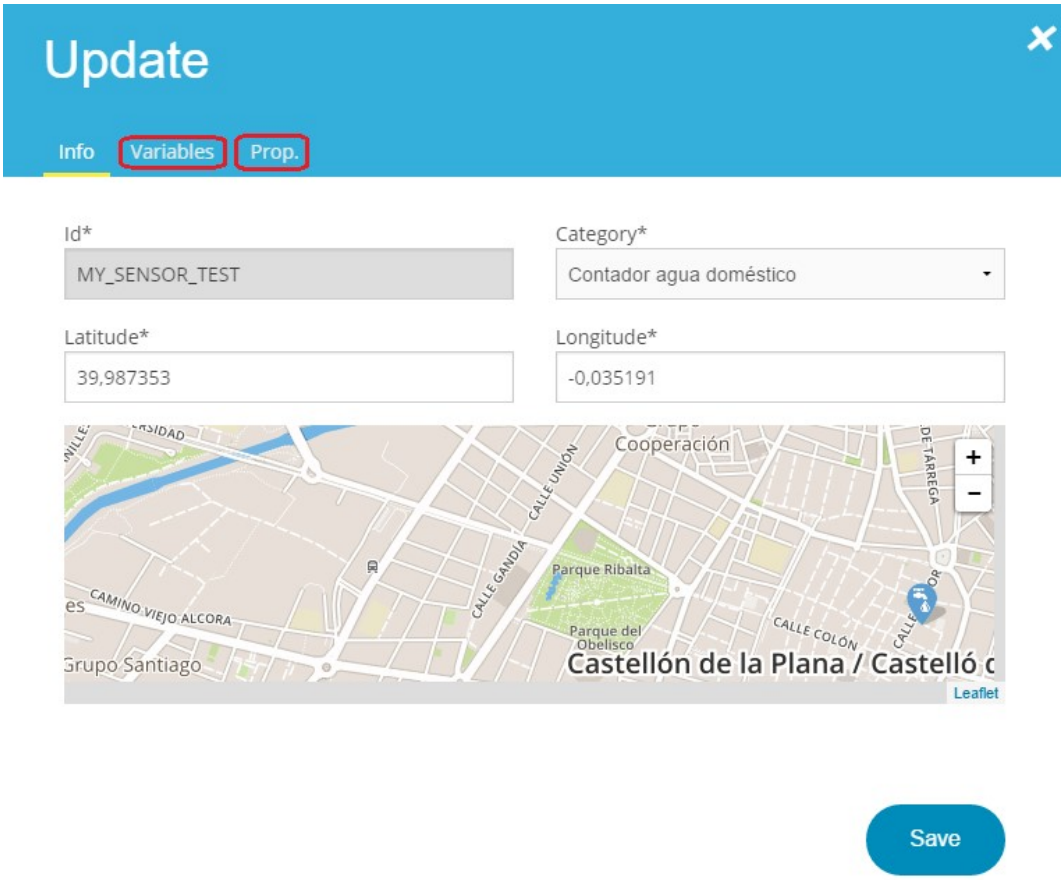



Add

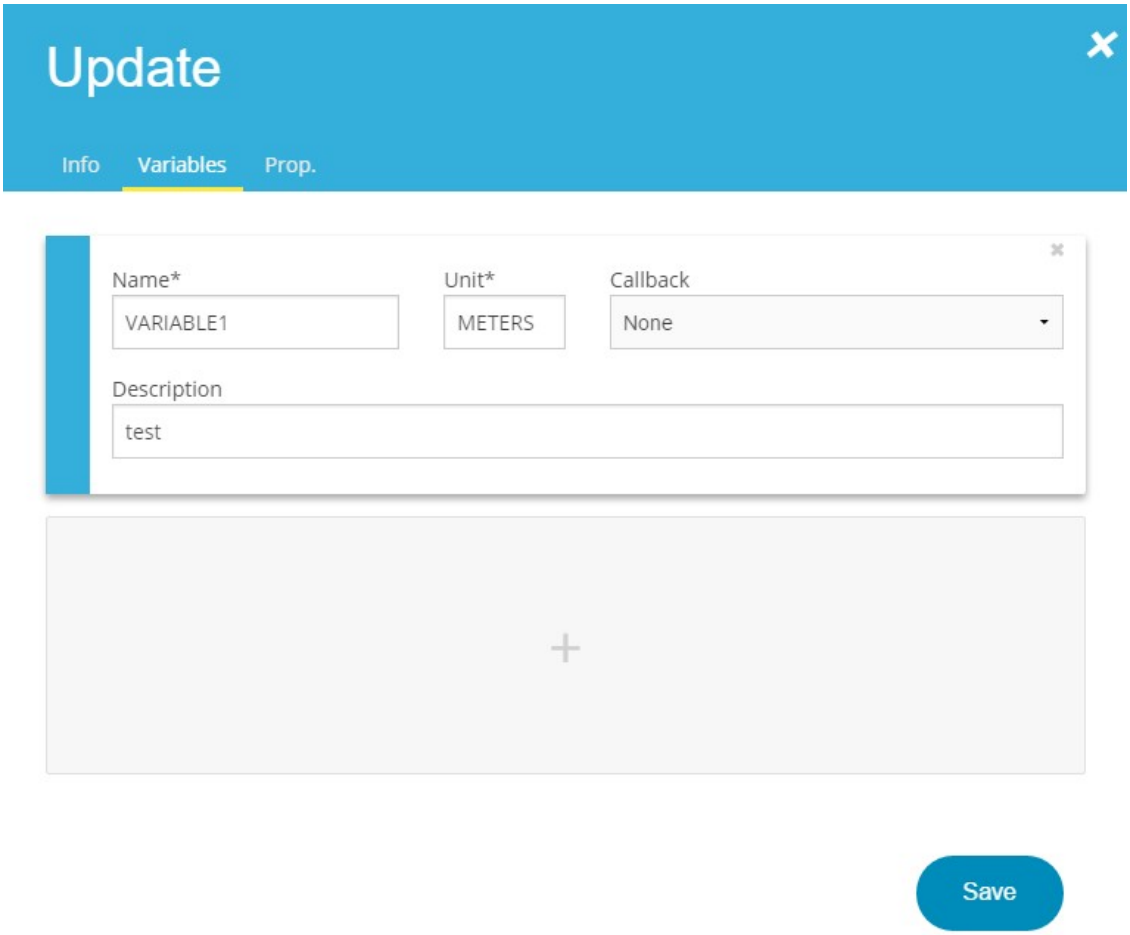
- At this point, the sensor appears on the map. It is possible to add variables and properties by clicking the details button (pencil button)



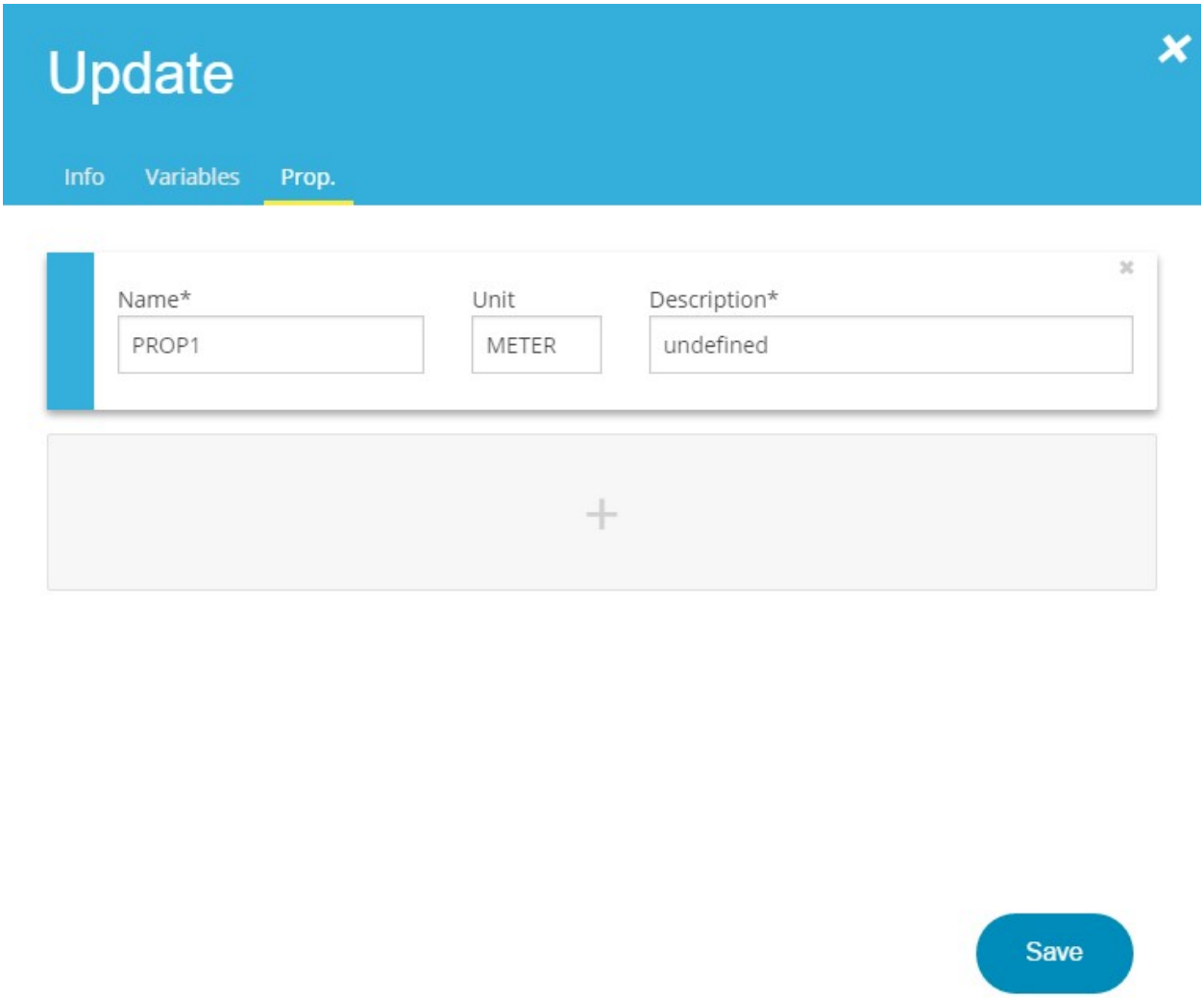
An then click on the “Variables” and “Prop” tabs as follows:



-Fill in details of the variable as follows:



-Fill in details of the properties as follows:



Now, it is possible to retrieve the entered information of this sensor from the service /sensors through the API

**GET** → `http://api.iotsens.com/v1/sensors/MY_SENSOR_TEST`

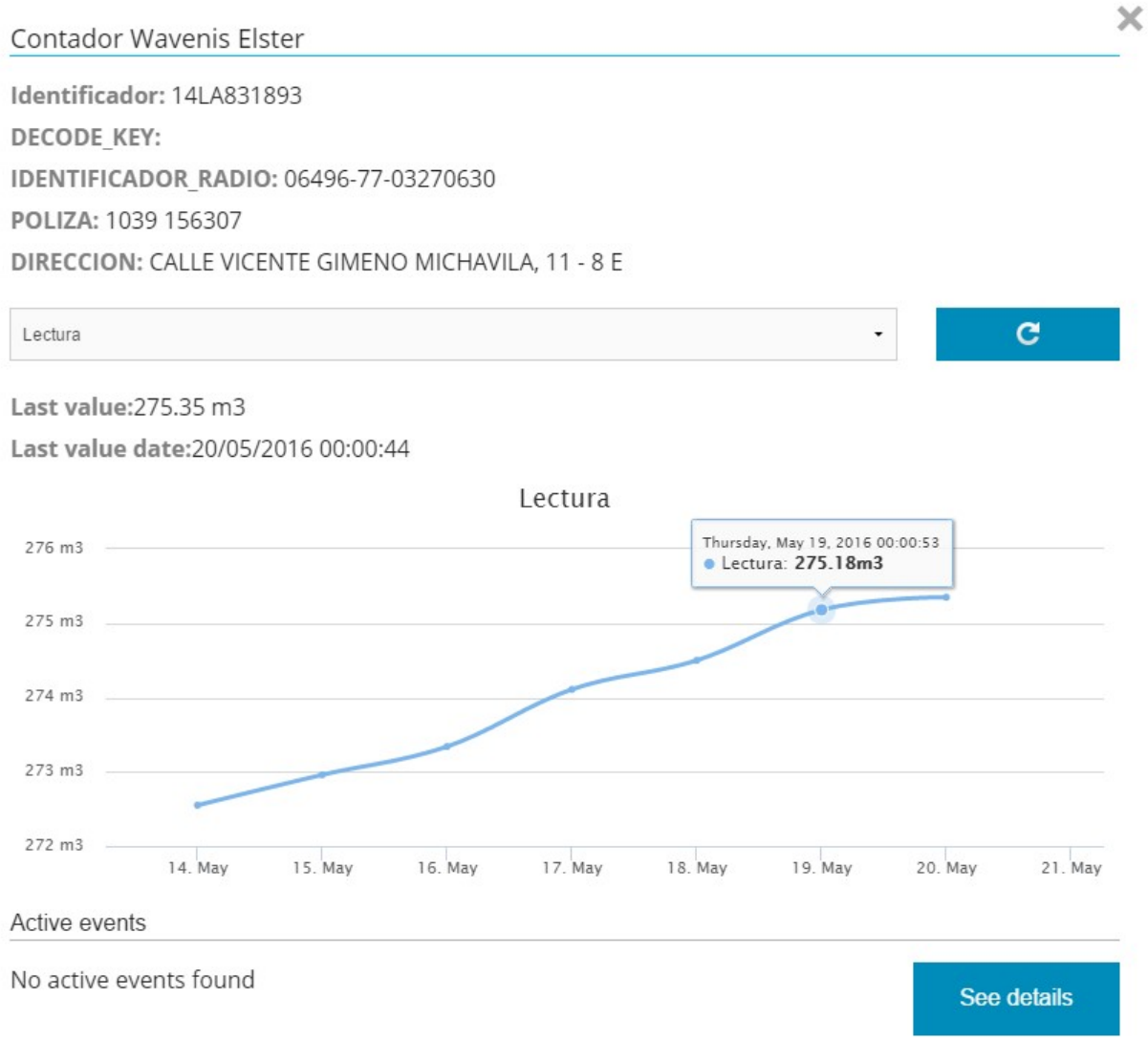
JSON Response


```
{
  "success": true,
  "msg": null,
  "data": {
    "id": 1993,
    "uniqueId": "MY_SENSOR_TEST",
    "variables": [
      {
        "id": 13334,
        "name": "VARIABLE1",
        "type": null,
        "defaultGraphType": null,
        "unit": "METERS",
        "needsPolling": false,
        "pollingGap": null,
        "maxInactivitySeconds": null,
        "description": "test",
        "measureConversors": [],
        "eventGenerators": [],
        "derivedFrom": null,
        "rangeSummaryMinTimeunit": null,
        "summaryOperation": null,
        "displayOrder": 10,
        "roundingDecimals": null,
        "summarizer": {},
        "derived": false,
        "rangesVariable": false
      }
    ],
    "properties": [
      {
        "id": 5859,
        "name": "PROP1",
        "type": null,
        "unit": "METER",
        "writeable": true,
        "remote": false,
        "lastValue": null,
        "lastValueDate": null,
        "displayOrder": 10
      }
    ],
    "sourceTemplate": null,
    "sourceTemplateVersion": null,
    "latitude": 39.987353,
    "longitude": -0.035191,
    "templateEntity": false,
    "enable": true,
    "referenceNode": null,
    "category": {
      "id": 1,
      "name": "Contador agua doméstico",
      "color": "",
      "listingOrder": 30
    },
    "mobile": false,
    "generatedEvents": []
  },
```

```
  "total": 0
}….
```


## Check the measures of an existing sensor

Choose and select any sensor on the map to see the information of the sensor (a popup window will appear)




In This instance, the sensor id is 14LA831893. Click on “See Details” button to see the values of the variable “Lectura” (CURRENT_VALUE)

| Date | Sensor | Variable | Value |
|---|---|---|---|
| 20/05/2016 00:00:44 | 14LA831893 | CURRENT_VALUE | 275.35 |
| 19/05/2016 00:00:53 | 14LA831893 | CURRENT_VALUE | 275.18 |
| 18/05/2016 00:00:44 | 14LA831893 | CURRENT_VALUE | 274.50 |
| 17/05/2016 00:00:44 | 14LA831893 | CURRENT_VALUE | 274.11 |
| 16/05/2016 00:00:44 | 14LA831893 | CURRENT_VALUE | 273.34 |
| 15/05/2016 00:00:44 | 14LA831893 | CURRENT_VALUE | 272.96 |
| 14/05/2016 00:01:07 | 14LA831893 | CURRENT_VALUE | 272.55 |
| 13/05/2016 00:00:46 | 14LA831893 | CURRENT_VALUE | 271.92 |

This information could be retrieved using the API as follows:

**GET →**

`http://api.iotsens.com/v1/sensors/14LA831893/variables/CURRENT_VALUE/measures`

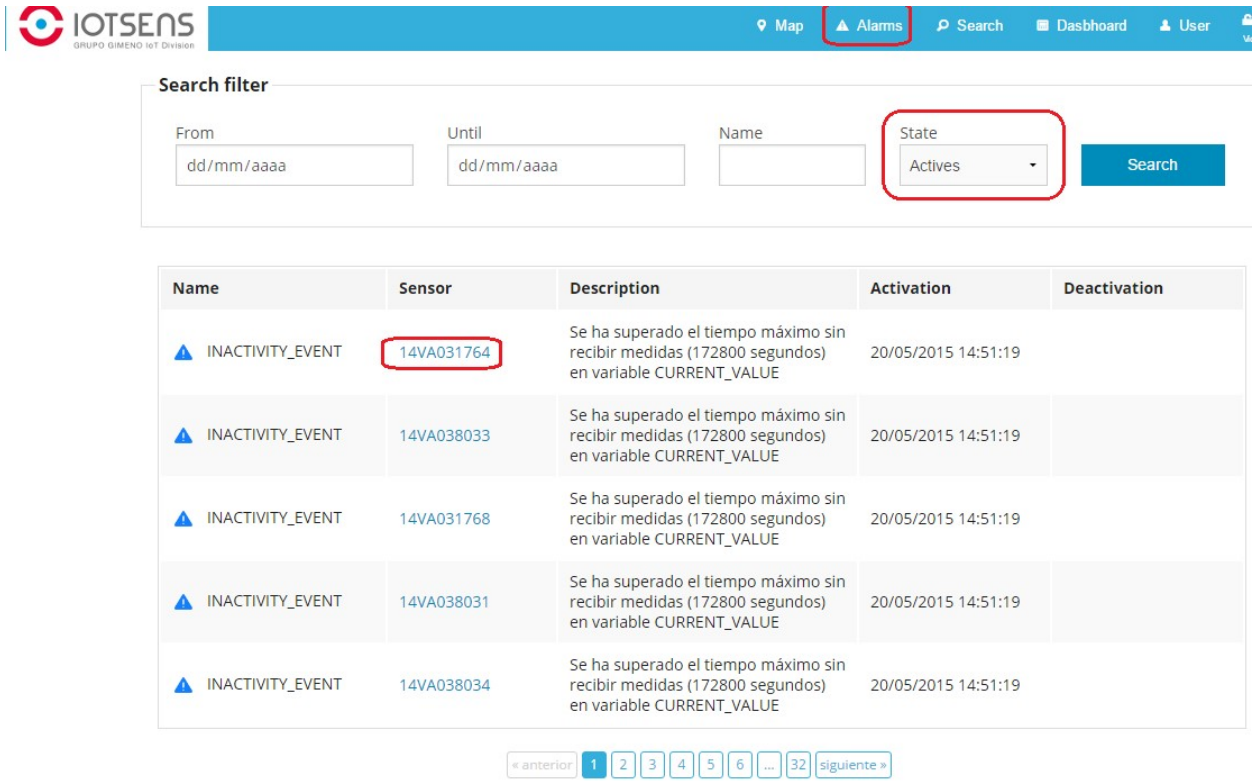
JSON Response


```
{
  "success": true,
  "data": [
    {
      "timestamp": "20/05/2016 00:00:44",
      "variableName": "CURRENT_VALUE",
      "sensorId": "14LA831893",
      "value": "275.35",
      "rawValue": "275.348"
    },
    {
      "timestamp": "19/05/2016 00:00:53",
      "variableName": "CURRENT_VALUE",
      "sensorId": "14LA831893",
      "value": "275.18",
      "rawValue": "275.182"
    },
    {
      "timestamp": "18/05/2016 00:00:44",
      "variableName": "CURRENT_VALUE",
      "sensorId": "14LA831893",
      "value": "274.50",
      "rawValue": "274.496"
    },
    {
      "timestamp": "17/05/2016 00:00:44",
      "variableName": "CURRENT_VALUE",
      "sensorId": "14LA831893",
      "value": "274.11",
      "rawValue": "274.107"
    },
    {
      "timestamp": "16/05/2016 00:00:44",
```

```
      "variableName": "CURRENT_VALUE",
      "sensorId": "14LA831893",
      "value": "273.34",
      "rawValue": "273.336"
    },
    {
      "timestamp": "15/05/2016 00:00:44",
      "variableName": "CURRENT_VALUE",
      "sensorId": "14LA831893",
      "value": "272.96",
      "rawValue": "272.963"
    },
    {
      "timestamp": "14/05/2016 00:01:07",
      "variableName": "CURRENT_VALUE",
      "sensorId": "14LA831893",
      "value": "272.55",
      "rawValue": "272.545"
    }
  ]
}
```


## Check the events of an existing sensor

Click on the “Alarms” option in the main menu to check all the existing events (alarms) and the sensor to which belongs to. Besides,  you can search for the state of the event (active / not active and all)



If we used the API to retrieve the events for sensor 14VA031764, we'll see the one on the first row

**POST →**

`http://api.iotsens.com/v1/events`

Post Parameters:

| | |
|---|---|
| sensorId | 14VA031764 |

JSON Response


```
{
  "success": true,
  "data": [
    {
      "timestamp": "20/05/2015 14:51:19",
      "id": 24074,
      "variableName": "CURRENT_VALUE",
      "name": "INACTIVITY_EVENT",
      "sensorId": "14VA031764",
      "data": "Se ha superado el tiempo máximo sin recibir medidas (172800 segundos) en variable
CURRENT_VALUE",
      "active": true,
      "type": "ALARM",
      "ackState": "NOT_REQUIRED"
    }
  ]
}
```


## ANNEX: Data schemas for JSON Responses

The following is the detailed model of the services covered in this document, showing their attributes types for each JSON returned by the services.

```
SensorCategory {
       id: number // Category Identifier
       listingOrder:number // The order for sorting the sensors categories list
       color: string // The name of the category
       name:  string // The name of the category
}
```

```
SensorBasic {
        id: number // Internal identifier for the sensor
        uniqueId: string // A unique textual code for the sensor
        latitude: number (float) // The latitude where the sensor is located
        longitude: number (float) // The longitude where the sensor is located
        category: SensorCategory
        sourceTemplateVersion: number // The source template version (revision)
        sourceTemplate: {
                id: number // Internal identifier for the template
                templateName: string // The name of the template
        }
        enable: boolean // the status of the sensor
        templateEntity: boolean // whether this sensor could be used as a template for
        new ones
}
```

```
SensorBasicWithProperties {
        id: number // Internal identifier for the sensor
        uniqueId: string // A unique textual code for the sensor
        latitude: number (float) // The latitude where the sensor is located
        longitude: number (float) // The longitude where the sensor is located
        category: SensorCategory
        sourceTemplateVersion: number // The source template version (revision)
        sourceTemplate: {
                id: number // Internal identifier for the template
                templateName: string // The name of the template
        }
        enable: boolean // the status of the sensor
        templateEntity: boolean // whether this sensor could be used as a template for
        new ones
        properties:
                [
                SensorProperty
                ]
        }
```

```
SensorWithPropertiesAndVariables {

        id: number // Internal identifier for the sensor

        uniqueId: string // A unique textual code for the sensor

        latitude: number (float) // The latitude where the sensor is located

        longitude: number (float) // The longitude where the sensor is located

        category: SensorCategory

        enable: boolean // the status of the sensor

        templateEntity: boolean // whether this sensor could be used as a template for
        new ones

        sourceTemplateVersion: number // The source template version (revision)

        sourceTemplate: {

                id: number // Internal identifier for the template

                templateName: string // The name of the template
        }
        referenceNode: {
        }

        mobile: boolean // Mobile

        generatedEvents:
                [
                 {}
                ]
        properties:
                [
                SensorProperty
                ]
        variables:
        [
                SensorVariable
        ]
}
```

```
SensorProperty {

        id: number // Internal identifier for the property

        name: string // The name of the property

        lastValue: string // The last valid value for the property

        type: string // Type of the property
                Enum:
                        Array[8]
                        0:"STRING"
                        1:"UNSIGNED_INTEGER"
                        2:"INTEGER"
                        3:"LONG"
                        4:"FLOAT"
                        5:"DOUBLE"
                        6:"BOOLEAN"
                        7:"LOG"
}
```

```
SensorVariable {

        id: number // Internal identifier for the variable

        name: string // The name of the variable

        description: string // A human readable description of the variable contents

        type: string // The type of the measures of this variable

        defaultGraphType: string // Default graphical representation
                    Enum:
                           Array[2]
                           0:"LINE"
                           1:"BARS"

        unit: string // Physical unit as string of the variable

        needsPolling: boolean // Flag to set polling to the sensor from the platform

        pollingGap: number // Gap between polling requests

        maxInactivitySeconds: number // Maximum time allowed before raising an alarm

        measureConversors:
        [
               {} // Array of conversors to automatically apply sequencially

        ]

        eventGenerators:
        [
               {} // Array of notifications generators to automatically apply
        sequencially

        ]

        derivedFrom: number // Original Variable from which the variable is derived from

        rangeSummaryMinTimeunit: string // Enumeration of availables intervals to
        performs the aggregations

        summaryOperation: string // Enumeration of available aggregation operations

        displayOrder: number // Visual order for this variable

        roundingDecimals: number // Number of decimals to round for the retrieved value

        derived: boolean // flag to know whether the variable is derived from another
        one

        rangesVariable: boolean // flag to know whether this variable is an aggregation
}
```

**Event** {

id: number *// Event identifier*

sensorId: string *// The unique identifier of the sensor that generated this event*

variableName: string *// The name of the variable of the sensor that generated this event*

timestamp: string (date-time) *// The instant when this event was activated*

name: string *// The name of the event*

active: boolean *// If this event is still active or not*

deactivationTime: string (date-time) *// The instant when this event was deactivated, if it has been*

data: string *// Some data that can be used to give more information about the event*

}

**EventComment** { *//A user comment about a generated event*

id: number *// The identifier of the event comment*

timestamp: string (date-time) *// The instant when this comment was generated*

owner: **User**

comment: string *// The content of the comment*

}

**EventDescription** { *//The description of a event that can be generated by some sensor*

id: number *// The identifier of the event description*

SensorCategory: **SensorCategory**

name: string *// The name of this event description*

type: string *// The kind of event*

```
Enum:
      Array[3]
      0:"ALARM"
      1:"WARNING"
      2:"INFO"
```

defaultTemplate: **NotificationTemplate**

}

**NotificationTemplate** { *//The template of the notification which is generated with an event*

id: number *// The identifier of the notification template*

titleTemplate: string *// The identifier of the notification template*

preScript: string *// A javascript routine needed to render the notification*

contentsTemplate: string *// The content of the notification with the fields that will be substituted by real values*

}

**Measure** {

sensorId: string *// The unique identifier of the sensor that generated this measure*

variableName: string *// The name of the variable of the sensor that generated this measure*

timestamp: string (date-time) *// The instant when this measure was generated*

value: string *// The value of this measure*

rawValue: string *// The value of this measure as it was sent by the sensor*

}

**SummarizedMeasure** {

sensorId: string *// The unique identifier of the sensor that generated this measure*
variableName: string *// The name of the variable of the sensor that generated this measure*

timestamp: string (date-time) *// The instant when this measure was generated*

value: string *// The value of this measure*

rawValue: string *// The value of this measure as it was sent by the sensor*

summaryTimeUnit: string *// The time unit which represents the measure*

```
Enum:
Array[7]
0:"SECONDS"
1:"MINUTES"
2:"HOURS"
3:"DAYS"
4:"WEEKS"
5:"MONTHS"
6:"YEARS"
```

}

```
SensorEventSubscription {

        eventType: {

                name: string // The name of the event type

                description: string // The description of the event type
        }
}

PersonalEventGenerator {

        id: integer // The identifier of the personal event generator

        privateEvent: boolean // If other users can subscribe to this event

        eventGenerator:  EventDescription {
                generatedEvent: EventDescription
        }
}


EventGenerator {

        id: integer // The identifier of the event generator

        typeName: string // The kind of condition that triggers the event
                Enum:
                Array[4]
                0:"HasExactValueEventGenerator"
                1:"LowerTresholdEventGenerator"
                2:"UpperTresholdEventGenerator"
                3:"BooleanValueEventGenerator"

        generatedEvent: EventDescription

        variable: SensorVariable

        treshold: integer // The value that triggers the event

        automaticDeactivation: boolean // If the generated event is deactivated
        automatically
}
```