



SOFIA2 IOT WORKSHOP

December 2016

Versión 1



1 Introduction

The goal of this workshop is the realization of a real example on which we can evaluate the capabilities of Sofia2 platform.

To do this, we are going to simulate a building that has several floors, and three devices for the reading of energy consumption, temperature and humidity in each one of them. Finally, we will create a dashboard to display this information and we will publish data in an API to be consumed in a simple way by any application.

2 Data Model

2.1 The Model

The main support of an IoT Project is its data model. It must contain all relevant information, both for immediate use and for further analysis of the information.

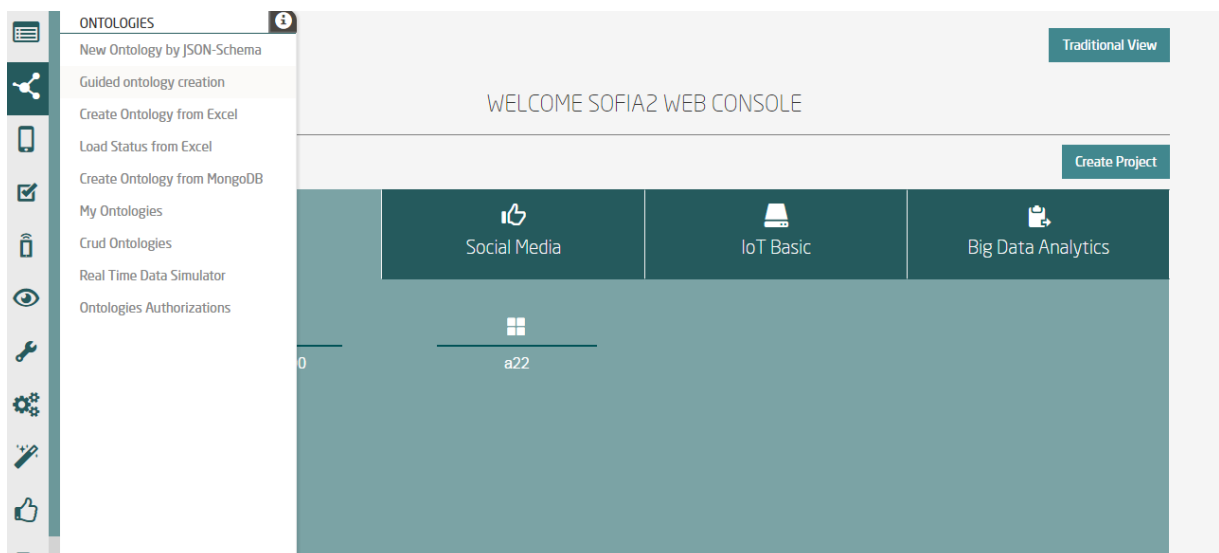
It is significant that the devices don't have to send irrelevant information. This only would generate a higher cost in the communications.

In the generation of a balanced model lies the complexity of the data model design.

2.2 Ontology

On Sofia2, the data model is named Ontology. We can define it in a very superficial way as a JSON schema that will explicitly define the data it will store.

The sofia2 centralized management console has several methods for creating an ontology (graphical editor, text mode, wizard and from data source).



We are going to use the Guided Ontology Creation.

ONTOLOGIES / GUIDED ONTOLOGY CREATION

Ontology

Name (*)

Minimum of 5 characters

Actual Template Version

3

☐ Public

Description

☒ Active

RTDB and HDB configuration

Remove RTDB information (no BDH copy) of the previous :

1 day

☒ Remove RTDB information (no BDH copy)

Preprocess class RTDB to BDH

☐ Group data

Dependencies

The first thing we have to do is give a name to our ontology, we will call it TallerIoTPTG (WorkshopIoT<and ours initials>).

We put the ontology as active and it is a good practice to indicate the description of the purpose that our ontology has.

ONTOLOGIES / GUIDED ONTOLOGY CREATION

Ontology

Name (*)

TallerIoTPTG

Minimum of 5 characters

Actual Template Version

3

☐ Public

☒ Active

We can ignore the details of BDTR and BDH Configuration and dependence between ontologies, since it has no relevance for this workshop

Now we are going to add the fields of our data model, we will work with a very simple model that will contain the following information:

ID	String	required
LOCATION	String	required
TEMPERATURE	Number	not required
WATTS	Number	not required
HUMIDITY	Number	not required
TYPE	String	required

New Ontology Property

Property

Datatype







required

TYPE

string

required

Add

Property	Datatype	required	
ID	string	required	
LOCATION	string	required	
TEMPERATURE	number	not required	
WATTS	number	not required	
HUMIDITY	number	not required	
TYPE	string	required	

We set Additional Properties to false to prevent our ontology from containing any other information. And we press the 'generate ontology' button.

Additional Properties

false

Generate ontology

In the Scheme section, we will see the definition of the JSON-Schema that defines our ontology and has to comply with all the instances of ontology we use.

Schema

Template

Empty

Text

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "title": "WorkshopIoTPTG Schema",
  "type": "object",
  "required": [
    "WorkshopIoTPTG"
  ],
  "properties": {
    "WorkshopIoTPTG": {
      "type": "string",
      "$ref": "#/datos"
    }
  }
},
"datos": {
  "description": "Info WorkshopIoTPTG",
  "type": "object",
  "required": [
    "ID",
    "LOCATION",
    "TEMPERATURE",
    "WATTS",
    "HUMIDITY",
    "TYPE"
  ],
  "properties": {
    "ID": {
      "type": "string"
    },
    "LOCATION": {
      "type": "string"
    },
    "TEMPERATURE": {
      "type": "number"
    },
    "WATTS": {
      "type": "number"
    },
    "HUMIDITY": {
      "type": "number"
    },
    "TYPE": {
      "type": "string"
    }
  }
}
```

If we click the Generate Instance button it will show us an example of an ontology instance.

JSON Instance

```
{"WorkshopIoTPTG":{"ID":"string","LOCATION":"string","TEMPERATURE":28.6,"WATTS":28.6,"HUMIDITY":28.6,"TYPE":"string"}}
```

Generate Instance

Cancel

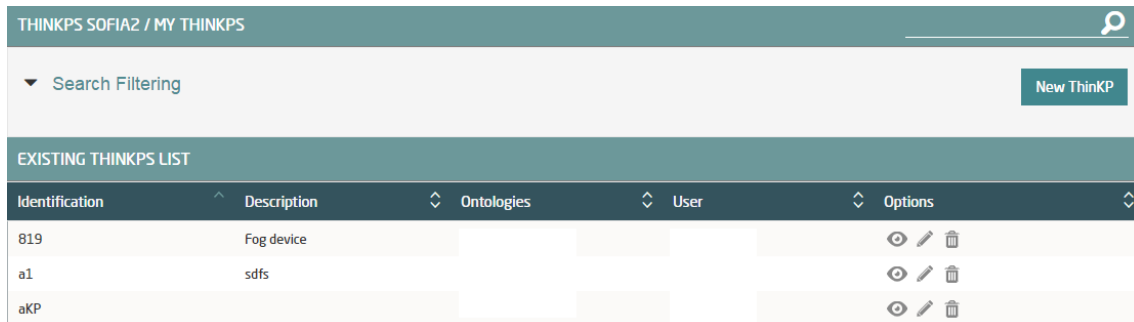
New










Finally, we click the Create button.

3 ThinKP

Once we have defined the data model and put it into an ontology, we have to create the ThinKP, the logical configuration of the devices that are going to interact with our ontology.

To do this, we access My ThinkKPs menu and click on the New ThinkKP button.



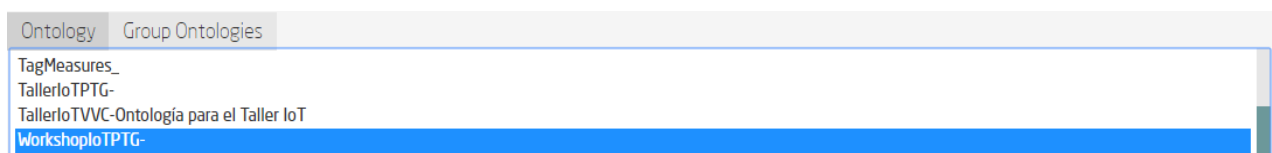
Identification	Description	Ontologies	User	Options
819	Fog device			  
a1	sdfs			  
aKP				  

We assign a name to our ThinkKP, we will call it TallerIoT (WorkshopIoT<and ours initials>).



ThinkKP	Mis Tokens	Mis Instancias
Identificación	Clave de cifrado	
WorkshopIoTPTG	c29f3321-14a9-4327-b050-	

We can give our ThinkKP a description. And we must select the ontology we created at point 3. Should be called TallerIoTPTG (WorkshopIoT<and ours initials>).



Ontology	Group Ontologies
TagMeasures_	
TallerIoTPTG-	
TallerIoTVC-Ontología para el Taller IoT	
WorkshopIoTPTG-	

Once given this information we can click the Create button, we will see the detail screen of ThinKP.

THINKPS SOFIA2 / SHOW THINKP

ThinKP My Tokens My Instances

Identification

WorkshopIoTPTG

Encryption Key

c29f3321-14a9-4327-b050-

User

Description

Meta-Inf


Ontologies

WorkshopIoTPTG

We can always access to our tokens using the My Tokens tab.

THINKPS SOFIA2 / SHOW THINKP

ThinKP My Tokens My Instances

ThinKP	Owner	Token	Last Connection	Active
WorkshopIoTPTG		f6196896ed2642d99e77f46aa		

Showing 1 to 1 of 1 entries

First Previous 1 Next Last

Cancel

We select the editing icon of our ThinkKP (pencil).

THINKPS SOFIA2 / MY THINKPS

▼

Filtrar Búsqueda

New ThinkKP

EXISTING THINKPS LIST

Identification	Description	Ontologies	User	Options
aThinK1	ThinKP de prueba	aMapa1 apruebaonto		<div> <div></div> <div></div> <div></div> </div>
aThinKPRopa		aRopa		<div> <div></div> <div></div> <div></div> </div>
Instancia de kp_		SensorTemperaturaEjBienvenida		<div> <div></div> <div></div> <div></div> </div>
KpScada_Victor Vicente	Kp para la suscripcion y el envio de información desde SCADA	TagMeasures_Victor Vicente		<div> <div></div> <div></div> <div></div> </div>
sdfsdfsd		TallerloTPTG		<div> <div></div> <div></div> <div></div> </div>
TallerloTPTG		TallerloTPTG		<div> <div></div> <div></div> <div></div> </div>
WorkshoploTPTG		WorkshoploTPTG		<div> <div></div> <div></div> <div></div> </div>

Showing 1 to 7 of 7 entries

First

modify

Previous

1

Next

Last


We select My Tokens tab. We will see a list with the Tokens that are assigned and the options to activate / deactivate, to unsubscribe and to add new Tokens.

THINKPS SOFIA2 / MODIFY THINKP

ThinkP

My Tokens

My Instances

ThinKP	Owner	Token	Last Connection	Active	Options
WorkshoploTPTG		f6196896ed2642d99e7		<input checked="" type="checkbox"/>	

Showing 1 to 1 of 1 entries

First

Previous

1

Next

Last

1

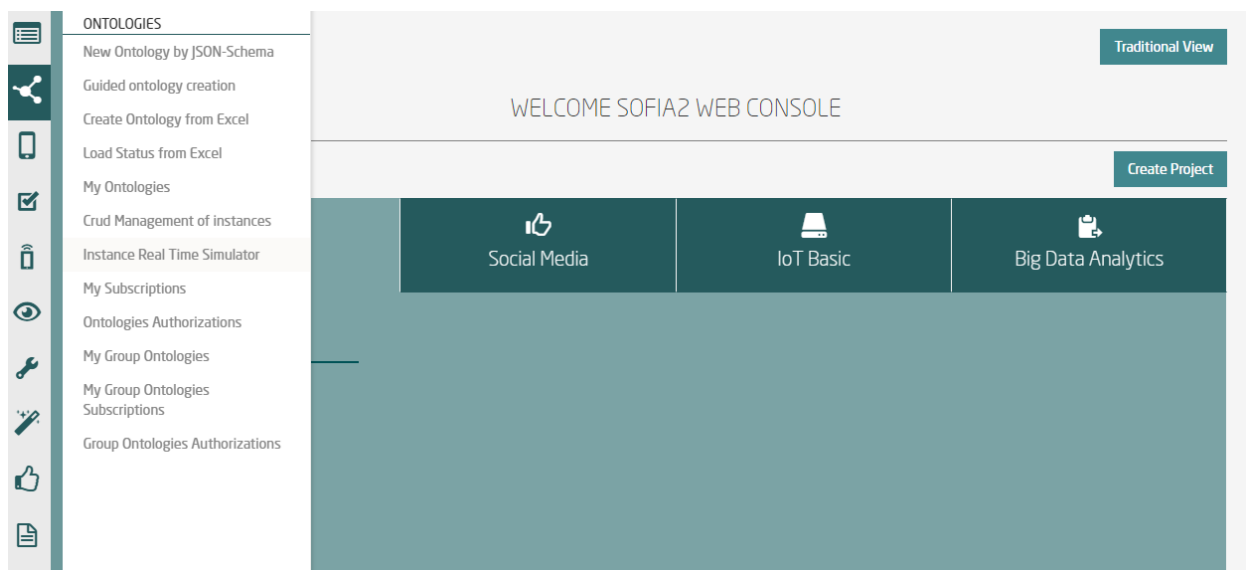
Generate Tokens

Cancel

4 Simulate input data

4.1 Simulator configuration

Since we don't have a real device that inserts information into our ontology, we will make use of Sofia2 tools to insert simulated data, to do this, we will access to the 'Instance Real Time Simulator' from the 'Ontologies' menu.



Click on 'Create Simulator' button. The first thing we are going to create are the Instance Generators we are going to use. An Instance Generator is a data test definition.

We are going to create the next generators:

- First of all, the ID field generator for our supposed Wattmeter. As a name we'll write 'WorkshopIoT WATPTG' (WorkshopIoT WAT+our initials), as Generator Type, 'Fixed String' and as value, the same as the name we have given. Click 'Add Instance Generator' button.

^ INSTANCE GENERATORS

Generator List

boolean
codigoprocesso
coseno prueba
d
de 1 a 10
Fechas desde 2016 a 2020
Fechas desde 2016 a 2021
Fechas junio 2016
GadgetReal
generador_abn
generador_puc_abn
generator
generator2
gerador_temperatura
indicadorFlecha
Integer 0
Integer 0 a 1
Integer 0 a 10
Integer 0 a 100
Integer 0 a 2
Integer 0 a 3

Generator Name

WorkshoploTwATPTG

Generator Type

Fixed String

Insert Instance Each

5

seconds

Value

WorkshoploTwATPTG

Add Instance Generator
Delete Instance Generator

- Now the ID field generator for our supposed Thermostat, as the name we'll write 'WorkshoploTThermostatPTG' (WorkshoploTThermostat+our initials), as Generator Type, 'Fixed String' and as value the same as the name we have given. Click 'Add Instance Generator' button.
- To end with the ID, the ID field generator for our supposed humidity meter, as the name we'll write 'WorkshoploTHPTG' (WorkshoploTH+our initials), as Generator Type 'Fixed String' and as value the same as the name we have given. Click 'Add Instance Generator' button.
- As a values generator we are going to create a single generator that we'll use it to simulate temperature, humidity and consumed watts, we'll call it 'WorkshoploTVALUEPTG' (WorkshoploTVALUE+our initials), as Generator Type 'Random Number', values from 1 to 100 and decimal precision 2. Click 'Add Instance Generator' button.
- For the location we'll create a 'Random String' generator, with the word list HALL, GF, F1, F2, F3, B1 and B2, simulating the floors of a building. And we are going to call it WorkshoploTLOCATIONPTG. Click 'Add Instance Generator' button.
- Finally, we are going to create the generator types for the 'TYPE' field, which will be of 'Fixed String' type and will be called 'WorkshoploTTYPEHPTG' and 'HUMIDITY' value, 'WorkshoploTTYPETPTG' and 'TEMPERATURE' value and 'WorkshoploTTYPEWPTG' and 'WATTS' value. Click 'Add Instance Generator' button.

Once we have defined the generators we'll create three simulators, the temperature meter, humidity meter and wattmeter, to do this, in the 'identification field' we'll write the name 'WorkshoploTSIMULATORPTG', 'WorkshoploTSIMULATORHPTG' and 'WorkshoploTSIMULATORWPTG'.

Simulator

Identification

WorkshoploTSIMULATORPTG

In the 'Ontology' tab, we select our ontology WorkshoploTPTG (WorkshoploT+our initials).

^ ONTOLOGY

Identification

WorkshoploTPTG

We create the 'Temperature Simulator' configuration.

WorkshoploTPTG

Info WorkshoploTPTG

ID WorkshoploTThermo

LOCATION WorkshoploTLOCATI

TEMPERATURE WorkshoploTVALUEP

WATTS VACIO

HUMIDITY VACIO

TYPE WorkshoploTTYPETP

Click 'Create Simulator' button.

We create the 'Humidity Simulator' configuration.

WorkshoploTPTG ▼

Info WorkshoploTPTG

ID WorkshoploTHPTG ▼

LOCATION WorkshoploTLOCATIK ▼

TEMPERATURE VACIO ▼

WATTS VACIO ▼

HUMIDITY WorkshoploTVALUEP ▼

TYPE WorkshoploTTYPEHF ▼

We create the 'Wattmeter Simulator' configuration.

WorkshoploTPTG ▼

Info WorkshoploTPTG

ID WorkshoploTWATPTC ▼

LOCATION WorkshoploTLOCATIK ▼

TEMPERATURE VACIO ▼

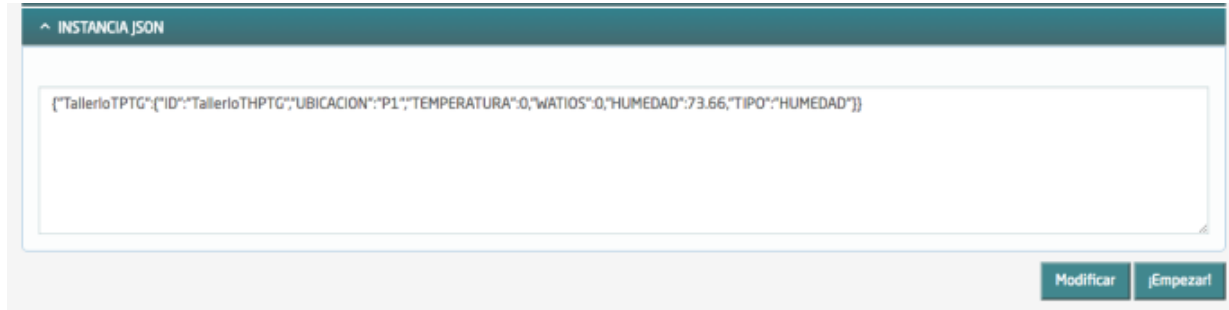
WATTS WorkshoploTVALUEP ▼

HUMIDITY VACIO ▼

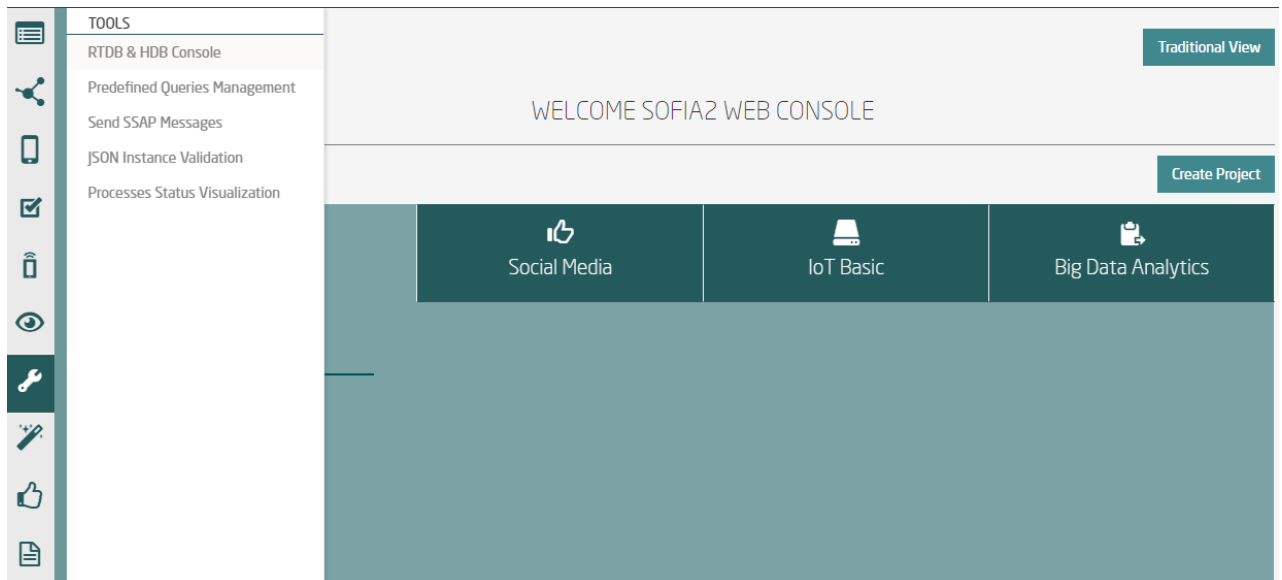
TYPE WorkshoploTTYPEWi ▼

4.2 Simulator Execution

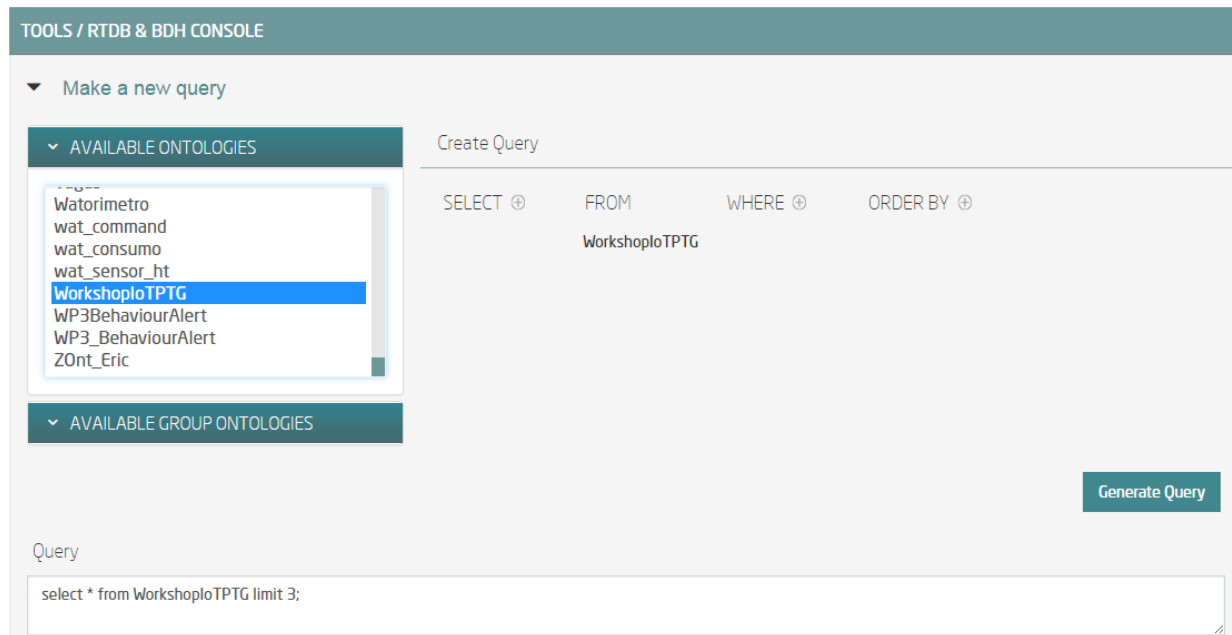
We open three new tabs in the browser, we access each of them to one of the three simulators and we press the button Start! in each one of them.



We open a fourth tab and we access the option menu 'TOOLS' -> 'RTDB & BDH CONSOLE'.



We select our ontology.



TOOLS / RTDB & BDH CONSOLE

▼ Make a new query

▼ AVAILABLE ONTOLOGIES

- Watorimetro
- wat_command
- wat_consumo
- wat_sensor_ht
- WorkshopIoTPTG**
- WP3BehaviourAlert
- WP3_BehaviourAlert
- ZOnt_Eric

▼ AVAILABLE GROUP ONTOLOGIES

Create Query

SELECT ⊕ FROM WHERE ⊕ ORDER BY ⊕

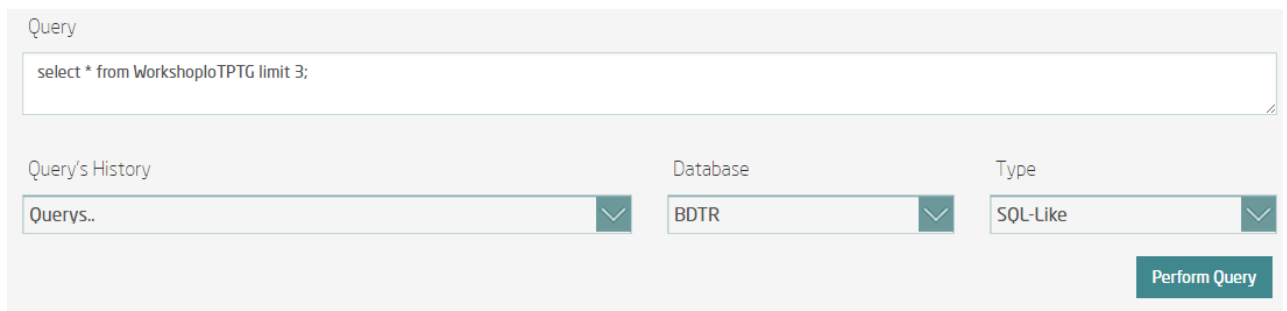
WorkshopIoTPTG

Generate Query

Query

select * from WorkshopIoTPTG limit 3;

We'll see a default query for querying RTDB using the SQL-Like language.



Query

select * from WorkshopIoTPTG limit 3;

Query's History Database Type

Querys.. BDTR SQL-Like

Perform Query

If we click on 'Perform Query' button, it will return the data contained in the Database. We must be aware that the platform by configuration will return us at most 100 Registries.

```
[
  {
    "_id": {
      "$oid": "582da3e2e4b05a47"
    },
    "contextData": {
      "session_key": "3a0bee53-df11-4f26-b9a4-",
      "user": " ",
      "kp": "KP_SIB_API",
      "kp_instancia": "16d50bcc-ba63-4aaf-b679-",
      "timestamp": {
        "$date": "2016-11-17T12:34:41.085Z"
      }
    },
    "WorkshopIoTPTG": {
      "ID": "WorkshopIoTHTPG",
      "LOCATION": "F1",
      "TEMPERATURE": 0,
      "WATTS": 0,
      "HUMIDITY": 98.51,
      "TYPE": "HUMIDITY"
    }
  },
]
```

Stop our three simulators by clicking on the 'Stop!' button.

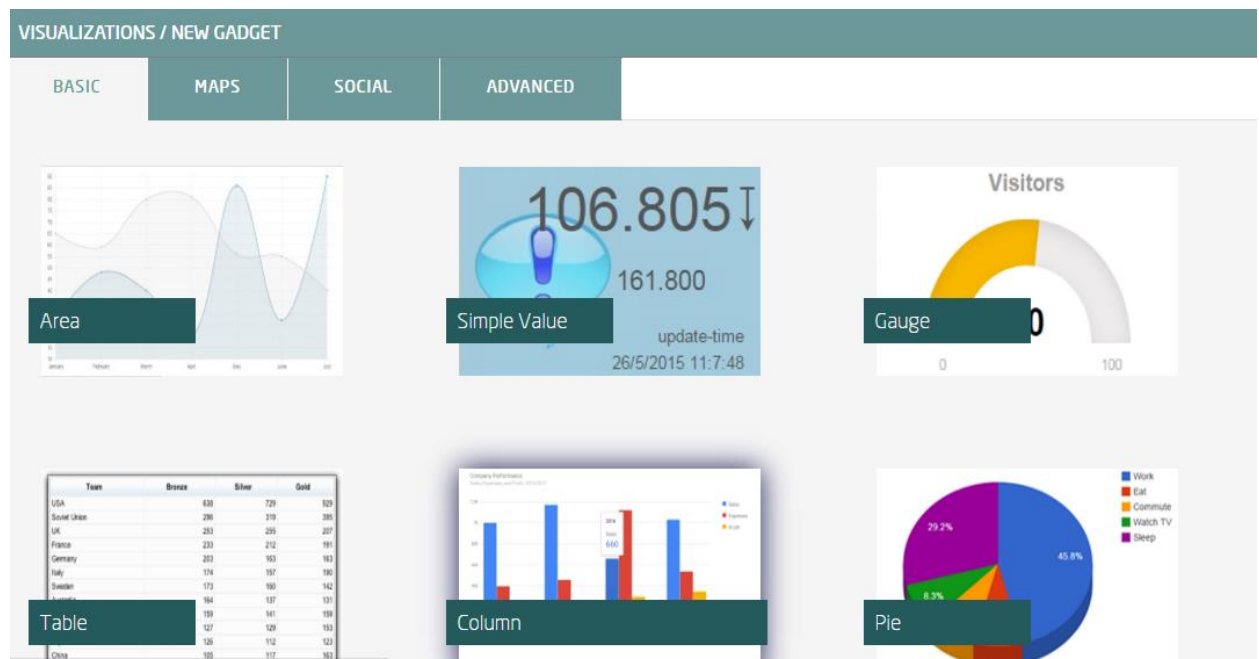
```
[{"TallerioTPTG":{"ID":"TallerioTTERMOSTATOPTG","UBICACION":"P1","TEMPERATURA":17.8,"WATIOS":0,"HUMEDAD":0,"TIPO":"TEMPERATURA"}]
```


5. Control Panel

The next step is to create a Control Panel using the graphical presentation capabilities of the platform.

5.1 Gadget Creation

Go to 'My Gadgets' menu and click on 'New Gadget', select the Column type.



As a name, we assign the IoTWORKSHOPGADGETHPTG (IoTWORKSHOPGADGETH + our initials) and select the ThinkP that we created.

Name	IoTWORKSHOPGADGETHP
KP	WorkshopIoTPTG ▼

Select the 'Get data from query' tab, use the 'select * from WorkshopIoTPTG where WorkshopIoTPTG.TYPE = 'HUMIDITY'; And select as measures for the X axis 'contextData.timestamp', we must transform it using 'new Date(\$0)' and for the Y axis 'HUMIDITY'.

Query

Ontology	Query	
WorkshopIoTPTG	select * from WorkshopIoTPTG where WorkshopIoTPTG.TYPE = 'HUMIDITY'	<button>Remove</button>

Measures

X axis	X data transform	Y axis	Y data transform	Swap axis	Swap a
contextData.timestamp	new Date(\$0)	HUMIDITY			

contextData.timestamp new Date(\$0) HUMIDITY

Token

f6196896ed2642d99e77f4

Cancel Create

We'll do the same to create the consumption graph of Watts, which we'll call IoTWORKSHOPGADGETW + Our initials. In this case with the query 'select * from WorkshopIoTPTG where WorkshopIoTPTG.TYPE = 'WATTS'; Select as measures for the X axis 'contextData.timestamp', we must transform it using 'new Date(\$0)'. And in the Y axis 'WATTS'.

Query

Ontology	Query	
WorkshopIoTPTG	select * from WorkshopIoTPTG where WorkshopIoTPTG.TYPE = 'WATTS'	<button>Remove</button>

Measures

X axis	X data transform	Y axis	Y data transform	Swap axis	Swap a
contextData.timestamp	new Date(\$0)	WATTS			

contextData.timestamp new Date(\$0) WATTS

Token

f6196896ed2642d99e77

Cancel Save

And to create the temperature graph, which we'll call IoTWORKSHOPGADGETT + Our initials. In this case with the query 'select * from WorkshopIoTPTG where WorkshopIoTPTG.TYPE = 'TEMPERATURE''. Select as measures for the X axis 'contextData.timestamp', we must transform it using 'new Date(\$0)', and in the Y-axis 'TEMPERATURE'.

Query

Ontology	Query	
WorkshopIoTPTG	select * from WorkshopIoTPTG where WorkshopIoTPTG.TYPE = 'TEMPERATURE'	<button>Remove</button>

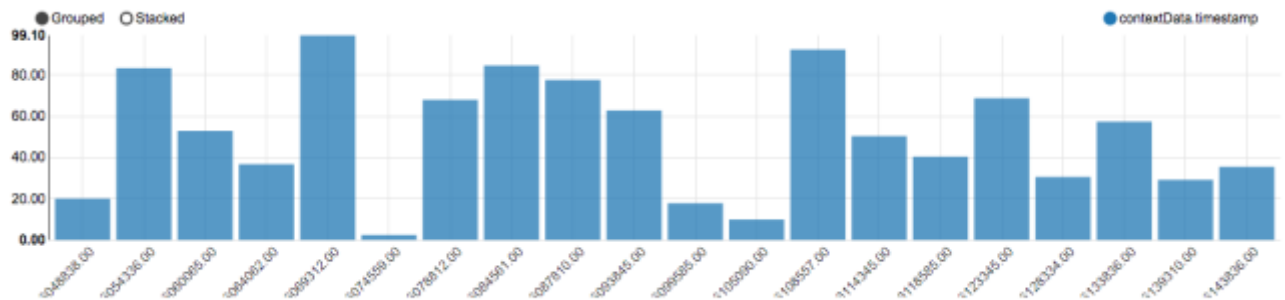
Measures

Y axis transform	Y axis	Y data transform	Swap axis	Swap axis transform	Serie name	
new Date(\$0)	TEMPERATURE					<button>Remove</button>
<input type="text" value="\$0"/>	<input type="text" value="TEMPERATURE"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<button>Add</button>

Token

CancelCreate

For each one of the previous Gadgets, you'll create a chart like this:



Finally, we'll create a table type Gadget, we'll call it `IoTWORKSHOPGADGETTABLE` + Our initials, we'll select the tab 'Get live data' and we'll add the following columns:

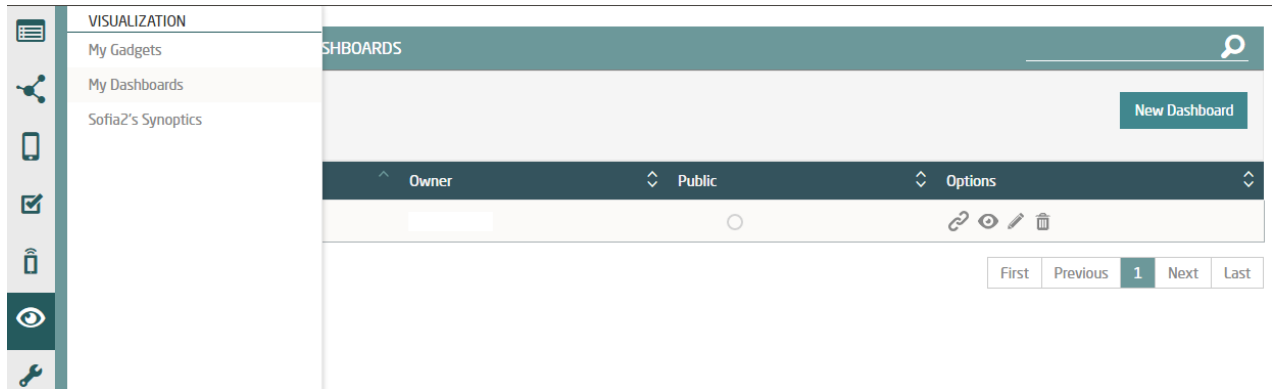
Measures	
Ontology	Column
WorkshopIoTPTG	TYPE
WorkshopIoTPTG	LOCATION
WorkshopIoTPTG	HUMIDITY
WorkshopIoTPTG	TEMPERATURE
WorkshopIoTPTG	WATTS

We get a table like this:

	TYPE	LOCATION	HUMIDITY	TEMPERATURE	WATTS
1	HUMIDITY	B2	25,35	0	0
2	TEMPERATURE	F3	0	80,71	0
3	HUMIDITY	B2	36,07	0	0
4	WATTS	B1	0	0	73,67
5	TEMPERATURE	F1	0	65,97	0
6	HUMIDITY	F3	67,66	0	0
7	WATTS	GF	0	0	69,44
8	TEMPERATURE	F3	0	35,57	0
9	HUMIDITY	F2	25,99	0	0
10	TEMPERATURE	GF	0	31,21	0
11	WATTS	GF	0	0	75,26

5.2 Dashboard creation

Once we have created the Gadgets, we are going to create a Dashboard that uses them, to do this, go to 'My Dashboards' menu and click on 'New Dashboard'



We'll call it IoTWORKSHOPDASHBOARD + Our initials and we'll check it as public. Press the 'New Page' button.

Name

IoTWORKSHOPDASHBOARDPTG ☒ Public

Logo image

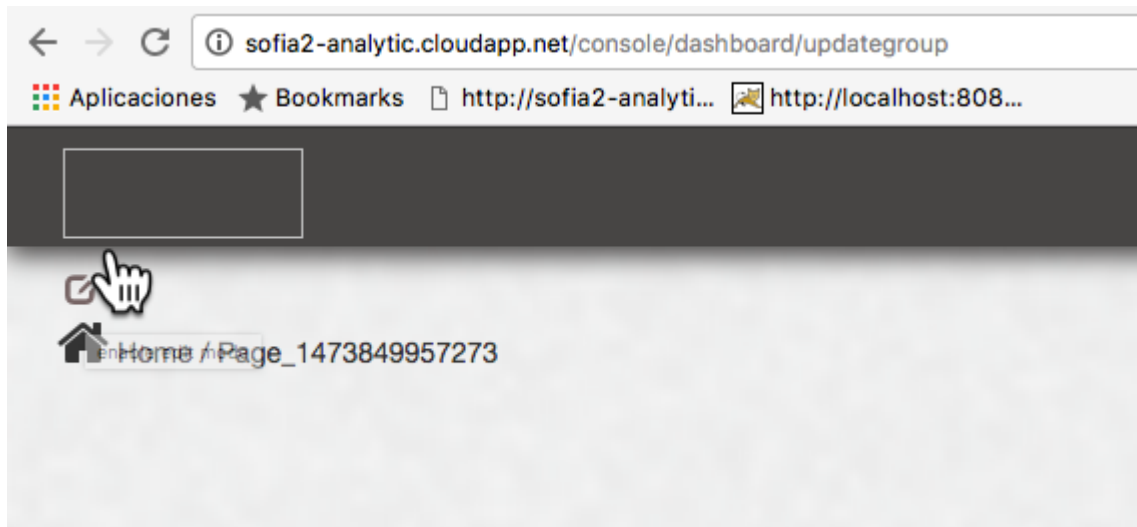
Style

Menu

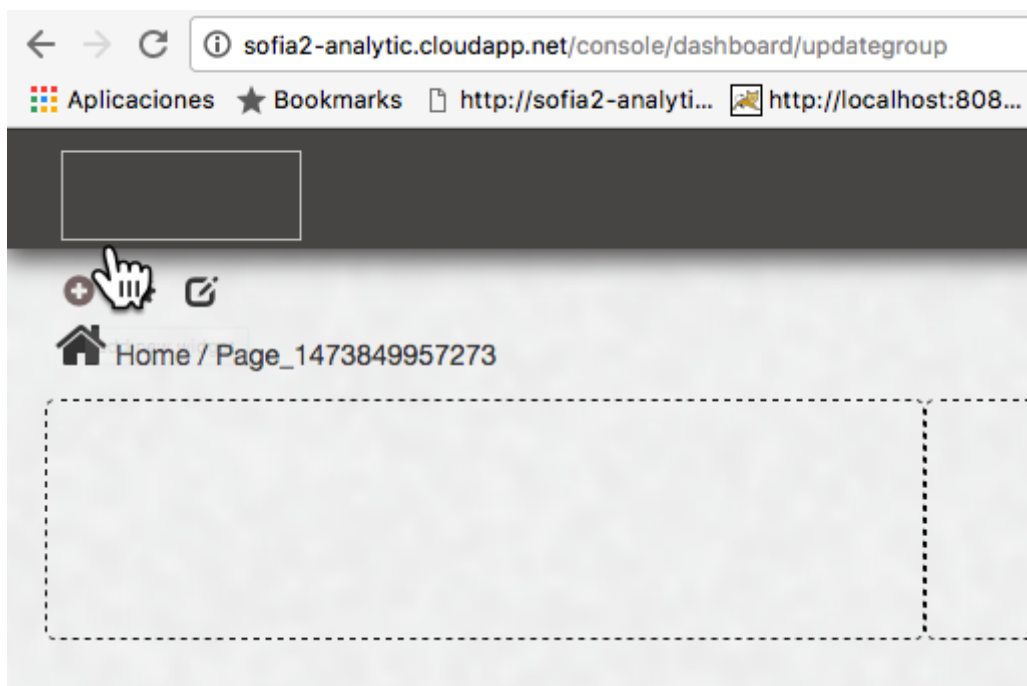
EXISTING PAGES IN THE CURRENT DASHBOARD

Name	Options
------	---------

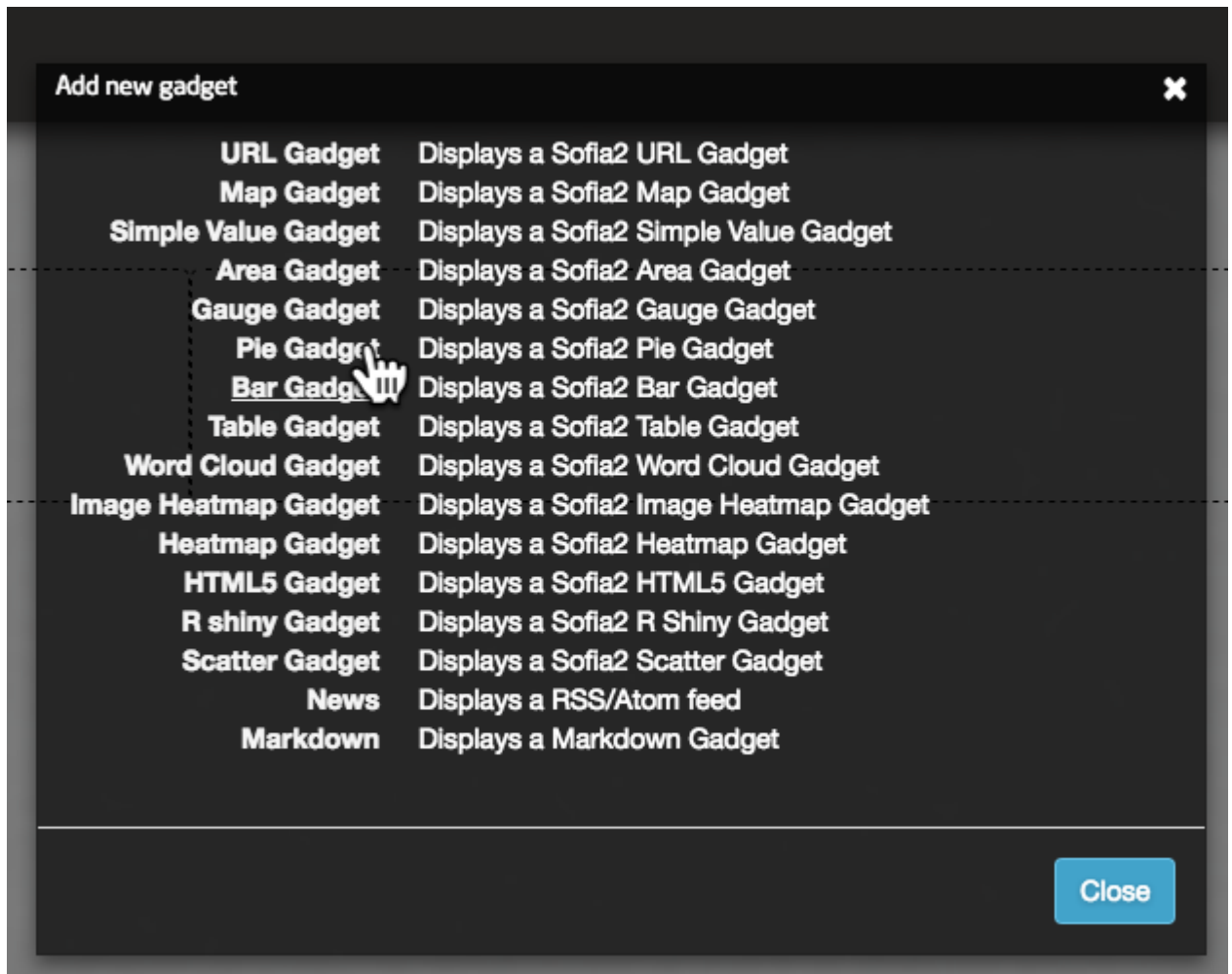
We enable editing mode.



Click on the + symbol that will allow us to add a new Gadget.



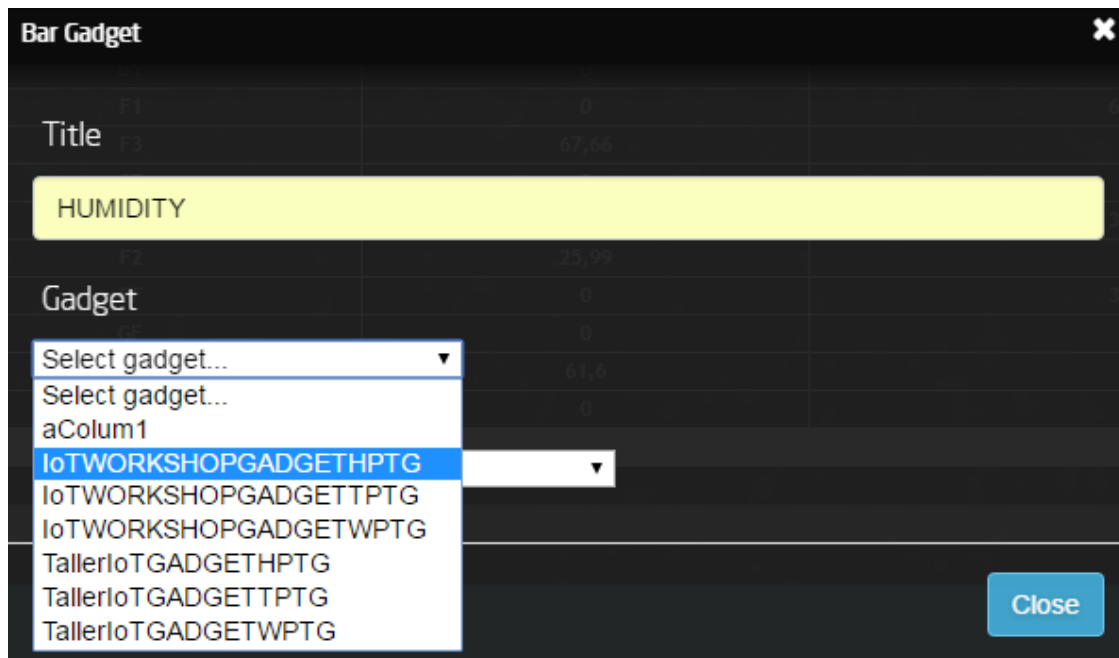
Select the Gadget type we want to add, in our case are three of Bar type and one of Table type.



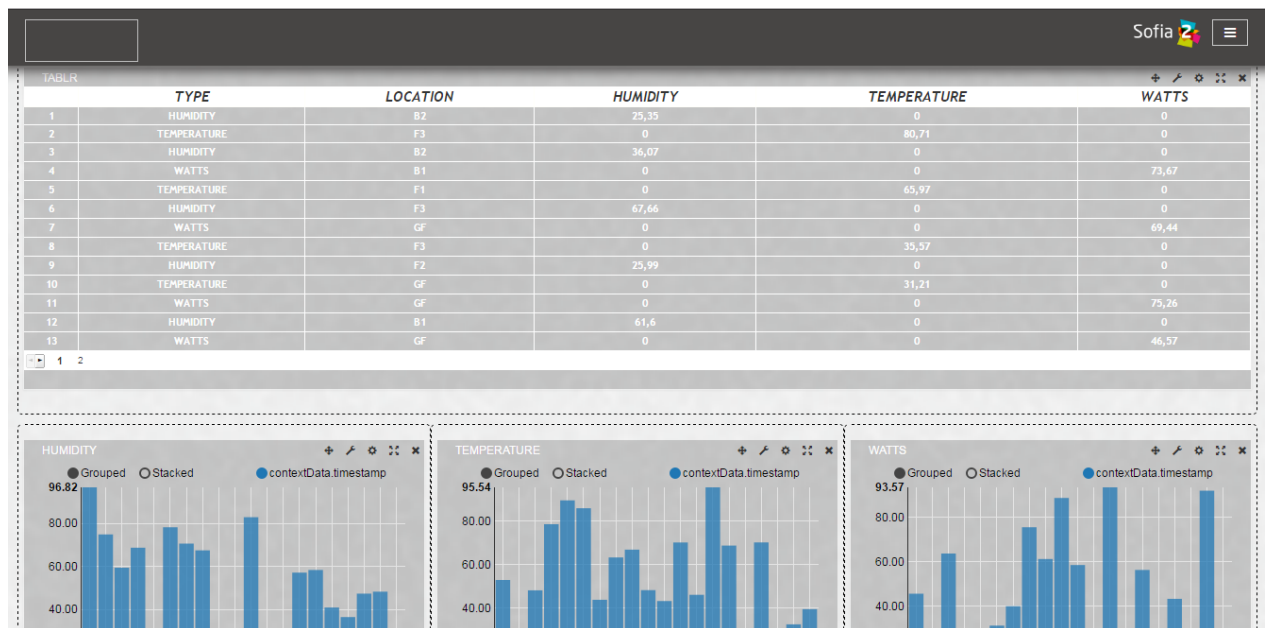
Once the Gadget type is added, click on the Settings button.



Select the Gadget we want to add to our Dashboard.



The result will be the Dashboard with all the gadgets we have added.



In the Gadget listing, if we click on the link symbol,

VISUALIZATIONS / MY DASHBOARDS

New Dashboard

Name	Owner	Public	Options
aDashPrueba1		<input type="radio"/>	Link Eye Edit Delete
IoTWORKSHOPDASHBOARDPTG		<input checked="" type="radio"/>	Link Eye Edit Delete

Showing 1 to 2 of 2 entries

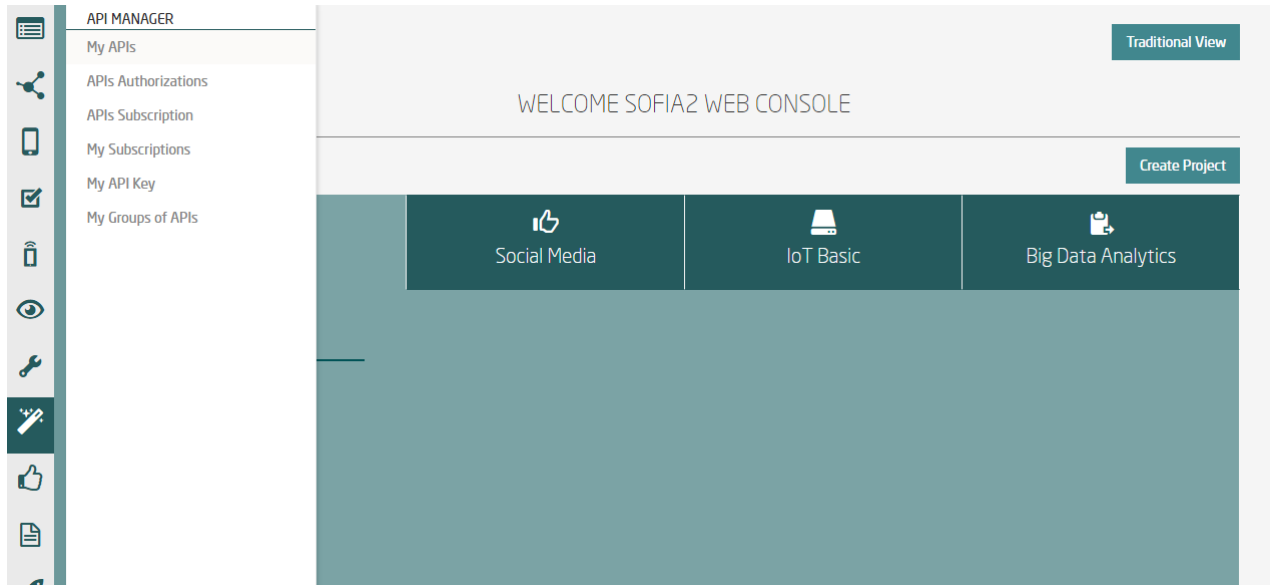
Show Dashboard URL

FirstPrevious1NextLast

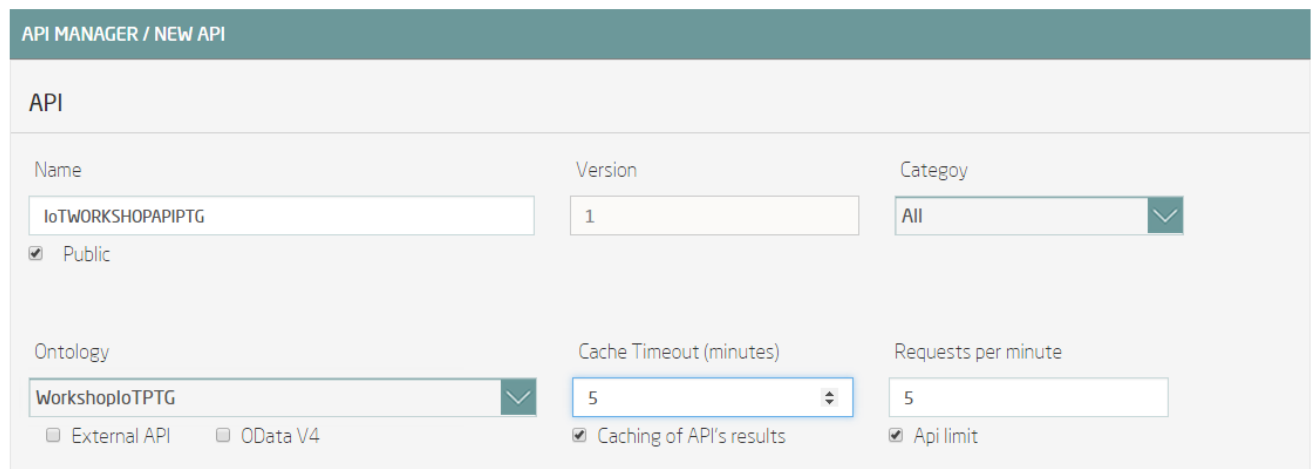
We have a dialog box with an URL where we can access and publish directly to the Dashboard.

6. Publish Ontology as API

Sofia2 allows us to publish our ontologies as Api RST, to do this we have to access to 'My APIs' menu.



Click on 'New API' button. We write the name IoTWORKSHOPAPI + Our initials, and we check it as public. Uncheck the option 'External API' and select our ontology.

The screenshot shows the 'API MANAGER / NEW API' form. It has fields for 'Name' (IoTWORKSHOPAPIPTG), 'Version' (1), and 'Category' (All). There is a checked checkbox for 'Public'. Below these are fields for 'Ontology' (WorkshopIoTPTG), 'Cache Timeout (minutes)' (5), and 'Requests per minute' (5). At the bottom, there are checkboxes for 'External API' (unchecked), 'OData V4' (unchecked), 'Caching of API's results' (checked), and 'Api limit' (checked).

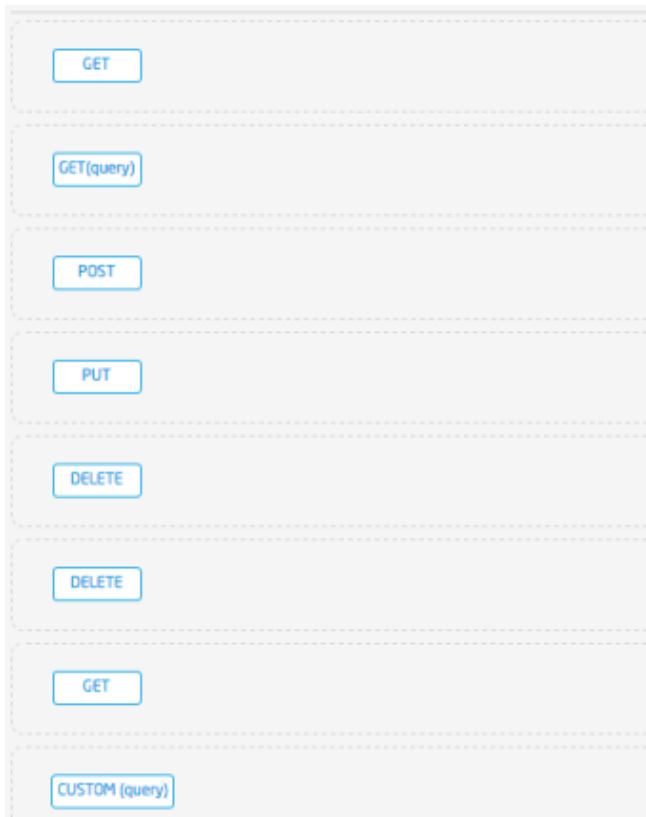
Let's set a 5 minute cache for query results. And a limit of 5 API queries per minute.

We can see the EndPoint API access:

Base EndPoint

<http://sofia2.com/sib-api/api/v1/IoTWORKSHOPAPIPTG>

We must write a description and a value for the Meta-Inf field. Finally we can see the operations we can create.



GET
GET(query)
POST
PUT
DELETE
DELETE
GET
CUSTOM (query)

Let's create three CUSTOM, one for each type of data we store: Humidity, Temperature and Watts.

Custom Query Operation

Method

GET

Name

Temperature

Query

select * from WorkshopIoTPTG where WorkshopIoTPTG.TYPE = 'TEMPERATURE'

QUERY PARAMETERS

QUERY CONFIGURATION

Query Type

SQLLIKE

Target DB

BDTR

Result Format

JSON

Description

Temperature query

POST-PROCESSING

Save

Cancel

We should obtain the following three APIs.

CUSTOM (query)	Temperature \\Temperature select * from WorkshopIoTPTG where WorkshopIoTPTG.TYPE = 'TEMPERATURE' Temperature query	Delete Edit
CUSTOM (query)	Humidity \\Humidity select * from WorkshopIoTPTG where WorkshopIoTPTG.TYPE = 'HUMIDITY' Humidity query	Delete Edit
CUSTOM (query)	WATTS \\WATTS select * from WorkshopIoTPTG where WorkshopIoTPTG.TYPE = 'WATTS' Watts query	Delete Edit
CUSTOM (query)		


We put the API as Published by clicking the 'Publish' button in the API listing.

API MANAGER / MY APIS		
▼ Filtrar Búsqueda		New API
	IoTWORKSHOPAPIPTG - V 1 Victor Vicente Created Api IoT workshop	Development Publish
	talleriotptges - V 1 Victor Vicente Deleted Api del talleriotptges	


Access to 'My API Key' menu, where we must copy the User Token, which we need to invoke APIs.

API MANAGER / MY API KEY	
User Token	
e0f0aa47e9ae4a0fb4452ec	Regenerate

Access to 'My Subscriptions' menu, where the APIs we have published will appear.

API MANAGER / MY SUBSCRIPTIONS 

 **IoTWORKSHOPAPIPTG - V 1** [Test & Doc](#)

 Published

Api IoT workshop

By clicking 'Test & Doc' we access to a page of API tests where, in the right part, appear the operations we have exposed.

\Humidity
\Temperature
\Watts

API INFO

Base path: <http://sofia2.com/sib-api/api/v1/IoTWORKSHOPAPIPTG>
Version: 1 - 2016-11-22

APIs

[IoTWORKSHOPAPIPTG](#)

Objects

IOTWORKSHOPAPIPTG

Api IoT workshop

\WATTS [GET](#)

\Temperature [GET](#)

\Humidity [GET](#)

When you click on each option, you'll see the meta service information. At bottom of the page we have the 'Submit' button. In the X-SOFIA2-APIKey header, paste the User Token that you copied in the previous point.

APIs

[IoTWORKSHOPAPIPTG](#)

Objects

\WATTS [GET](#)

Path	\WATTS
Description	Watts query
Method	GET
Produces	application/json
Consumes	application/json
Headers	X-SOFIA2-APIKey Token de autenticación

\Temperature [GET](#)

\Humidity [GET](#)

PLAYGROUND

\WATTS

Headers

X-SOFIA2-APIKey

X-SOFIA2-APIKey

Accept

☒ application/json

[Submit](#)

When we execute it we'll get the query result we had defined.

Al ejecutarlo obtendremos el resultado de la consulta que habíamos definido.

Accept
☒ application/json

Submit

Response text Response info Request info

```
{
  "data": "[ { \"_id\": { \"$oid\": \"582da3ebe4b05a-\" }, \"contextData\": { \"session_key\": \"803b5312-17d1-4986-944a-\", \"user\": \"\" } } ]",
  "error": null,
  "errorCode": null,
  "ok": true
}
```

At 'Request Info' tab we can see the invocation URL of the operation, which will be the End Point that was created when we generated the API and the operation.

Accept
☒ application/json

Submit

Response text Response info Request info

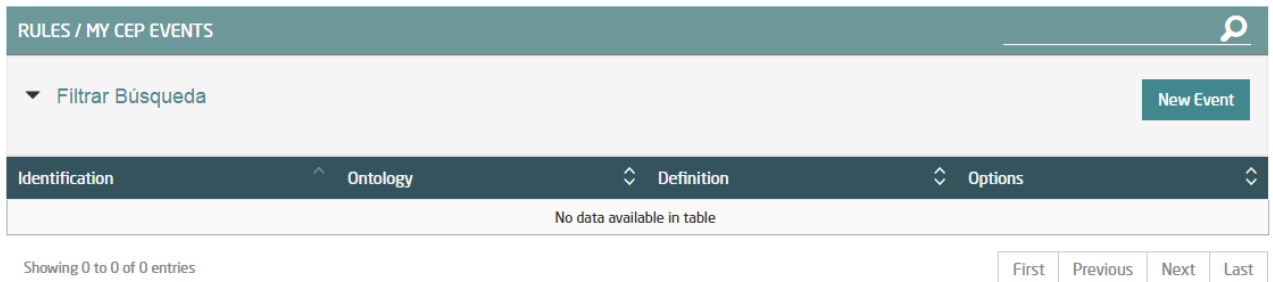
Request URL

```
http://sofia2.com/sib-api/api/v1/IoTWORKSHOPAPIPTG\WATTS
```

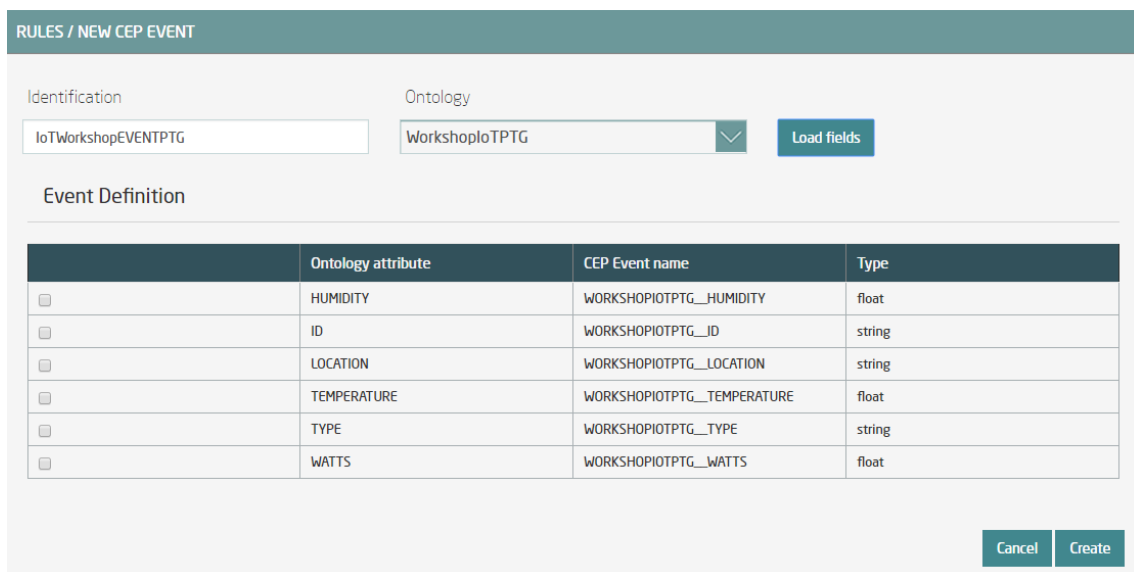
7. ANNEX

7.1 CEP Rule Creation

Access to 'My CEP Events' menu and click on 'New Event'.



As Identification we'll assign it 'IoTWorkshopEVENTPTG' + Our initials, we'll select our ontology and press the 'Load fields' button.



	Ontology attribute	CEP Event name	Type
<input type="checkbox"/>	HUMIDITY	WORKSHOPIOTPTG__HUMIDITY	float
<input type="checkbox"/>	ID	WORKSHOPIOTPTG__ID	string
<input type="checkbox"/>	LOCATION	WORKSHOPIOTPTG__LOCATION	string
<input type="checkbox"/>	TEMPERATURE	WORKSHOPIOTPTG__TEMPERATURE	float
<input type="checkbox"/>	TYPE	WORKSHOPIOTPTG__TYPE	string
<input type="checkbox"/>	WATTS	WORKSHOPIOTPTG__WATTS	float

Select 'TEMPERATURE', 'TYPE' and 'LOCATION' fields and press 'Create' button.

Let's look at 'CEP Event name' column, that will be the name we should use in the next point.

RULES / SHOW CEP EVENT

Identification


Ontology

Event Definition

Ontology attribute	CEP Event name	Type
LOCATION	WORKSHOPIOTPTG__LOCATION	string
TEMPERATURE	WORKSHOPIOTPTG__TEMPERATURE	float
TYPE	WORKSHOPIOTPTG__TYPE	string

[Cancel](#) [Create](#) [Modify](#) [Delete](#)

Now go to 'My CEP Rules' menu and click on 'New Rule' button.

RULES / MY CEP RULES 

▼ Filtrar Búsqueda [New Rule](#)

Identification ^ Query ^ Active ^ Options ^

Showing 0 to 0 of 0 entries

[First](#) [Previous](#) [Next](#) [Last](#)

Select the Event we created.

CEP Events

Availables

Selected

Search

In 'from' field we set the compliance parameters of the rule.

From

IOTWORKSHOPEVENTPTG [WORKSHOPIOTPTG__TYPE == "TEMPERATURE" and WORKSHOPIOTPTG__TEMPERATURE >30]

In 'select' we set the fields we want to retrieve when CEP rule is released.

Select

```
WORKSHOPIOTPTG__TEMPERATURE as VALUE, WORKSHOPIOTPTG__LOCATION as LOCATION
```

In 'Insert Into' we set the rule we want to generate, in our case WORKSHOPIOTRULE + Our initials. Once you have entered the three boxes, press the 'Create' button.

Insert Into

```
WORKSHOPIOTRULEPTG
```

We have already created a rule that will generate an event each time an ontology instance arrives with value 'TEMPERATURE' greater than 30 and is 'TEMPERATURE' type.

RULES / SHOW CEP RULE

Identification

```
WORKSHOPIOTRULEPTG
```

Active ☒

Query

```
from IOTWORKSHOPEVENTPTG [WORKSHOPIOTPTG__TYPE == "TEMPERATURE" and WORKSHOPIOTPTG__TEMPERATURE > 30]  
select WORKSHOPIOTPTG__TEMPERATURE as VALUE, WORKSHOPIOTPTG__LOCATION as LOCATION insert into WORKSHOPIOTRULEPTG
```

Cancel

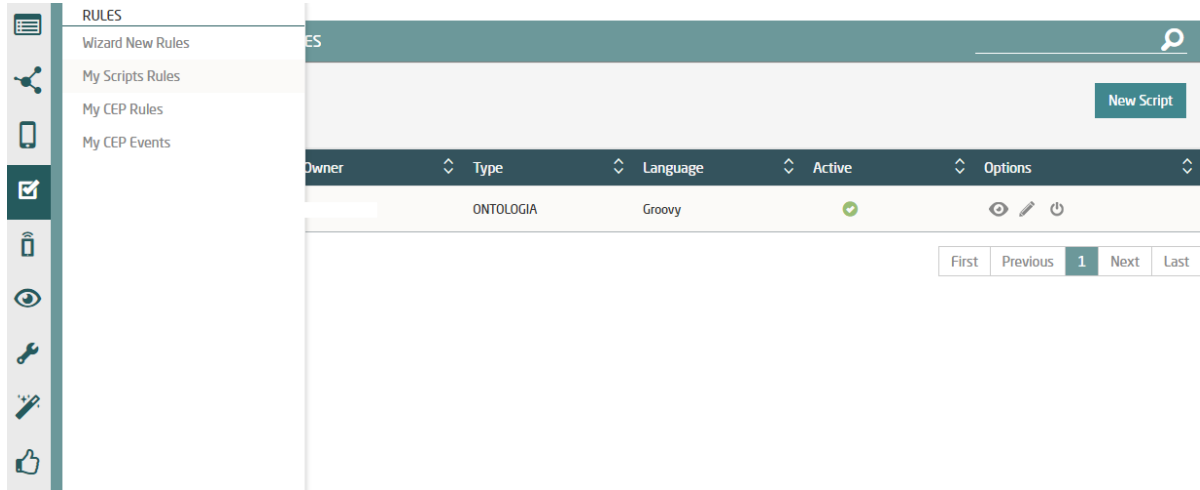
New

Modify

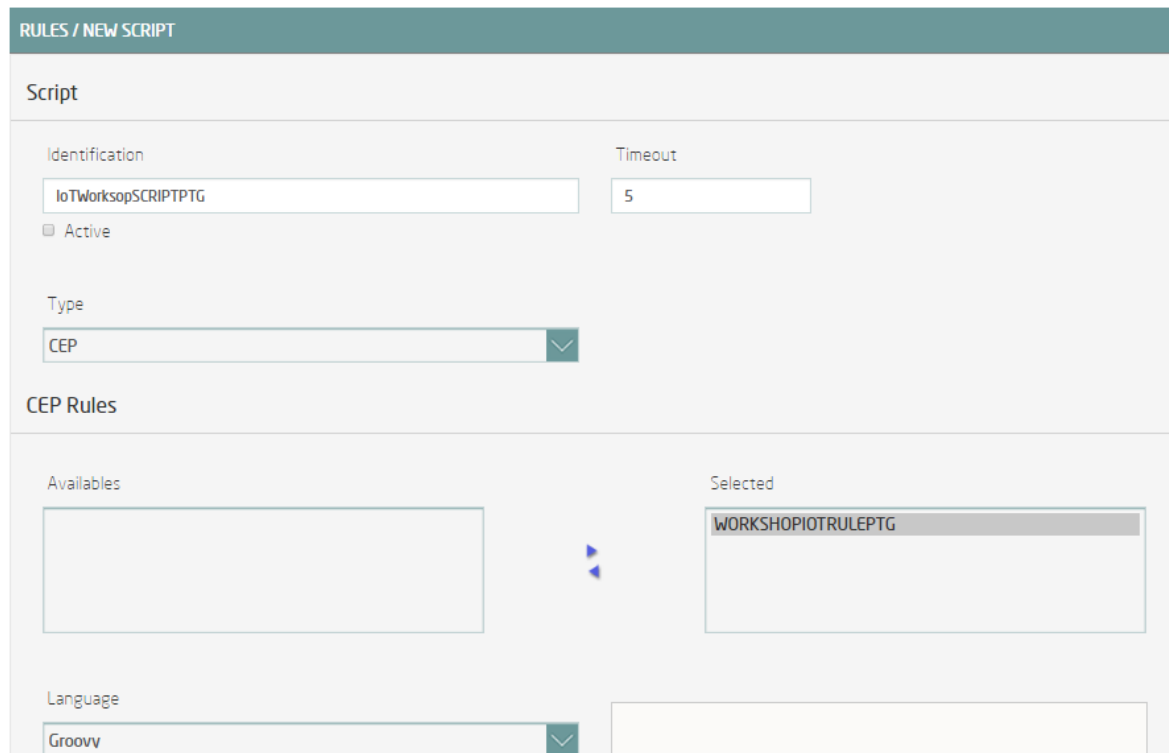
Delete

7.2 SCRIPT Rule Creation

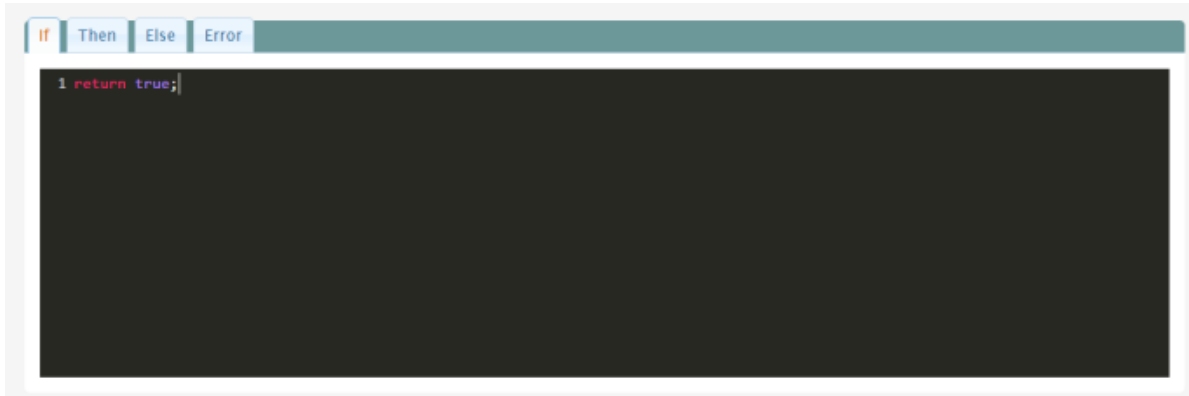
Access to 'My Script Rules' menu and click on 'New Script'.



We assign IoTWorkshopSCRIPT + Our initials to the Script the name. We assign a timeout of 5 seconds, choose the type of Script 'CEP' and select the rule we created before. Now, when the event associated with our rule is released, this script will be executed. Finally, we chose 'Groovy' language to the Script.

The screenshot shows the 'RULES / NEW SCRIPT' form. The form is divided into several sections. The 'Script' section contains an 'Identification' field with the value 'IoTWorkshopSCRIPTPTG', a 'Timeout' field with the value '5', and an 'Active' checkbox. The 'Type' section has a dropdown menu with 'CEP' selected. The 'CEP Rules' section is divided into 'Availables' and 'Selected' areas. The 'Availables' area is empty, and the 'Selected' area contains the rule 'WORKSHOPIOTRULEPTG'. At the bottom, the 'Language' section has a dropdown menu with 'Groovy' selected.

We force the execution of the 'Then' block by adding a return true in the 'if' evaluation block.

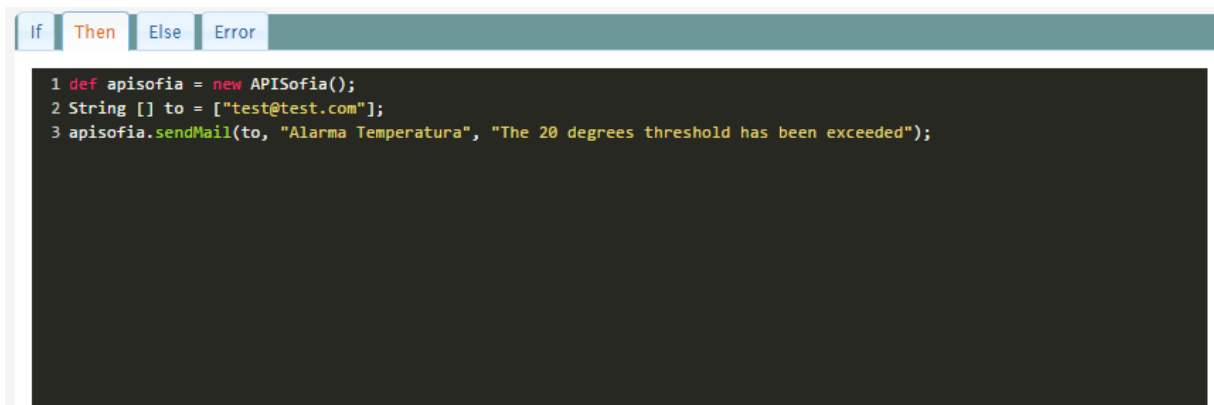


```
1 return true;
```

In the ['Motor Scripting Usage Guide'](#) we'll find more information about the use of Script and the APIs that it offers.

7.3 Final Exercise

In the 'then' block we'll add the logic we want to execute when the CEP events occur. The following code sends an email notifying us that we have exceeded 30 degrees.



```
1 def apisofia = new APISofia();
2 String [] to = ["test@test.com"];
3 apisofia.sendMail(to, "Alarma Temperatura", "The 20 degrees threshold has been exceeded");
```

If we want to retrieve the Event data, the projection we made through the 'select' clause of the CEP Rule, we have the 'InEvents' Object.

```
1 def apiutils = new APIUtils();
2 def jsonCepData = apiutils.getValueJson(cepData, "inEvents");
3 def valor = apiutils.getValueJson(jsonCepData, "VALUE");
```

And through the attribute 'getValuesJson' we can retrieve each of the attributes of the Event, which were 'VALUE' and 'LOCATION'.

To finish, I propose to create a new ontology, we'll call it IoTWORKSHOPALARM + Our initials, this should contain the fields LOCATION String and VALUE Number, both required.

We can use the same ThinkP we created at point 4 and also assign this ontology to it, and finally use the API Script to perform an insertion in the ontology Alarm when an event occurs.

Here's an example of how to insert an ontology from the Script Rules:

```
1 def apissap = new APISSAP();
2 def data = [
3     VALUE : 35,
4     LOCATION : 'S1'
5 ];
6 def json = groovy.json.JsonBuilder(data);
7 apissap.sendInsertMessage("TOKEN", "THINKP IDENTIFICATOR:FREE LITERAL", "ONTOLOGY NAME", json.toPrettyString());
```

In [Sofia2 Documentation](#) you have all the documentation of the platform.

The [Apis SCRIPT Guide](#) describes the APIs availability.

So far the IoT Workshop Sofia2. We hope you enjoyed it and we'll be waiting for you at the next workshop!