Sofia2

Release 1.0

Indra

May 08, 2017

Introducción a Sofia2

Aquí podrá encontrar toda la información y documentación necesaria sobre la plataforma Sofia2.

CHAPTER 1

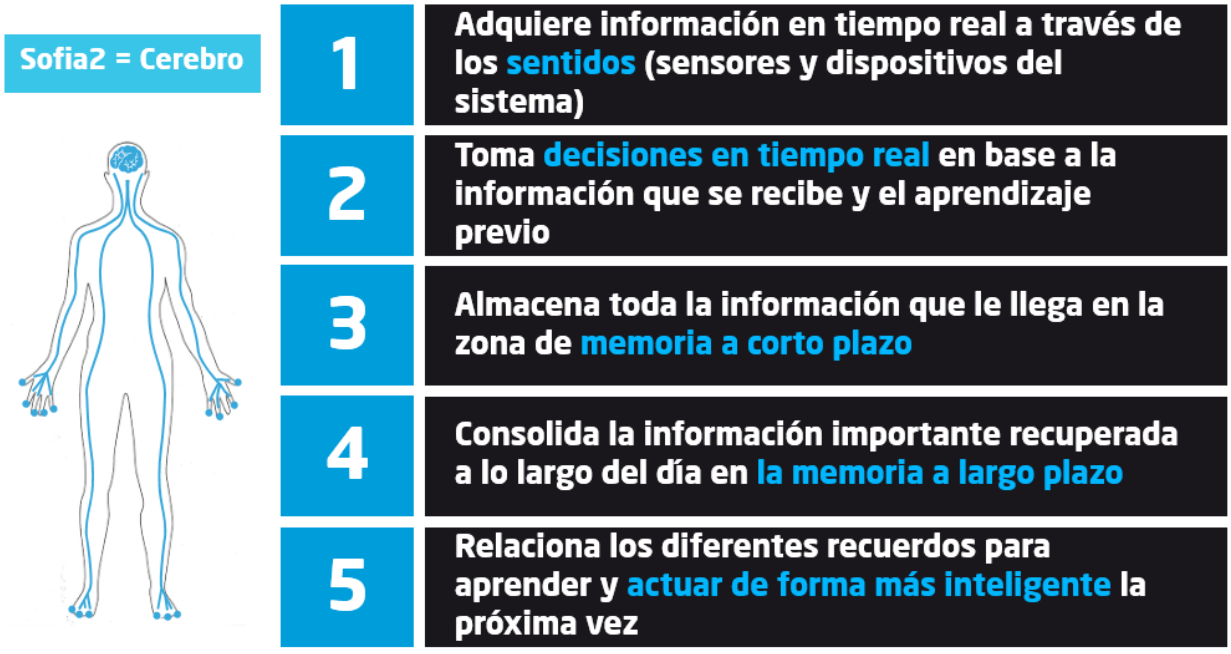
Qué es Sofia2

Es una **Plataforma IoT & Big Data** pensada para facilitar y acelerar la construcción de nuevos sistemas y soluciones digitales y así lograr la transformación y disrupción de los negocios.

Permite poner información real a disposición de aplicaciones inteligentes (**Internet of Things**).

Su propósito es lograr la interoperabilidad entre diferentes aplicaciones que comparten **conceptos semánticos.**.

Sofia2 (FEEP Enablement IoT Platform) forma parte de la Plataforma FEEP Enablement Platform de Indra/Minsait



Sofia2 como Smart Platform.

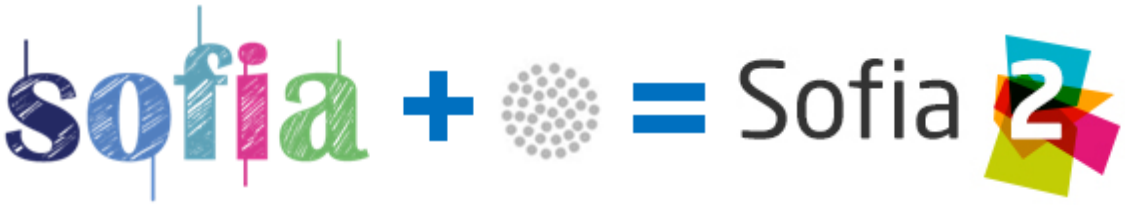
CHAPTER 2

Origen y Evolución

Sofia2 surge de un proyecto I+D europeo denominado SOFIA.

SOFIA es el acrónimo de **SMART OBJECTS FOR INTELLIGENT APPLICATIONS** y es una plataforma que surge de un proyecto de I +D Artemis de tres años finalizado en Marzo de 2012, en el que participan 19 partners de cuatro países de la UE, entre los cuales están Nokia, Philips, Fiat, Acciona e Indra.

Tras el proyecto Artemis Indra ha seguido evolucionando el proyecto SOFIA original creando una plataforma enfocada a su uso empresarial: **Sofia2**



Sofia2 se ha enfocado en estas áreas:

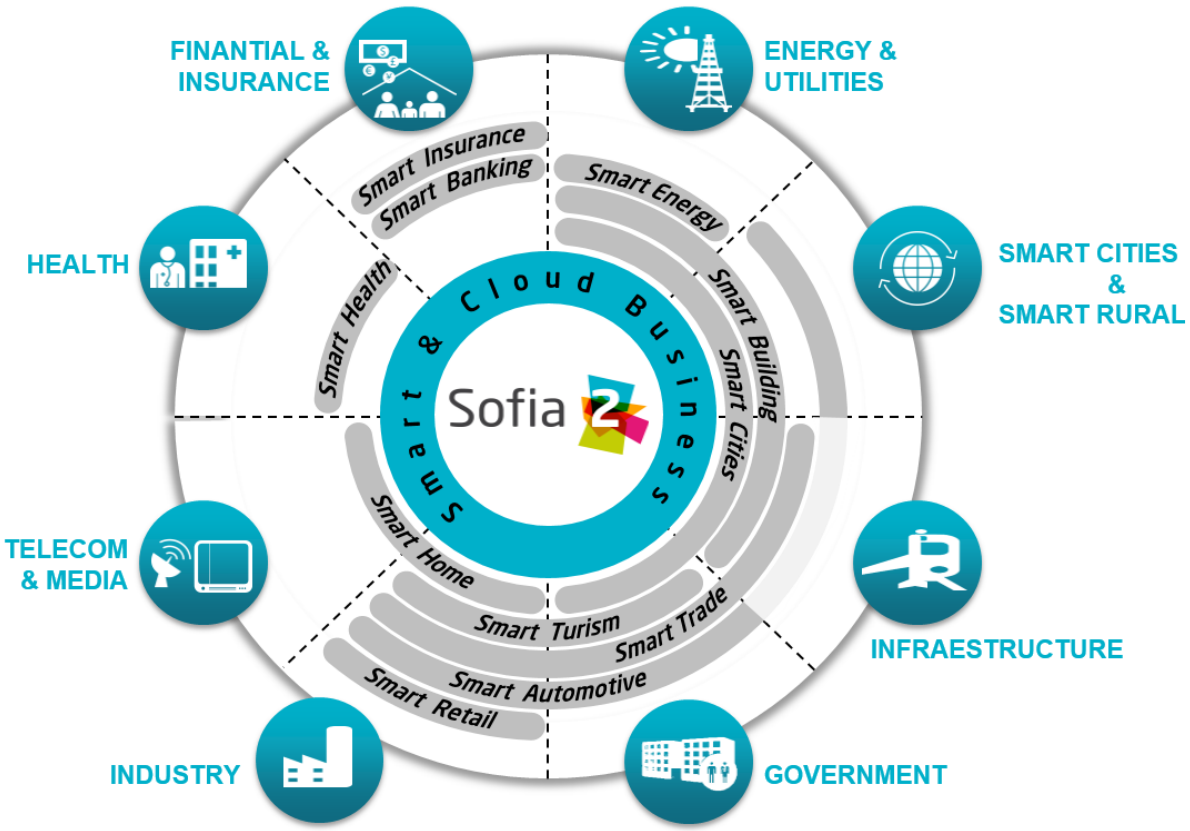
- **Adaptación al mundo empresarial**: funcionamiento en Alta Disponibilidad, con centros de datos distribuidos,...
- Se ha simplificado el trabajo con la plataforma, especialmente en estas áreas:
  - Desarrollo de las ontologías (pasando a ser ligeras).
  - Lenguaje de consultas.
  - Protocolo SSAP: con una implementación JSON además de la estándar XML.
- Interfaces Big Data (Hadoop) para almacenamiento de grandes volúmenes de datos y datawarehouse.
- Capacidades integración con Backends con protocolos estándares como Web Services,...
- Concepto de plugins para ampliar el SIB.
- Almacenamiento y Consultas GIS integradas.
- Inclusión de mecanismos de seguridad plugeables.

- Interfaces REST para poder conectar de forma simple desde smartphones, dispositivos, aplicaciones RIA,. . .

CHAPTER 3

Ámbitos de aplicación

Internet of Things se posiciona transversalmente cruzando distintos sectores y negocios. La plataforma Sofia2 ofrece **soluciones de negocio end-to-end** en todos los ámbitos que el cliente pueda ir incorporando conforme vaya necesitando.



Smart Cities.

Aportamos soluciones Smart City que sitúan al ciudadano en el centro de actuación pública y de gobierno.

Sofia2 como cerebro de OS de la ciudad:

- Recolección de datos de los sensores y dispositivos de la ciudad.
- Integrado con resto de sistemas de la ciudad.
- Evaluación de Reglas y Motor CEP para toma de decisiones en la ciudad.
- Suscripción a eventos, alarmas,...
- Soporte multidispositivo.

Sofia2 como cerebro de la ciudad (Pdf)

SmartCity: Un modelo de mobilidad sostenible para las ciudades del futuro

Smart Energy.

Aportamos soluciones Smart Energy para la gestión eficiente de los procesos de generación, distribución y comercialización.

SOFIA2 como Plataforma para dispositivos domiciliarios:

- Recoge información de dispositivos domóticos.
- Almacena, procesa y toma decisiones sobre grandes volúmenes de información.
- Gestión de los dispositivos domóticos.
- Soporta servicios adicionales sobre la plataforma domiciliaria.

Smart Health.

Aportamos soluciones Smart Health para un modelo de Salud sostenible, que prioriza la atención y la prevención de enfermedades.

SOFIA2 como Plataforma de Interoperabilidad entre Sistemas:

- Desplegable en dispositivos para salud domiciliaria.
- Funcionamiento como Bus de comunicación entre Sistemas de Salud.
- Gestión centralizada de las reglas, variables,...
- Almacenamiento de información histórica.

Industria 4.0.

Productos, procesos y modelos de negocio preparados para la Cuarta Revolución Industrial.

- Información enriquecida de equipos y procesos sobre dispositivos móviles, superponiendo información virtual sobre elementos del mundo real.
- Integración de múltiples dispositivos y sensores de manera inteligente.
- Algoritmos avanzados de mantenimiento predictivo, optimización de inventarios o gestión de rutas, para habilitar una óptima gestión de recursos.
- Funcionalidades de integración y colaboración en procesos logísticos, compras o facturación entre compañías.
- Funcionalidades de eficiencia energética para monitorizar y controlar el consumo energético de edificios e instalaciones industriales.
- Proyectos avanzados de I+D+i en curso para la fabricación de drones y vehículos autoguiados para entornos industriales.

Smart Retails.

Aportamos soluciones Smart Retail que conectan a las marcas con clientes, mejoran procesos logísticos, maximizan la adquisición ....

SOFIA2 en la optimización de modelos comerciales de tiendas:

- Gestión Smart de un supermercado permitiendo conocer el número de visitantes, analítica online de productos más solicitados,...
- Gateway en cualquier máquina vending o TPV permitiendo aplicar analítica para mejora de ventas.
- Gestión del SW.

Smart Banking.

Soluciones Smart Banking para desarrollar Banca Digital y liderar procesos de transformación de la información.

SOFIA2 encaja como Solución Transversal en Banca:

- Plataforma para el mobile payment e integración con otras empresas.
- Plataforma CEP para detección de fraude.
- Plataforma de gestión de logs: recepción centralizada, almacenamiento, explotación.

Soluciones Sofia2 (Pdf)

CHAPTER 4

Casos de Uso

Smart Cities: Coruña Smart City

Coruña Smart City es el proyecto de ciudad inteligente liderado por el Ayuntamiento de A Coruña y desarrollado en colaboración con Sofia2 (Minsait by Indra).

Coruña Smart City ofrece una nueva forma de acceder a la información de la ciudad.

Los ciudadanos pueden a través de canales electrónicos (ordenadores, teléfonos móviles), realizar tramitaciones administrativas o consultar informaciones útiles sobre la ciudad y sus servicios e intervenir en plataformas de participación.

Sofia2 es capaz de aportar beneficios en cuatro ámbitos principales: la gestión de la ciudad, la personalización de

los servicios para el ciudadano, la sostenibilidad y competitividad de la ciudad y la disponibilidad de información centralizada en un único sistema de control.

Acceder al portal

Plataforma Smart Coruña en acción

Smart Energy / Smart Home: ENDESA - PLATAFORMA MULTISERVICIOS ÍTACA.

- **Sofia2 como Plataforma de adquisición de datos y control** de los dispositivos ubicados en el domicilio del cliente.
- Plataforma como core de **eficiencia energética**.
- Comercialización por parte de ENDESA a **miles de hogares**.
- Integración de **dispositivos de diversos fabricantes** como KPS Sofia2.
- **Generación de Alarmas** que puede crearse el propio usuario desde la aplicación móvil.
- **Visualización y control del consumo** por parte del cliente de forma online a través de móvil.
- **Programación de horarios** de uso de energía aprovechando tarifas valle.
- **Almacenamiento y tratamiento de elevados volúmenes de datos** (terabytes) provenientes de elevados números de dispositivos (100.000).
- Funcionalidades específicas para particulares, PYMEs y Grandes Empresas.

Smart Energy para empresas: SGE ENDESA

- **Sofia2 como Plataforma de adquisición de datos y control** de los dataloggers ubicados en grandes empresas.
- Integración de **dispositivos de diversos fabricantes** como Carlo Gavazzi, Satel, Circutor y Telecon que realizan mediciones de energías, potencias, caudales de agua, consumo de gas...
- Comercialización por parte de ENDESA a grandes empresas como Carrefour, Banco Popular...
- **Generación de Alarmas** personalizadas con posibilidad de adaptarlas a la curva horaria de consumo.
- **Visualización y control del consumo**.
- **Programación de horarios** sobre los dispositivos.
- **Almacenamiento y tratamiento de elevados volúmenes de datos** con ejecución de algoritmos de recomendaciones.
- **Posibilidad de crear fórmulas** sobre las variables físicas en tiempo real por ejemplo para conocer los ratios de producción frente al consumo de energía


Dispositivos
NUEVOS DISPOSITIVOS
REFRESCAR LISTADO
Nombre dispositivo
Estancia
Equipamiento
Han abierto la puerta?
Entrada
Pinza Amperimétrica. General
cocina
Lampara Entrada
Entrada
0 W
On
Off
PlayStation y Televisión plana
Hab Beltrán
0 W
On
Off
Nuevo / Editar programa -
Nombre programa
Calefacción Salón
Dispositivo
Lampara Entrada
Activar programa
No se puede activar el programa. Este dispositivo ya tiene un programa activo. Desactívalo para activar uno nuevo
Establece la hora de encendido de los dispositivos
00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19 20 21 22 23 h.
L
M
X
J
V
S
D
CANCELAR
CREAR PROGRAMA

Smart Health: ZURICH - APLICACIÓN WEARABLES A SEGUROS

- **Adquisición de datos en tiempo reaL de la pulsera de actividad** (pasos diarios, calorías, tomas de tensión. . .) y alta de constantes en la web.
- **Cálculo online de indicadores: ranking usuario.**
- **Dashboard** para cliente y para usuario de negocio que ve información agregada.
- Agregación de información de diversas fuentes.
- Incorporación de **información de Redes Sociales**.
- **Aplicación móvil**.
- Dashboards personalizados por usuario y dashboards.
- **Almacenamiento y procesado de información masiva**, habilitando el análisis Big Data de la información.

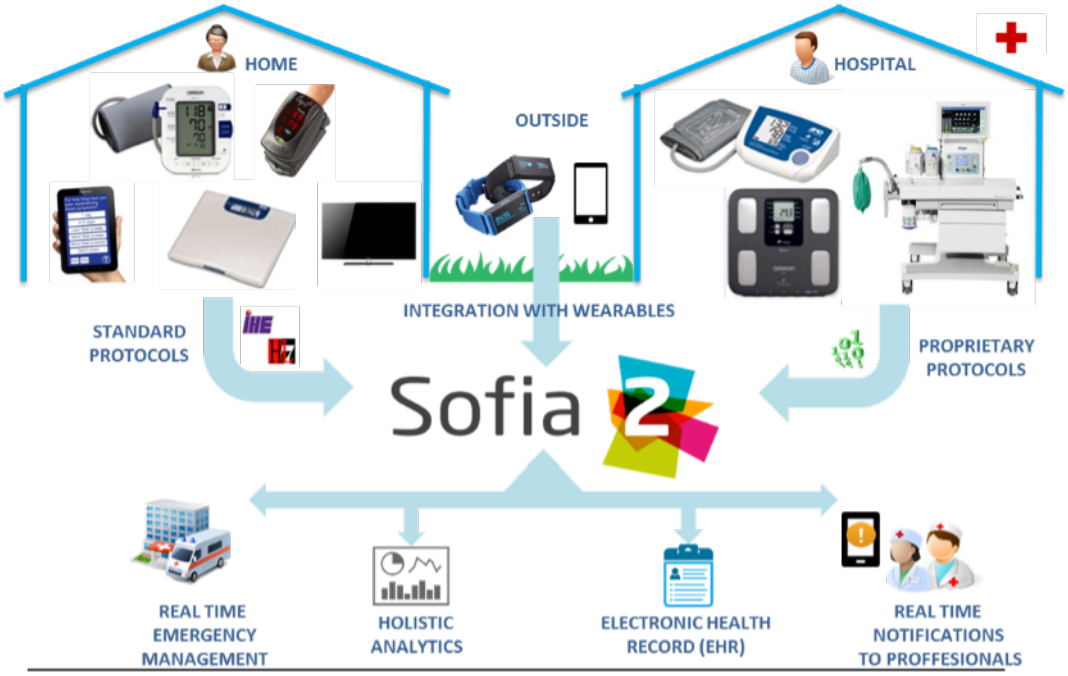
Smart Health: SERVIZO GALEGO DE SAÚDE - HOGAR DIGITAL ASISTENCIAL (TELEASISTENCIA)

El Proyecto de Hospitalización Experimental Hospital 2050, SISENS, desarrollado por Televés e Indra para el Servizo Galego de Saúde (SERGAS) consiste en un "concentrador inteligente", desarrollado y fabricado por Televés, conectado de manera ininterrumpida a SOFIA2.

SOFIA2 actúa como una plataforma centralizadora que recoge y analiza en tiempo real toda la información proveniente de los dispositivos implantados en el hospital, independientemente de sus características o fabricantes:

- Captación de información desde diversos dispositivos: pulsiómetro,electrocardiograma,tensiómetro,termómetro, báscula, glucómetro, coaguiómetro,body composit,peak flow monitor, cardiovascular, strength,independent,medication monitor...
- Monitorización de constantes vitales en remoto,evitando desplazamientos de médicos y pacientes con enfermedades crónicas/ hospitalización domiciliaria.
- Canal bidireccional de comunicación con el paciente, con calendario, alarmas, recordatorios, buenas prácticas a través de dispositivos móviles y Smart TVs.
- Sofia2 como Plataforma de recepción de señales biométricas con reglas y alarmas en base a valores anómalos informando a médicos y/o pacientes de situaciones de riesgo.
- Almacenamiento y procesado de información masiva, habilitando el análisis Big Data de la información biométrica de los pacientes del Sergas en futuros estudios.
- Smart room en el hospital.


Weight
Monthly
Weekly
100
75
50
25
2014 W38
Your weight (kg): 69.36
Fabian (kg): 96.68
Martin (kg): 78.54
Iniciar sesión
Zurich
Logout
Members (New)
Members
150
53%
44
65
80%
Distancia Recorrida (tiempo real)
Localización
Actividad
Diario
Semanal
Mensual
8596


HOME
OUTSIDE
HOSPITAL
STANDARD
PROTOCOLS
INTEGRATION WITH WEARABLES
PROPRIETARY
PROTOCOLS
Sofia 2
REAL TIME
EMERGENCY
MANAGEMENT
HOLISTIC
ANALYTICS
ELECTRONIC HEALTH
RECORD (EHR)
REAL TIME
NOTIFICATIONS
TO PROFFESIONALS

Sofia 2

CHAPTER 5

Características Generales

Características IoT

FEEP IoT Platform Sofia2, es un Middleware y repositorio que permite la interoperabilidad en tiempo real entre sistemas, redes sociales, dispositivos y sensores:

- Ontologías y visión sematica para garantizar independencia de protocolos. Permitiendo la representación del mundo físico en el mundo digital.
- Conectores de comunicación para diversos clientes y protocolos de comunicación ligeros (REST, WebSockets, MQTT, WS, JMS, AMQP...).
- Extensible en Java (APIs, Protocolos, Plugins) y APIs de desarrollo de clientes proporcionados en diversos lenguajes.
- Procesamiento en tiempo real de la información intercambiada.

- Auditoría de la actividad de integración.
- Configuración de reglas sencillas y complejas ejecutadas en tiempo real.
- Gestión y configuración integrado en Sofia2 Control Panel (HTML5).
- API Manager integrado basado en estándares (JSON, REST, RESTful) que incluye control completo del ciclo de vida de las APIs (Creada, en Desarrollo, Publicada, Deprecada, Eliminada), versionado.
- Seguridad integrada con el resto de elementos de la plataforma (autenticación, autorización, cifrado, . . .)
- Publicación de datos independientemente del repositorio (tiempo real o histórico) y publicación en portales Open Data.
- Integración transparente de APIs de terceros.
- Control de Throtling (gestión del número de peticiones que podrá realizar cada usuario por minuto.)

Características Big Data & Analytic

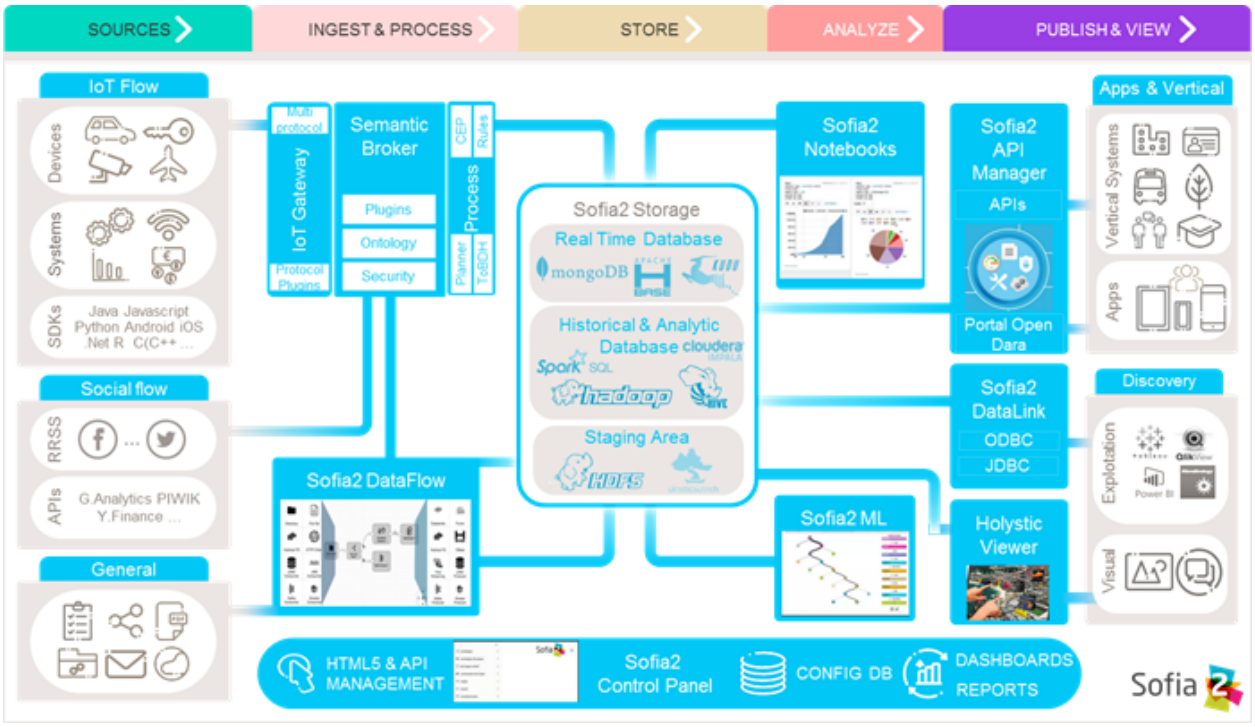
FEEP Big Data Platform Sofia2, permite la integración con fuentes de datos de forma visual y asistida y ofrece capacidades de analítica y Machine Learning que permite cargar datasets, lanzar algoritmos, crear modelos y publicarlos:

- Data flow, notebooks analytics y machine learning.
- Gestión de intérpretes (sh, jdbc, md, angular. . .).
- Ejecución multilenguaje sobre un mismo notebook (SparkSQL, R, Hive, Scala, Python) para la analítica de la información.
- Ejecución planificada de notebooks. Se pueden configurar reglas para capturar y visualizar datos de un pipeline en ejecución.
- Entorno compartido y multiusuario.
- Visualización instantánea con gráficas incorporadas.
- Entorno web integrado en panel de control.
- Parseo de datos en diversos formatos (ARFF, XLS, XLSX, CSV, SVMLight).
- Algoritmos: K-means, Generalized Linear Model, Distributed RF, Naïve Bayes, Principal Component Analysis, Gradient Boosting Machine y Deep Learning.
- Publicación de modelos.
- Además es posible consultar las estadísticas de ejecución de cualquier pipeline en tiempo real.

CHAPTER 6

Arquitectura global

Esta Vista describe la Plataforma desde el punto de Vista de los módulos funcionales que componen la Plataforma:



Módulos IoT

Éstos son los módulos necesarios para dar soporte a sistemas IoT:

- **SDKs**: La plataforma provee un set de herramientas para desarrolladores que facilita el desarrollo de clientes de (emisores y receptores de información) en diferentes lenguajes, y sobre una variedad de protocolos disponibles:
  - Lenguajes: Java, Javascript, Android, IOS, Python, Node.js, Arduino, C, .NET...
  - Protocolos: MQTT, MQTTS, REST, Websockets, WS...

- **Sofia2 Control Panel**: La Plataforma ofrece una completa web de administración/configuración que permite gestionar todos los conceptos que maneja la Plataforma. El resto de módulos de la Plataforma se operan/configuran desde este módulo, que persiste su configuración en la BDC (Base Datos Configuración) del Sofia2-Repository. Esta consola es accesible para los diferentes roles de la Plataforma, permitiendo operar de una u otra forma en función de este rol.

- **IoT Gateway**: capa de abstracción del protocolo de comunicación, que implementa el protocolo SSAP (Smart Space Access Protocol), sobre diferentes protocolos (MQTT, MQTTS, Websockets, WS, REST) y facilita la incorporación de nuevos protocolos gracias al despliegue de nuevos Plugins. De esta manera, la información gestionada por las subsiguientes capas de la plataforma es completamente agnóstica del protocolo tecnológico usado para el envío del dato, dando lugar a su gestión desde un punto de vista semántico.

- **Semantic Broker**: módulo de la Plataforma que recibe, procesa y almacena toda la información de las aplicaciones, sensores y dispositivos conectados, actuando como Bus de Interoperabilidad. Esta capa validará la corrección sintáctica y semántica del dato recibido gracias a la definición previa de la estructura del dato esperado (ontología), identificando de qué dato trata, y aplicando la seguridad correspondiente al mismo. Mediante el despliegue de plugins se podrá ampliar o adaptar la funcionalidad por defecto de este componente de una manera sencilla.

- **Process**: módulo que incluye 2 motores para la definición de reglas a aplicar sobre la información que entra en la Plataforma: el motor de Reglas (Sofia2-Rules) y el motor CEP (Sofia2-CEP).

- **Sofia2 Storage**: módulo de almacenamiento de la información de la plataforma. Se compone de 3 repositorios:
  - Base de Datos Tiempo Real (BDTR)
  - Base de datos Histórica (BDH)
  - Area de Staging (HDFS)

- **Sofia2 API Manager**: permite publicar la información gestionada por la plataforma como APIs REST y a su vez permite la búsqueda de estas APIs, la suscripción por parte de clientes y la gestión del versionado y ciclo de vida de cada una de ellas. Además este API Manager permite disponibilizar Servicios REST externos a la Plataforma, lo que permite ofrecer un punto único de acceso a APIS internas y externas de la Plataforma.

- **Holystic Viewer**: módulo de visualización avanzada de la Plataforma, que soporta diferentes motores. Se trata de un sistema integral de visualización avanzada e interactiva que permite una gestión de información geolocalizada asociándola a un entorno de visualización tridimensional y multimedia. Sus capacidades de visualización avanzada resultan un plus de interés a la creación de cuadros interactivos.

Módulos Big Data

Estos módulos añaden capacidades avanzadas de procesamiento en tiempo real y de analítica Big Data sobre la plataforma:

- **Sofia2 DataFlow**: Módulo que permite definir un pipeline para la gestión de un flujo de datos desde el sistema de origen a los sistemas de destino, permitiendo definir de manera visual cómo transformar los datos a lo largo del camino. El diseño de este ETL sigue las siguiente reglas: un único modulo origen para representar el sistema de origen de la información, posibilidad de agregar múltiples procesadores intermedios para transformar los datos, y al menos un módulo de destino (pudiendo ser múltiples) para definir el grabado de la información.

- **Sofia2 Notebooks**: Permite realizar, de manera muy sencilla e interactiva, analítica sobre datos de fuentes muy variadas, incluidas las fuentes de datos de Sofia2. Se pueden realizar cargas de archivos desde HDFS a spark, cargar de datos en tablas hive, lanzar consultas o realizar un proceso complejo de machine learning mediante las librerías de MLlib de Spark. Este módulo posee la capacidad de combinar código Scala, SparkSQL, Hive, R, Shell, o muchos otros con contenido html o directivas reactivas de angular, permitiendo interacciones en tiempo real con una potente interfaz y todo ello en un entorno compartido y multiusuario.

- **Sofia2 ML**: Permite aplicar y modelar visualmente, de forma sencilla, diversas técnicas de aprendizaje, entre las cuales podemos destacar las siguientes:
  - Regression: Técnicas para estimar relaciones entre variables y determinar la importancia relativa de éstas en la predicción de nuevos valores.
  - Clustering: Técnicas para segmentar los datos en grupos similares.
  - Classification: Técnicas para identificar la pertenencia de un elemento a un grupo determinado.
  - Recommendation / Prediction: Técnicas para predecir el valor o preferencia de una entidad nueva basado en históricos de preferencias o comportamientos.

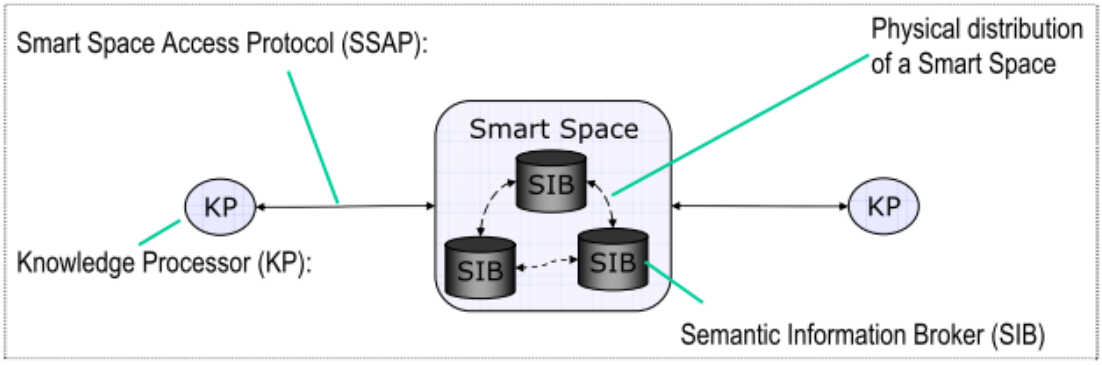
- **Sofia2 DataLink**: Actúa de interfaz con productos de analítica, ofreciendo conectores estándar JDBC, ODBC y REST y una capa de abstracción que permite operar a través de SQL independientemente del origen de los datos. De esta manera, se facilita la integración tradicional a nivel de datos, con los repositorios BDTR y BDH indistintamente, pudiendo incluso realizar consultas en las que se combine información de ambos.

CHAPTER 7

Conceptos de la Plataforma Sofia2

La Plataforma **Sofia2** se conceptualiza con estos 4 conceptos:

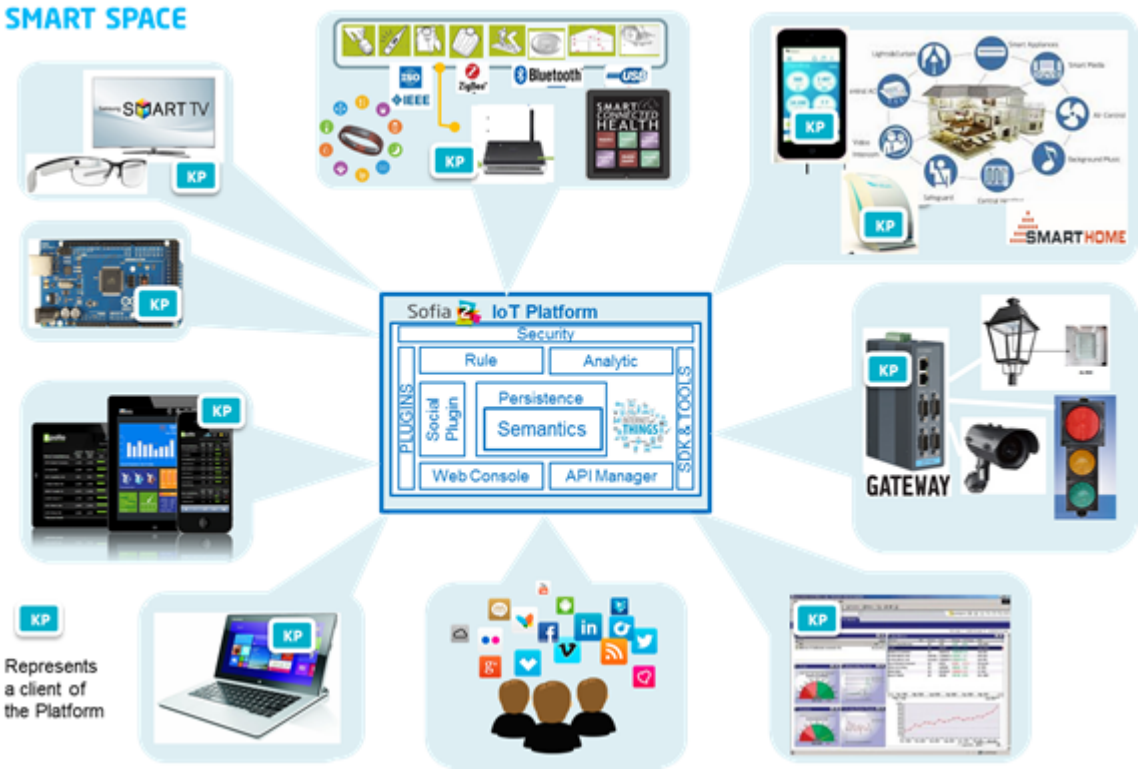
- **Smart Space**
- **SIB**
- **KP**
- **SSAP**



Smart Space

- Es el entorno virtual donde diferentes aplicaciones interoperan para ofrecer una funcionalidad compleja.
- El núcleo de un Smart Space es el **SIB**.

- En un Smart Space suele existir un único SIB (que puede ser un cluster), aunque en casos concretos pueden existir federaciones de SIBs.
- Los Smart Spaces pueden comunicarse entre ellos estableciendo relaciones de confianza.



SIB (Semantic Information Broker)

- Es el núcleo de la Plataforma.
- Recibe, procesa y almacena toda la información de las aplicaciones conectadas la plataforma SOFIA, actuando de Bus de Interoperabilidad
- En él se reflejan todos los conceptos existentes en el dominio (reflejados en las ontologías) y su estado actual (instancias particulares de ontologías).
- En SOFIA[2] se propone el uso de JSON para el intercambio de información (SSAP) y para la definición de las ontologías.

```
{
 "body":
 "{
    "query":
    "{
       SensorHumedad.medida: {$gt:18}
    }"
  }",
```

```
  "direction": "REQUEST",
  "ontology": "SensorHumedad",
  "messageType": "QUERY",
  "messageId": 121,
  "sessionKey": "88bf5ee7-84d4-4956-98a3-ff290222fd64"
}
```

- Existen implementaciones en diversos lenguajes y plataformas. Indra suministra un SIB JEE que corre sobre cualquier Servidor Web JEE (Tomcat, JBoss,...)
- Gateway soporta manejadores de transporte TCP/IP, HTTP, REST, Bluetooth y Zigbee
- **Ofrecer conectores** para comunicación desde diversos clientes:
  - REST: para clientes Javascript, smartphones,..
  - MQTT para comunicaciones bidireccionales y dispositivos limitados
  - Web Services/JMS para aplicaciones empresariales
  - Otros como Bluetooth, Zigbee,..
- SIB extensible a través de plugins.

KP (Knowledge Processor)

- Es cada una de las aplicaciones que interopera en el Smart Space a través del SIB.
- Cada aplicación trabaja con instancias de los conceptos relevantes del dominio (ontología) para la que están diseñada
- Implementaciones en diversos lenguajes como Java, Javascript, Arduino,...
- Hay 3 tipos de KPs:
  - **Producer**: KP que solo inserta información en el SIB.
  - **Consumer**:KP que solo recupera información del SIB.
  - **Prosumer**: KP que inserta y recupera información del SIB indistintamente
- En SOFIA^2 se propone el envío de mensajes SSAP en JSON que son más ligeros y adecuados a dispositivos embebidos.

SSAP (Smart Space Access Protocol)

- Es el lenguaje de mensajería estándar para comunicar entre los SIBs y los KPs.
- Lenguaje es independiente de la red subyacente (GPRS, 3G, WIFI, BlueTooth, HFC, Zigbee)
- Existen dos implementaciones:

  - **SSAP-XML**: formato XML (mayor ancho de banda)
  - **SSAP-JSON**: mensajes adaptados a este protocolo, pensado para comunicaciones con dispositivos móviles, navegadores,...

- Mensajes de 3 tipos:
  - **REQUEST**: petición, enviada desde el KP al SIB
  - **RESPONSE**: Respuesta, enviada desde el SIB al KP en respuesta a un mensaje de REQUEST.
  - **INDICATION**: Notificación, enviada desde el SIB al KP ante un evento al que el KP está subscrito.
- Las operaciones que se realizan entre el SIB y los KP son las siguientes
  - **JOIN:** conexión de un KP a un SIB (implica autenticación, autorización y creación de sesión en el Smart Space)
  - **LEAVE:** desconexión de un KP del SIB
  - **INSERT/UPDATE/DELETE**: permite a los KPs la inserción/actualización/borrado de información realizada sobre el SIB
  - **QUERY:** permite a los KPs recuperar información del SIB: Puede ir sobre la Base de Datos de Tiempo Real e Histórica.
  - **SUBSCRIBE**: permite a los KPs suscribirse a la ejecución de una consulta cada X segundos o bien al desencadenado de un evento en el SIB
  - **INDICATION**: resultado enviado por SIB a uno o varios KPs para resolver una suscripción
  - **UNSUBSCRIBE**: Da de baja una suscripción realizada
  - **CONFIG:** permite al KP solicitar la configuración asociada a su instancia.
  - Notificar cambios desde el SIB a suscriptores

Ontologías

Las **ontologías** son descripciones semánticas de un conjunto de clases, representan las entidades de mi sistema.

En Sofia2, estas ontologías están representadas en formato JSON-Schema, por ejemplo, una ontología que usa KP que representa a un sensor de temperatura sería la siguiente:

```
"SensorTemperatura":
{
    "coordenadaGps":
    {
     "altitud": 0,
     "latitud": 40.512274,
     "longitud": -3.675679
    },
    "identificador": "S_Temperatura_00001",
    "medida": 19,
    "timestamp": 1373887443001,
    "unidad": "C"
}
```

Estas ontologías JSON se dan de alta en la plataforma y tienen un esquema JSON que le permite validar si la información semántica enviada por el KP cumple las reglas de forma de dicha ontología:

El esquema JSON que cumple la ontología SensorTemperatura indicada en el ejemplo anterior es el siguiente:

```
{
 "$schema": "`**http://json-schema.org/draft-03/schema#** <http://json-schema.org/
↪draft-03/schema>`__",
 "title": "SensorTemperatura Schema",
 "type": "object",
 "properties":
 {
  "_id":
  {
    "type": "object",
    "$ref": "#/identificador"
  },
  "SensorTemperatura":
  {
    "type": "string",
    "$ref": "#/datos"
  }
 },
 "identificador":
 {
   "title": "id",
   "description": "Id insertado del SensorTemperatura",
   "type": "object",
   "properties":
    {
     "$oid":
      {
        "type": "string",
        "required": false
      }
    }
 },
 "datos":
  {
     "title": "datos",
     "description": "Info SensorTemperatura",
     "type": "object",
     "properties":
     {
      "identificador":
      {
        "type": "string",
        "required": true
      },
      "timestamp":
      {
        "type": "integer",
         "minimum": 0,
         "required": true
      },
      "medida":
      {
        "type": "number",
        "required": true
```

```
      },
      "unidad":
      {
        "type": "string",
        "required": true
      },
      "coordenadaGps":
      {
        "required": true,
        "$ref": "#/gps"
      }
    }
  },
  "gps":
  {
    "title": "gps",
    "description": "Gps SensorTemperatura",
    "type": "object",
    "properties":
    {
      "altitud":
      {
        "type": "number",
        "required": false
      },
      "latitud":
      {
        "type": "number",
        "required": true
      },
      "longitud":
      {
        "type": "number",
        "required": true
      }
    }
   },
  "additionalItems": false
}
```

Cuando una ontología es guardada en la BDTR, la plataforma le añade meta información relativa al contexto de uso de dicha ontología:

```
{
 "_id":
 {
    "$oid": "51e3dbd465701fd8e0f69828"
 },
 "contextData":
 {
   "session_key": "08bf50c8-6ea6-41dc-99ac-5d12a6f517a3",
   "user_id": 1,
   "kp_id": 9,
   "kp_identificador": "gatewaysensores",
   "timestamp": "1373887444356"
 },
 "SensorTemperatura":
 {
```

```
    "coordenadaGps":
     {
       "altitud": 0,
       "latitud": 40.512274,
       "longitud": -3.675679
     },
   "identificador": "S_Temperatura_00001",
   "medida": 19,
   "timestamp": 1373887443001,
   "unidad": "C"
 }
}
```

Como vemos en el contextData aparece la clave de sesión que ha establecido el KP con SIB, el identificador del usuario que usa el KP, el identificador del KP, el identificador de la instancia del KP conectada y una marca de tiempo en la que se insertó la información.

CHAPTER 8

Tecnologías de Referencia

-  como Plataforma de desarrollo de módulos de proceso:
  -  **y su ecosistema** como tecnología de soporte.
  -  para la ingesta de datos, analítica en tiempo real, procesamiento batch y exportación de datos.
  - **Despliegue estándar JEE** independiente de AppServer.

- **Motor CEP (Siddhi CEP)** para reglas sobre grandes volúmenes de eventos en los que interviene el tiempo.

- **Motor Scripting (Groovy, R y Phyton)** que permite definir reglas ante llegada de mensajes , soportando creación de nuevas operaciones (alarmas, notificaciones,...).

-  como DataGrid para soporte y comunicación entre módulos y HA SIBs.

-  como RealTime DB por su almacenamiento JSON, escalabilidad,...

-  como Historical DB, los datos que ya no son del tiempo real se pasan automáticamente según configuración a este repositorio:
  -  como datawarehouse.
  -  como motor de consultas online distribuido.

- **MQTT/WebSockets/REST/WS/...** como protocolos de comunicación con la plataforma.

- **Spring MVC + Thymeleaf + jQuery** como framework Web para el desarrollo de la consola web

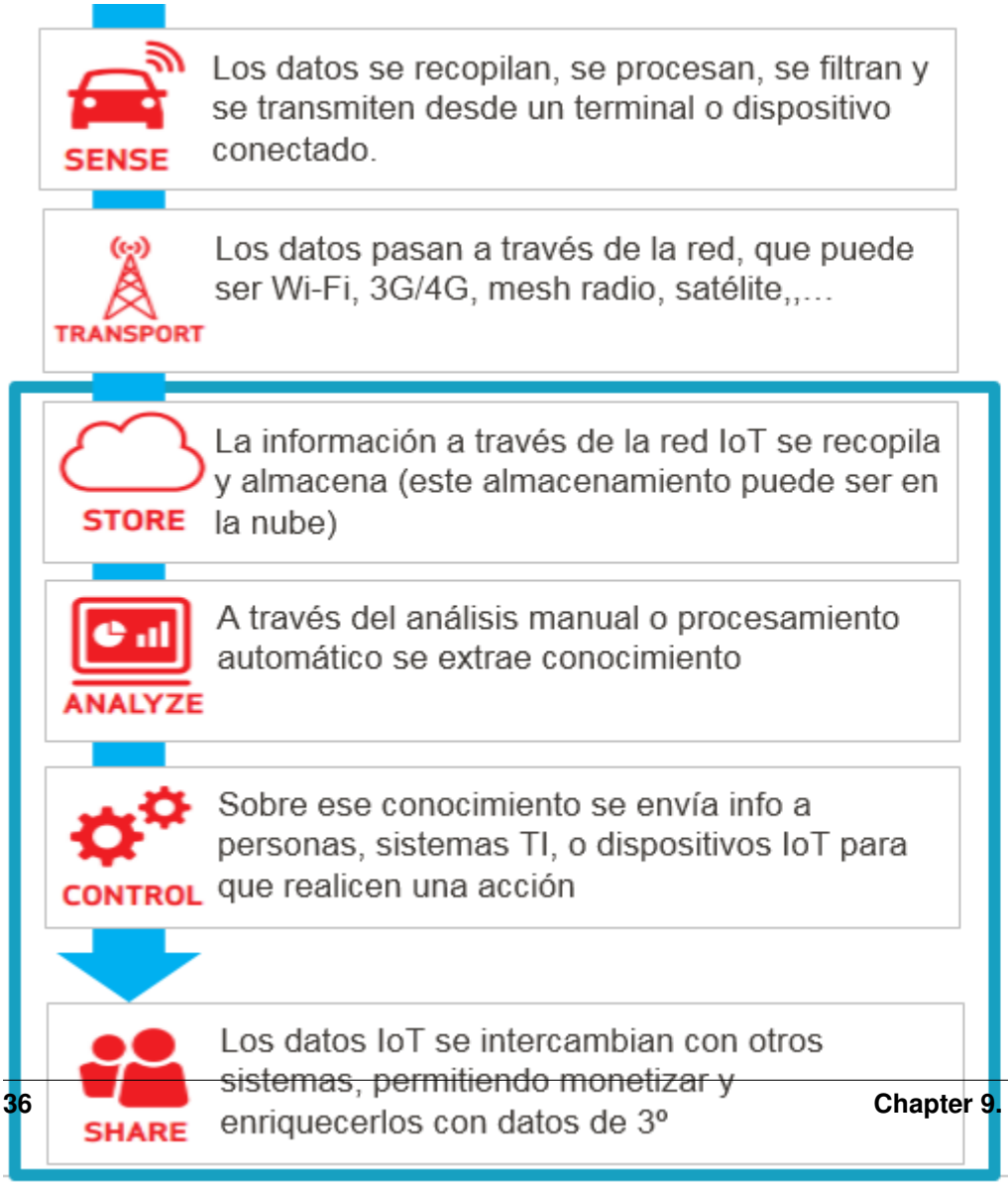


Con la modularidad de la solución se pueden llegar a sustituir o reemplazar algunas piezas si existe la necesidad. Por ejemplo BDH sobre Mongo, BDTR sobre Oracle,...


APACHE
DRILL
Scala
APACHE
HBASE
Zeppelin
Spark
Sofia 2

CHAPTER 9

Así trabaja Sofia2




Sofia 2

CHAPTER 10

Arquitectura Detallada

A continuación vamos a describir los módulos que componen la Plataforma sobre la vista modular con más detalle:

Módulos IoT:

SDK

IDE personalizado basado en Eclipse para el desarrollo simplificado de integraciones sobre la Plataforma.

Descargar

Sofia2 Control Panel

La Plataforma ofrece una **completa web de administración/configuración** desarrollada con tecnología HTML5 que permite gestionar todos los conceptos que maneja la Plataforma.

El resto de módulos de la Plataforma se operan/configuran desde este módulo, que persiste su configuración en la **BDC** (Base Datos Configuración) del Sofia2-Repository.

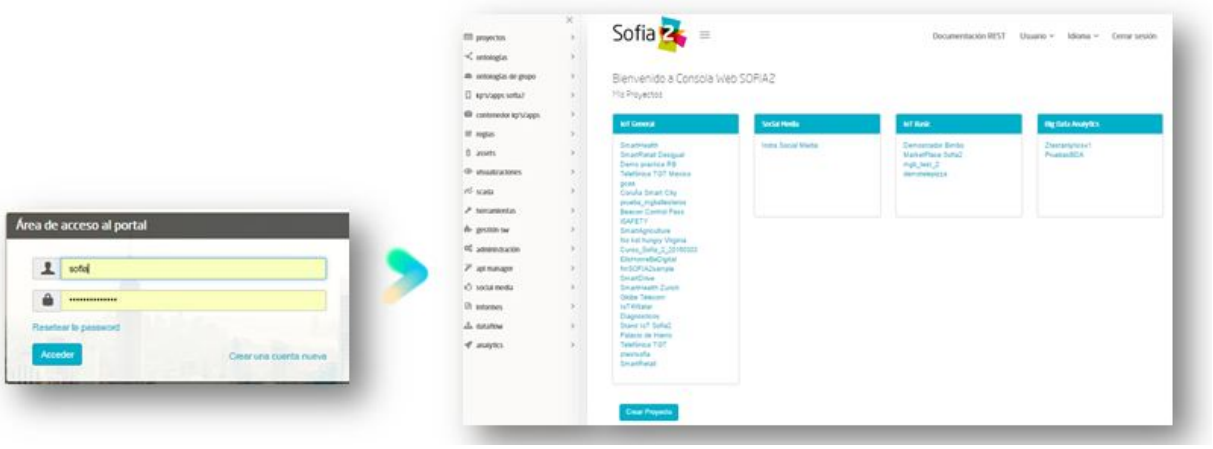
Toda la interacción con la Plataforma Sofia2 puede realizarse desde el Sofia2 Control Panel.

El panel de control es una herramienta que, mediante interfaces normalizadas, permite una representación estructurada de la información y un diseño de operación destinado a sacar el máximo rendimiento de los usuarios del sistema, facilitando el aprendizaje y reduciendo el tiempo de respuesta.

Generando proyectos de diferentes **Tipos de Proyectos** organizamos el trabajo que se realiza en la plataforma y tenemos una visión unificada de los conceptos que se manejan, además de permitir colaborar a diferentes usuarios en un proyecto.

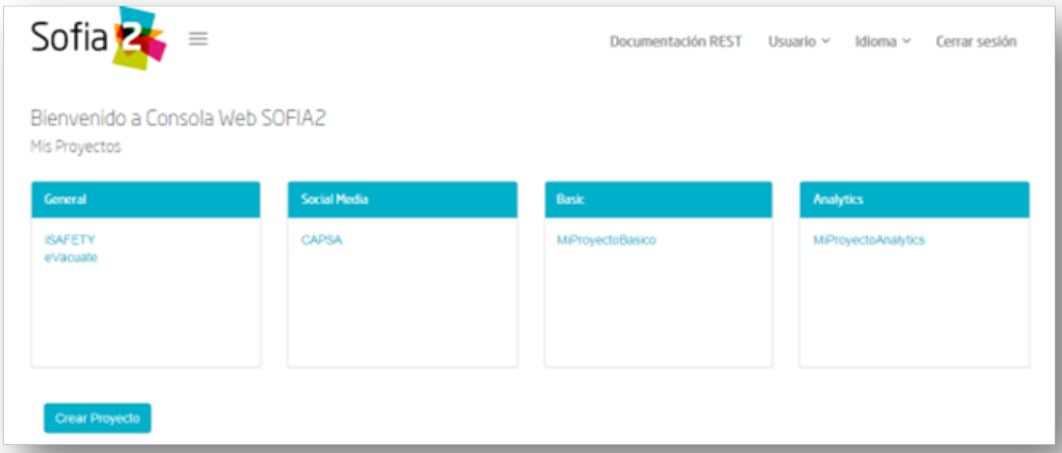
Existen tres tipos de Proyectos:

- **IoT Basic**: Proyectos que sólo necesitan acceso a las funcionalidad IoT básicas de la Plataforma.
- **IoT General**: Proyectos que necesitan del grueso de funcionalidades proporcionadas por la Plataforma.



- **Social Media**: Proyectos centrados en la analítica de Redes Sociales
- Big Data Analytics: Proyectos enfocados en el análisis, explotación, modelización, reporting,...

El Panel de Control se adaptará en función del tipo de proyecto seleccionado.



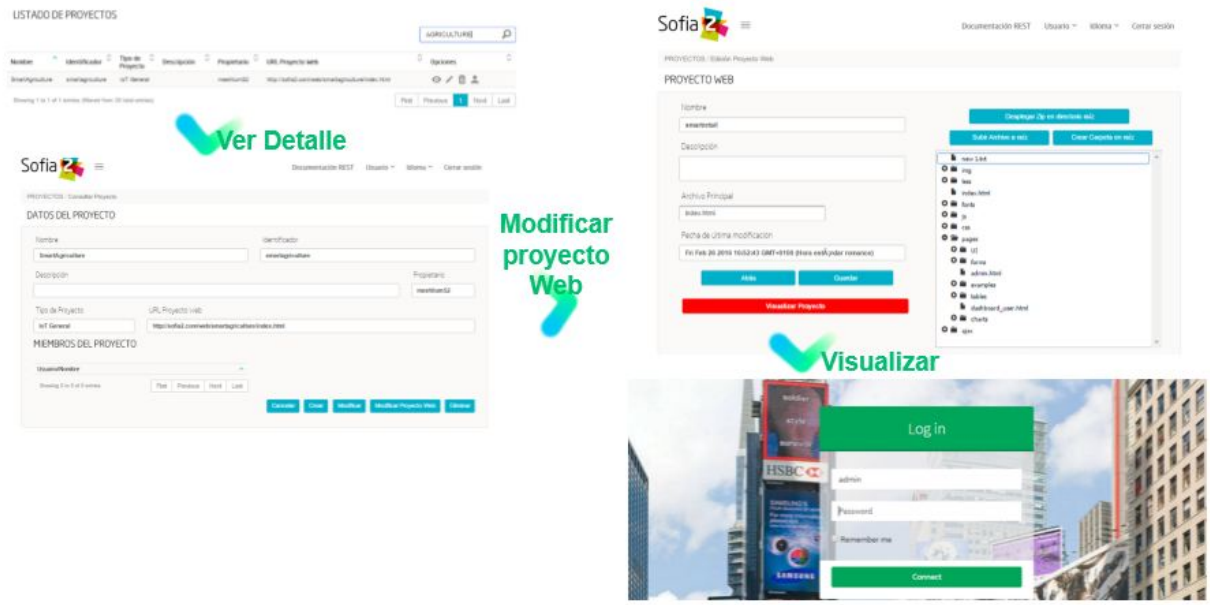
Asociado a los proyectos se puede crear una Definición de **proyectos Web asociados** a las ontologías e interfaces configurados en la plataforma. Edición y despliegue de recursos web (HTML5, css y Javascript):

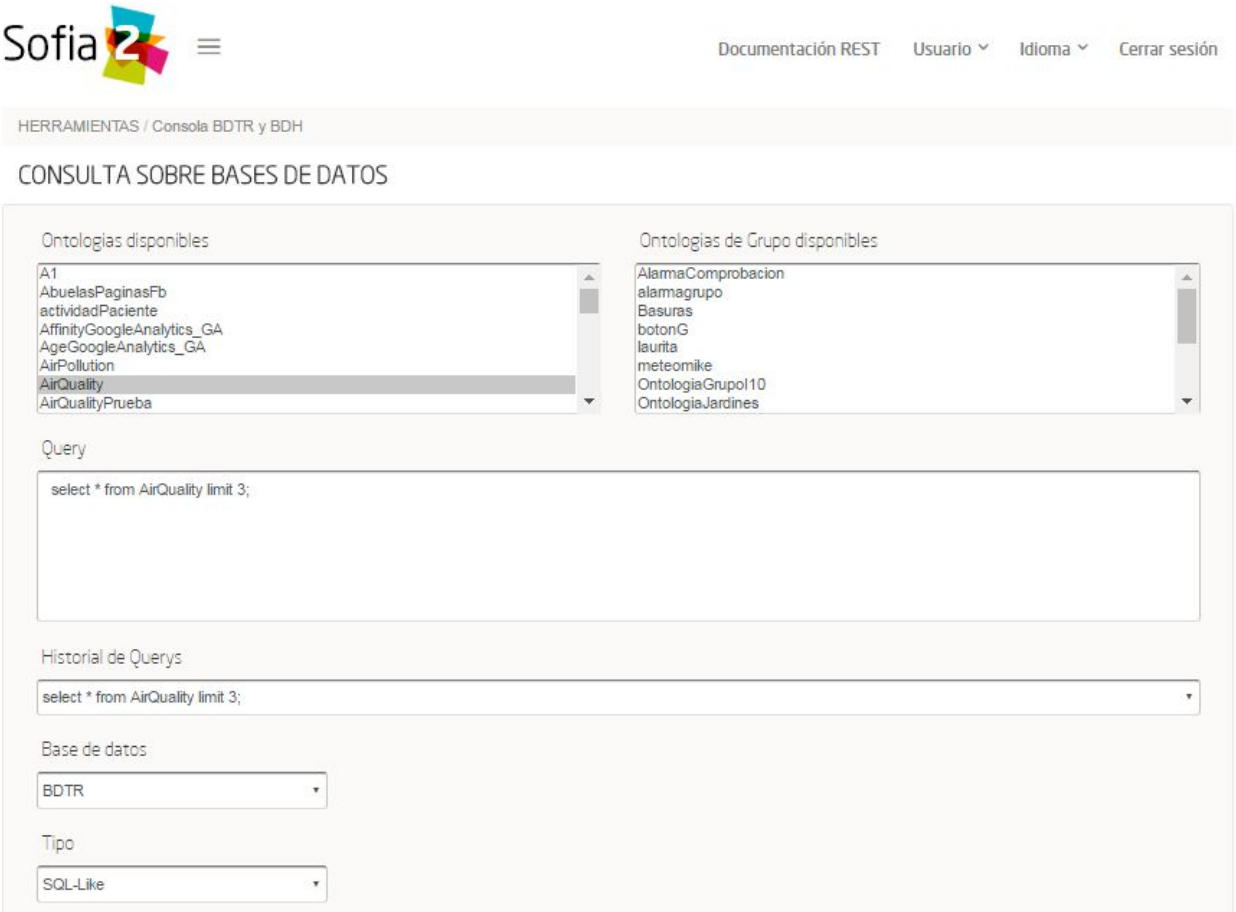
El Panel de Control provee a los usuarios (en función de los privilegios de su rol) funcionalidades como estas:

- **Herramienta de consulta sobre la BDTR y BDH**: permite acceder a los datos insertados en la plataforma a través de un motor de queries integrado.

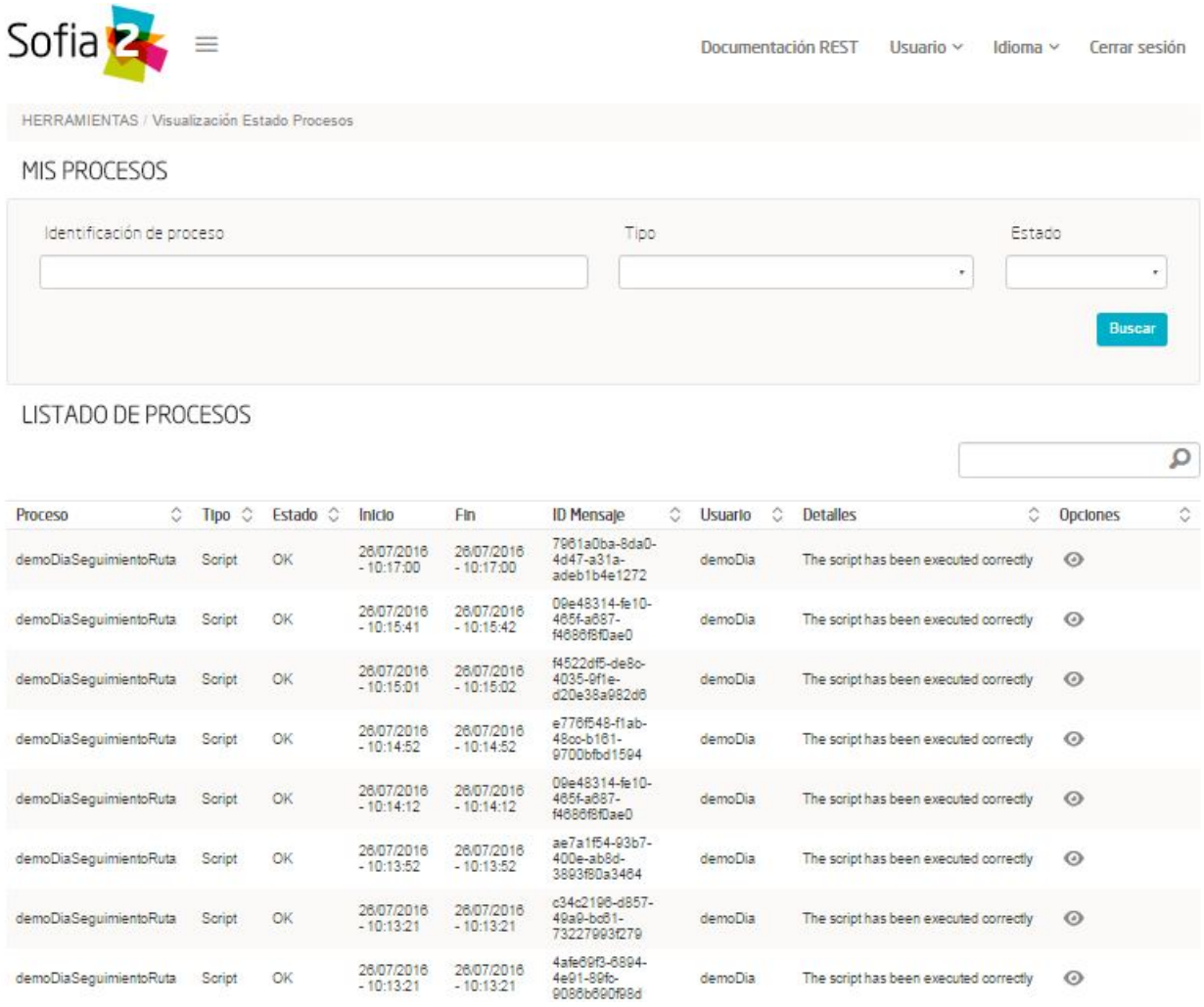
Esta interfaz posibilita lanzar queries en lenguaje SQL. Asimismo será posible el filtrado de los resultados (ej. por operador, identificador del jugador, tipo de juego, intervalo de fechas) mediante cláusulas where de SQL. El hecho de poder ejecutar sentencias SQL permitirá la actualización y borrado manual de registros a aquellos roles capacitados para tal fin.

- **Monitorización de procesos**: a través de la consola web es posible llevar el control sobre la planificación de procesos (como por ej. la carga de ficheros, el paso de datos a la BDH, etc.), permitiendo además su monitorización en tiempo real.


LISTADO DE PROYECTOS
Ver Detalle
DATOS DEL PROYECTO
MIEMBROS DEL PROYECTO
Modificar proyecto Web
PROYECTO WEB
Visualizar
Log in
admin
Password
Remember me
Connect


Documentación REST
Usuario
Idioma
Cerrar sesión
HERRAMIENTAS / Consola BDTR y BDH
CONSULTA SOBRE BASES DE DATOS
Ontologias disponibles
A1
AbuelasPaginasFb
actividadPaciente
AffinityGoogleAnalytics_GA
AgeGoogleAnalytics_GA
AirPollution
AirQuality
AirQualityPrueba
Ontologias de Grupo disponibles
AlarmaComprobacion
alarmagrupo
Basuras
botonG
laurita
meteomike
OntologiaGrupo10
OntologiaJardines
Query
select * from AirQuality limit 3;
Historial de Querys
select * from AirQuality limit 3;
Base de datos
BDTR
Tipo
SQL-Like

Esto posibilitará la monitorización del estado y funcionamiento de las cargas, cuánto han tardado, si ha habido algún error en ellas y cuál ha sido la naturaleza del mismo.



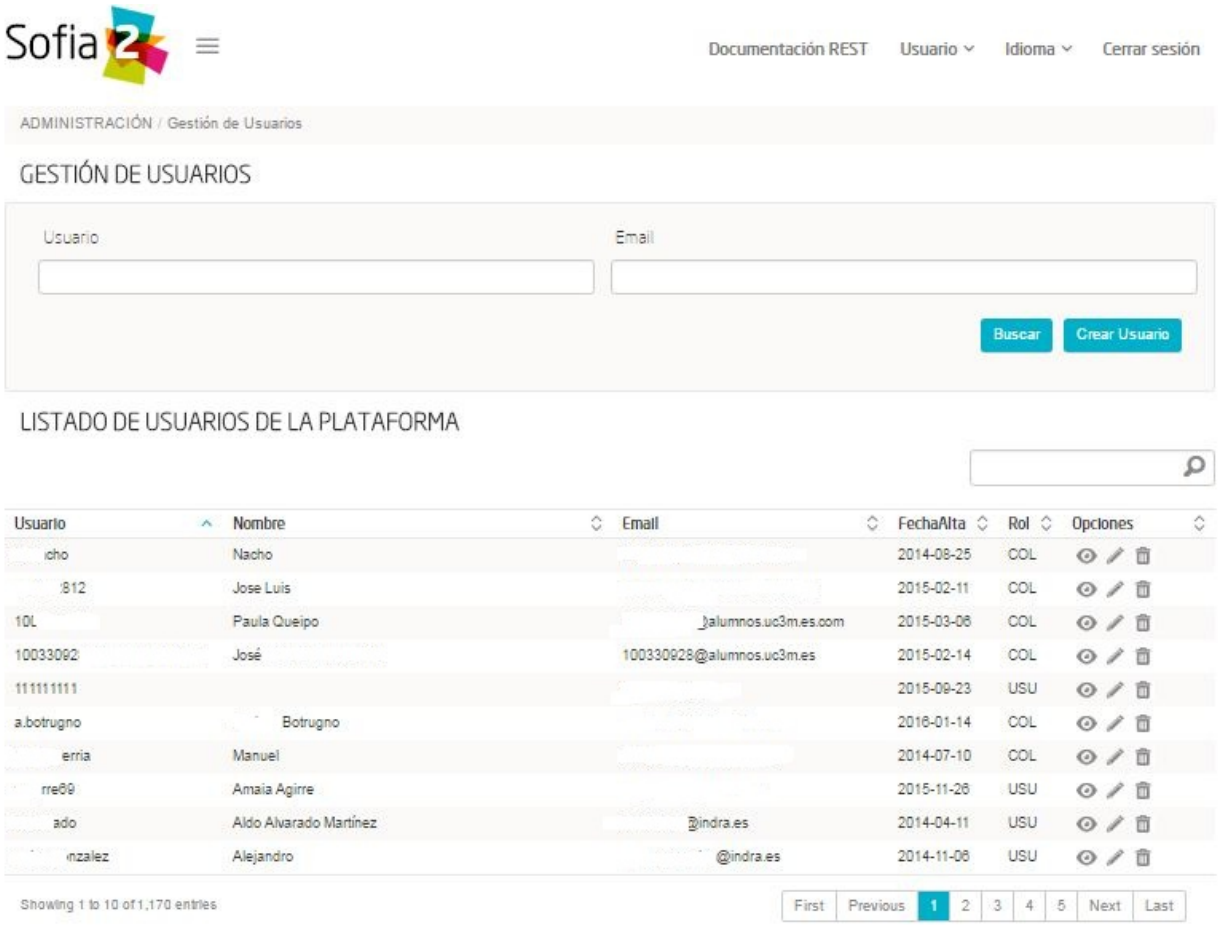
- **Gestión de usuarios y roles**: asignación de roles a usuarios, asignación de permisos sobre información almacenada en la plataforma.

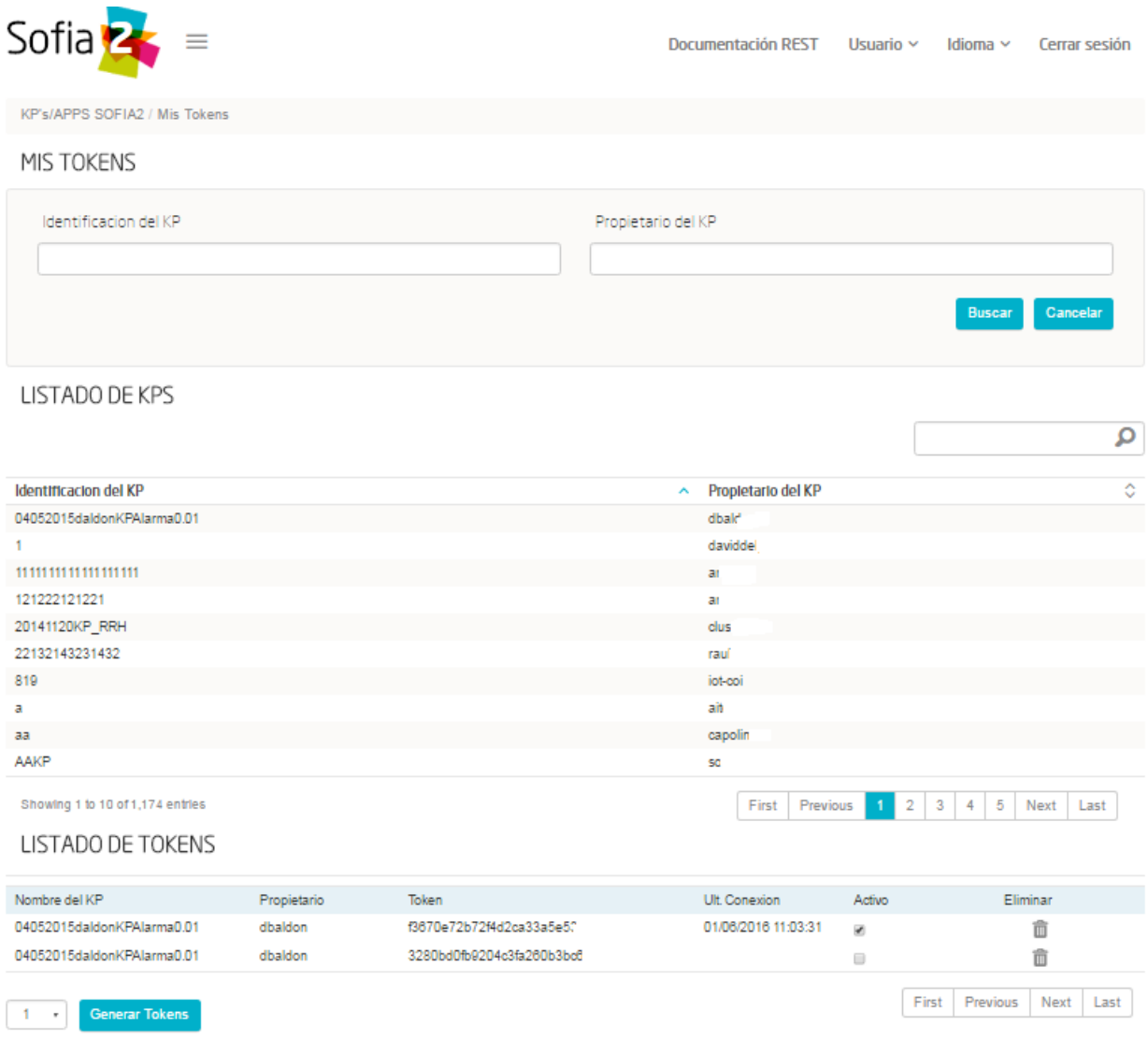
A través de esta capacidad será posible la integración con la gestión de perfiles centralizada usada en el Sistema.

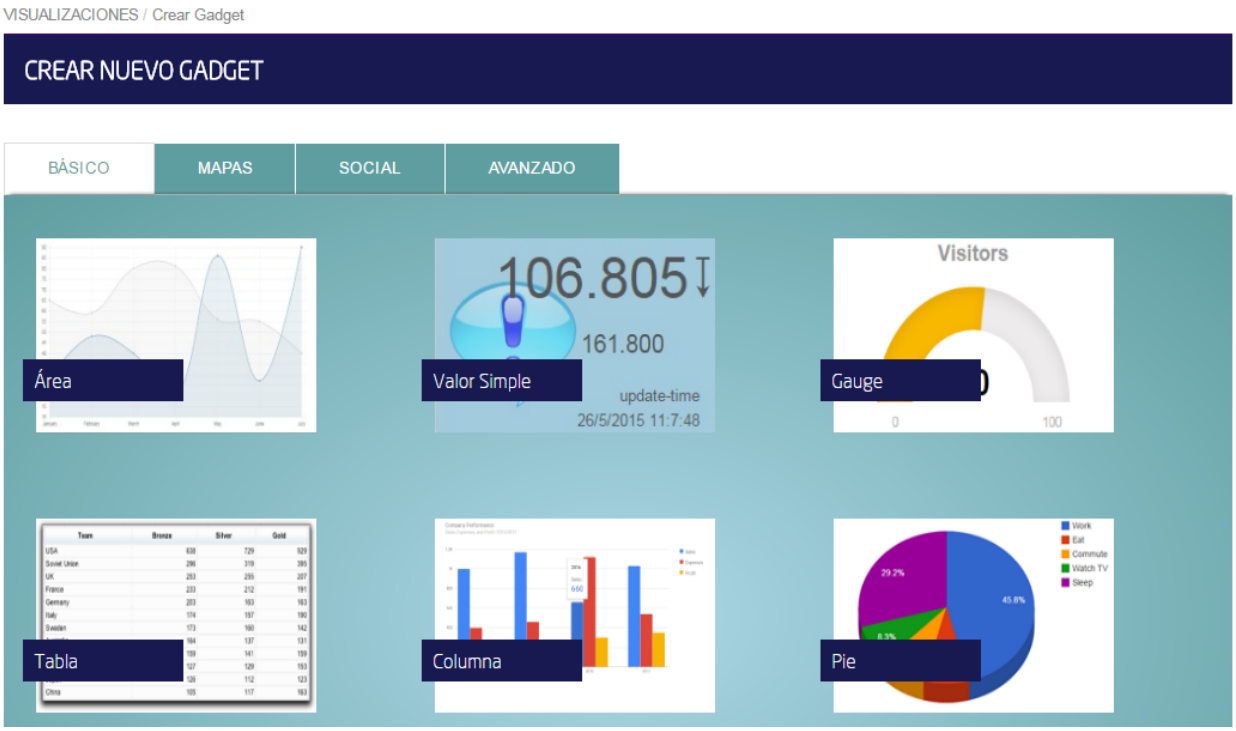
- **Gestión de clientes de la Plataforma y sus tokens** (requeridos para interactuar con la plataforma): permite una gestión integral de aquellos clientes (KPs) que están accediendo o insertando datos en la plataforma, pudiendo invalidar en cualquier momento tokens de acceso.
- **Visualizaciones**: La Plataforma permite acceder a la información gestionada por ella a través de cualquiera de sus APIs (por ejemplo API Javascript para desarrollo de Webs), a través del API Manager vía Interfaces REST y a través de conexión ODBC y JDBC. Además de esto ofrece 3 módulos que resuelven out-of-the-box las necesidades de visualización:
- **Dashboards**:

Este módulo permite crear de forma sencilla y visual Cuadros de Mando sobre la información gestionada por la Plataforma.

Este módulo permite crear cuadros de mando de forma sencilla y visual desde el Panel de control de la Plataforma. Para eso ofrece diversos tipos de gadgets:


Sofia 2
Documentación REST
Usuario
Idioma
Cerrar sesión
ADMINISTRACIÓN / Gestión de Usuarios
GESTIÓN DE USUARIOS
Usuario
Email
Buscar
Crear Usuario
LISTADO DE USUARIOS DE LA PLATAFORMA
Usuario
Nombre
Email
FechaAlta
Rol
Opciones
icho Nacho 2014-08-25 COL
812 Jose Luis 2015-02-11 COL
10L Paula Queipo _alumnos.uc3m.es.com 2015-03-06 COL
10033092 José 100330928@alumnos.uc3m.es 2015-02-14 COL
111111111 2015-09-23 USU
a.botrugno Botrugno 2016-01-14 COL
erria Manuel 2014-07-10 COL
rre69 Amaia Agirre 2015-11-26 USU
ado Aldo Alvarado Martínez @indra.es 2014-04-11 USU
nzalez Alejandro @indra.es 2014-11-06 USU
Showing 1 to 10 of 1,170 entries
First
Previous
1
2
3
4
5
Next
Last


Sofia 2
Documentación REST
Usuario
Idioma
Cerrar sesión
KP's/APPS SOFIA2 / Mis Tokens
MIS TOKENS
Identificacion del KP
Propietario del KP
Buscar
Cancelar
LISTADO DE KPS
Identificacion del KP
Propietario del KP
04052015daldonKPAlarma0.01
1
1111111111111111
121222121221
20141120KP_RRH
22132143231432
819
a
aa
AAKP
Showing 1 to 10 of 1,174 entries
First
Previous
1
2
3
4
5
Next
Last
LISTADO DE TOKENS
Nombre del KP
Propietario
Token
Ult. Conexion
Activo
Eliminar
04052015daldonKPAlarma0.01
dbaldon
01/06/2016 11:03:31
04052015daldonKPAlarma0.01
dbaldon
Generar Tokens
First
Previous
Next
Last



que pueden unirse para generar un completo Dashboard, bien sobre la información que se va añadiendo a la plataforma, bien sobre la información histórica:

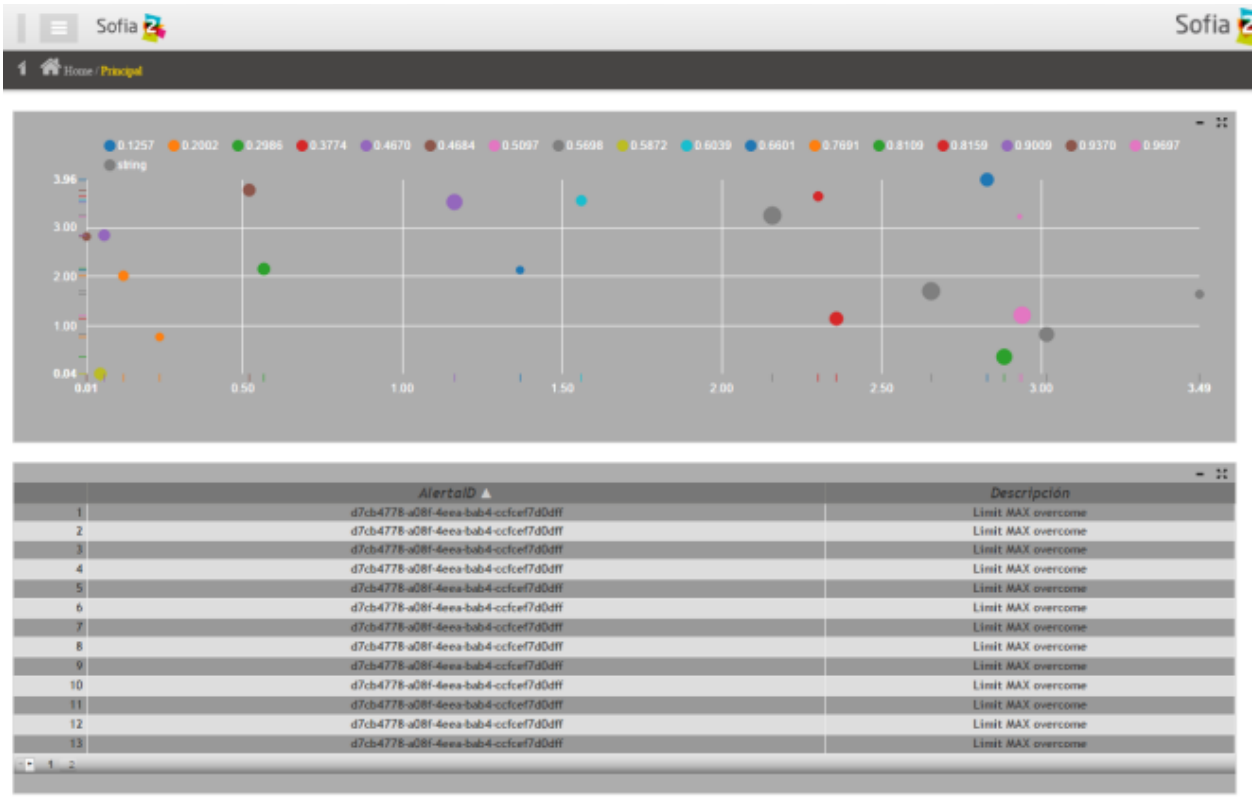
Por lo tanto las capacidades que nos ofrecen los son las siguientes:

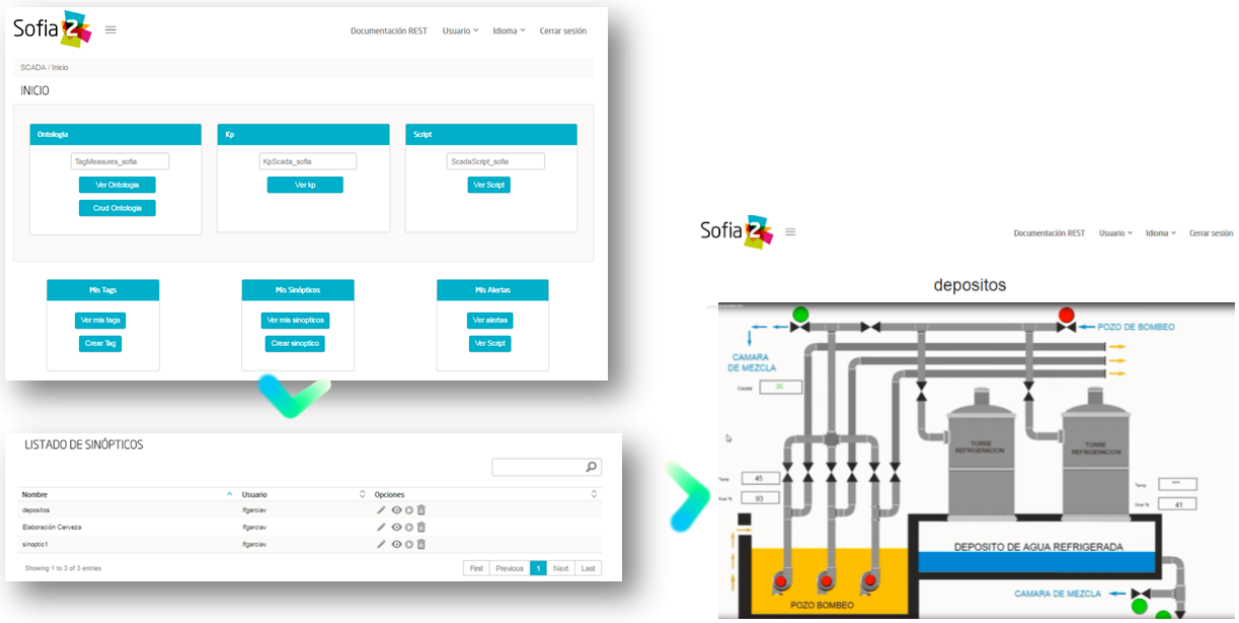
- **Biblioteca** para la configuración de **Gadgets**.
- Composición de **Dashboards** reutilizando Gadgets configurados.
- **Gestión** (Creación/modificación/eliminación) de Gadgets y Dashboards desde la misma interfaz **centralizada** de administración.
- **Integración** con los repositorios de información y con fuentes externas.
- **Exportación** de la información en distintos formatos (xls, csv, html)

- **Sinópticos**:

Este módulo permite desde el Panel de Control crear visualmente sinópticos tipo SCADA que reaccionan a los eventos (ontologías) generados en la Plataforma permitiendo crear completos cuadros de mando operacionales

Por lo tanto permite la **monitorización y actuación** en **tiempo real** sobre **procesos industriales** para los que se proporcionan las herramientas de diseño de Sinópticos, reglas, alertas y tags


Sofia
Sofia
Home / Principal
0.1257 0.2002 0.2986 0.3774 0.4670 0.4684 0.5097 0.5698 0.5872 0.6039 0.6601 0.7691 0.8109 0.8159 0.9009 0.9370 0.9697
string
3.96
3.00
2.00
1.00
0.04
0.01 0.50 1.00 1.50 2.00 2.50 3.00 3.49
AlertaID
Descripción
d7cb4778-a08f-4eea-bab4-ccfcef7d0dff
Limit MAX overcome
1 2
Grouped
Stacked
Producto
9784.00
8000.00
6000.00
4000.00
2000.00
0.00
string
Polaris
MP4
MP3
Google ...
GoPro
Movil
iPod
Camara
WMP3
white
navy
green
red
aqua
black
28.6%
Reuters: Technology News
With HoloLens, Microsoft aims to avoid Google's mistakes
U.S. consumers back investors' rush to self-driving cars : study
Indoors launches timing, location service as GPS back-up
Hackers probe defenses of Middle East banks : FireEye
valor
2016-03-31T14:55:21.162Z
0
update-time
31/3/2016 16:55:21
Mapa
Satélite
Google



- **Informes**:

Este módulo ofrece una funcionalidad de reporting sobre la información gestionada por el Sofia2 Storage (BDTR y la BDH).

Permite:

- Diseñar Informes de forma flexible desde el editor de informes visual integrado en Sofia2-SDK.
- Acceder a BDTR y BDH.
- Cargar informes en la Plataforma a través de la Consola Web de la Plataforma.
- Visualizar Informes en formato HTML integrados en la Consola.
- Generar informes en formatos HTML, PDF, Word, Excel,...
- Guardar informes generados, catalogarlos y darles permisos de visualización.

La implementación de referencia de este módulo se basa en la librería open-source Jasper Reports, que permite diseñar informes de manera flexible y personalizable:

Desde Sofia2 Control Panel y dependiendo del perfil del usuario se habilitarán las opciones a las que se tenga acceso.

Con el perfil de Administrador se pueden crear/modificar/eliminar informes y asignar permisos de consulta a otros usuarios. La opción Autorización Informes no estará disponible para usuarios con rol Usuario.

Sólo el propietario (o usuarios administradores) podrá consultar, editar y eliminar sus informes. Los informes generados se almacenarán y se pueden crear grupos de usuarios de consulta a dichos informes.

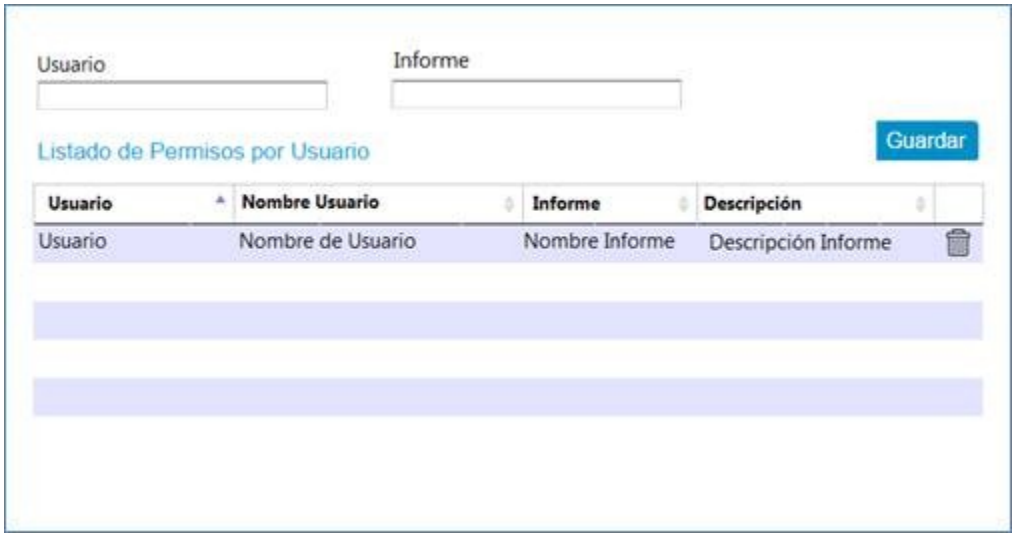
El usuario, desde la interfaz de gestión puede visualizar sus propios informes y sobre los tenga permiso. En caso de que se trate de un informe parametrizado, tras pulsar sobre la generación de uno de los tipos de informes, se solicitará mediante un diálogo los parámetros necesarios para su ejecución. Una vez introducidos, se presentará por pantalla el resultado.

Los informes se podrán visualizar en HTML o PDF y exportar los informes a formato PDF, a una hoja de cálculo de Excel o a un documento Word.

INFORMES

Generar Informes

Autorización Informes



Semantic Broker (SIB)

Módulo de la Plataforma que recibe, procesa y almacena toda la información de las aplicaciones, sensores y dispositivos conectados, actuando como Bus de Interoperabilidad. Esta capa validará la corrección sintáctica y semántica del dato recibido gracias a la definición previa de la estructura del dato esperado (ontología), identificando de qué dato trata, y aplicando la seguridad correspondiente al mismo.

- **Plugins**:

La Plataforma ofrece el concepto de plugin como mecanismos de extensión de la Plataforma que permite incorporar nuevas funcionalidades en esta (por ejemplo un nuevo conector basado en protocolo OneM2M) de forma sencilla.

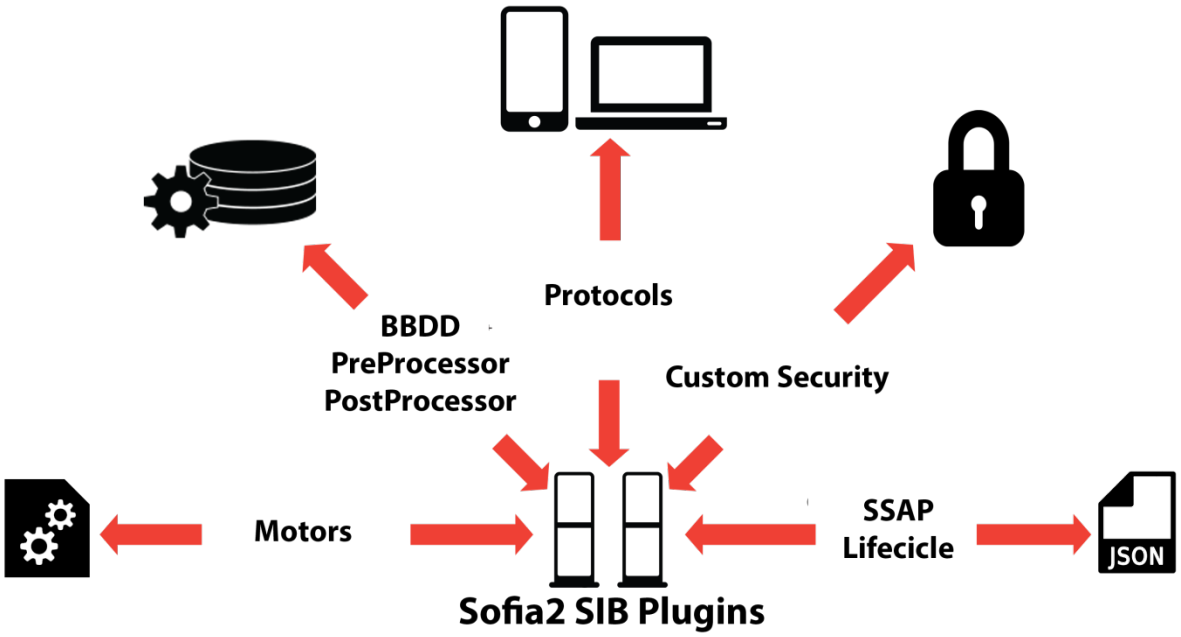
Mediante el despliegue de **plugins** se podrá ampliar o adaptar la funcionalidad por defecto de este componente de una manera sencilla.

El motor de plugins permite crear nuevos conectores, cambiar autenticación, auditar información, persistir en otros repositorios, generar KPIs, monitorizar, etc. , es decir, dotan de una flexibilidad máxima a la Plataforma.

Existen diversos tipos de plugins a desarrollar sobre la Plataforma:

- **Plugin Tipo Gateway**: permiten desarrollar conectores en otros protocolos
- **Plugins de pre y postprocesamiento**: Permiten procesas cualquier acción realizada dentro del SIB antes o después de su persistencia en BD.
- **Plugins de Seguridad**: permiten ampliar o cambiar modelo de autenticación y autorización, por ejemplo configurándolo contra un repositorio LDAP corporativo o una BD externa.
- **Plugins de Ciclo de Vida del SIB**: Permiten realizar acciones al parar o arrancar la Plataforma.
- **Plugins de Procesado de Mensajes**: permiten preprocesar y postprocesar cualquier mensaje incluyendo invocación a otros Backends.
- **Plugins de Motores**: permiten interceptar las acciones que realizan los diversos motores.

La plataforma incluye un conjunto de plugins preconstruidos y los mecanismos (APIs y guías) para construir nuevos plugins.



IoT Gateway

Capa de abstracción del protocolo de comunicación, que implementa el protocolo SSAP (Smart Space Access Protocol).

Este módulo se especializa en el esquema de comunicación con dispositivos, sensores y sistemas TI en un contexto IoT, donde se debe facilitar el uso de protocolos de comunicación ligeros en un entorno tecnológico heterogéneo. El protocolo SSAP proporciona tanto la ligereza del mensaje como su homogeneización a nivel de aplicación. De esta manera, la información gestionada por las subsiguientes capas de la plataforma es completamente agnóstica del protocolo tecnológico usado para el envío del dato, dando lugar a su gestión desde un punto de vista semántico.

Para agilizar la integración con la plataforma (desde sensores, dispositivos o sistemas TI), este componente ofrece la interpretación de multitud de protocolos "out of the box":

- REST y WebSockets: para clientes Javascript, smartphones,..
- MQTT para comunicaciones bidireccionales y dispositivos básicos.
- Web Services/JMS/AMQP para aplicaciones empresariales.

Además, se facilita la incorporación de nuevos protocolos gracias al despliegue de nuevos Plugins.

Esto, sumado a las **APIS multilenguaje** (Descargar) que ofrece la plataforma facilita el desarrollo de cualquier cliente que quiera comunicarse con la plataforma, permitiendo la abstracción de los detalles técnicos del protocolo a utilizar (ya considerados en el API).

Process

Se compone de los dos módulos siguientes:

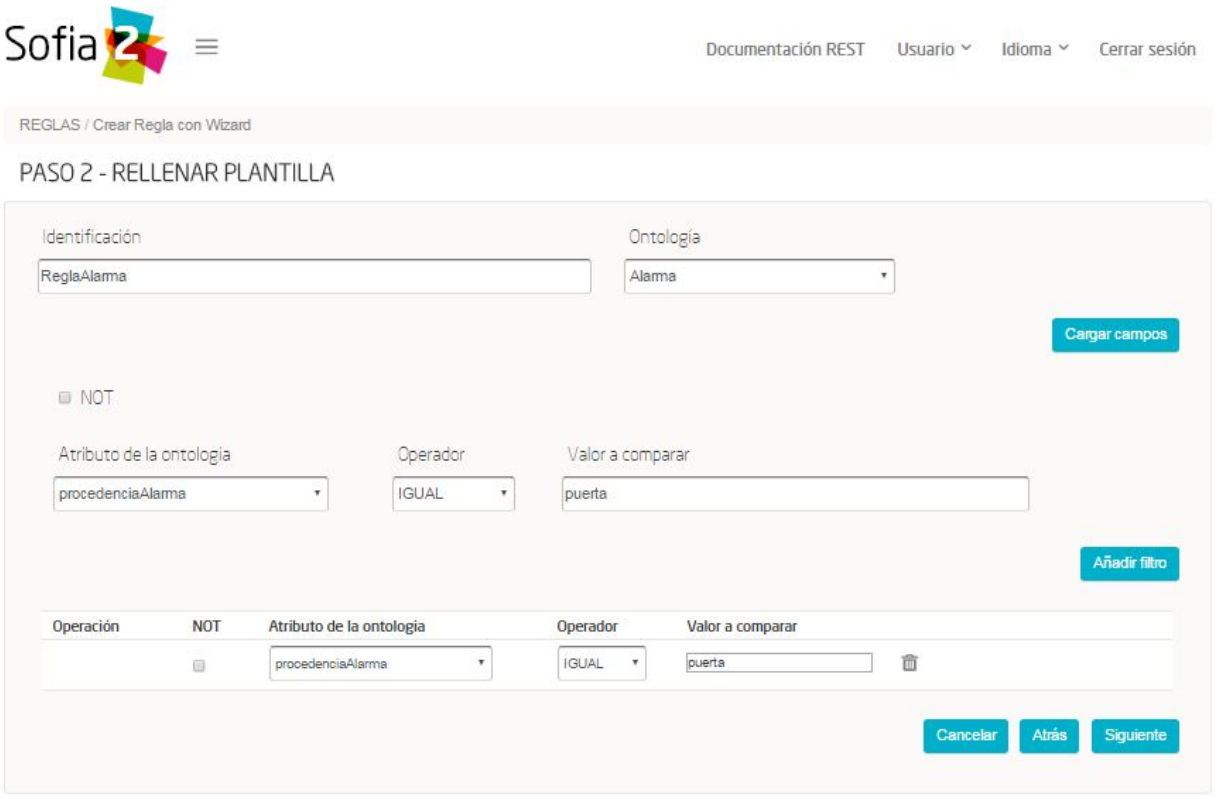
Motor de Reglas (Sofia2-Rules)

El módulo Sofia2-Rules permite ampliar el funcionamiento de la Plataforma permitiendo definir reglas que se ejecutan ante ciertas condiciones (inserción de un nuevo dato o cada cierto tiempo).

Estas reglas dan la capacidad de definir, en base a Scripting, acciones que ejecuta la Plataforma. Gracias a ellas se pueden gestionar y tratar los datos de la plataforma.

Por ejemplo se pueden definir transformaciones a los datos existentes o implementar el motor de reglas en tiempo real sobre patrones/datos/eventos.

Todas estas reglas se crean desde la Consola Web de forma sencilla y sin programar. La creación de reglas desde la consola va en función del rol, por ejemplo usuario administrador puede crear reglas sobre cualquier ontología mientras que un usuario colaborador puede definir reglas sobre ontologías de las que es propietario.



La Plataforma disponibiliza un conjunto de acciones predefinidas que se pueden ejecutar dentro de la regla, como Enviar Mail, Generar Alarma,... Además, se pueden crear nuevas acciones predefinidas en tiempo de desarrollo.

Podemos diferenciar dos tipos de Reglas Scripting:

- **Reglas Temporizadas**. Ejecutan el Script definido cada X segundos definidos en la creación de la regla, este tipo de Reglas únicamente define el THEN pues no existe un flujo alternativo a una condición.
- **Reglas Ontología**. Son reglas asociadas a una Ontología y por lo tanto únicamente entran en acción cuando se realiza la inserción de una Instancia de Ontología en el SIB. Definen una condición sobre un valor de la instancia de la Ontología, con los operadores (=,>,<, ¡=), está escrito en lenguaje Groovy e interpretado como un Script.

Las reglas pueden ser desarrolladas en Groovy (lenguaje sencillo y de alta productividad basado en Java), Python, R o directamente en Java.

Los scripts pueden ser desplegados en caliente en la Plataforma, de forma que pueden actuar en tiempo real directamente tras su creación.

La ejecución de cada uno de los Script se realiza de forma independiente de la JVM que gestiona el SIB aislando al SIB y los diferentes Script de posibles errores.

Las Reglas se desarrollan haciendo uso de una biblioteca de APIS que permiten interactuar con los diferentes elementos de la plataforma (p.e. ontologías, BDTR) y con elementos externos a ella (p.e. envío de correos, redes sociales). Estas APIS son extensibles por el administrador de forma sencilla, estando disponibles out-of-the-box librerías para gestión de conexiones HTTP, JMS, LOGS, mail, BDTR, ejecución de Scripts, SSAP, Twitter, entre otras.

Motor CEP (Sofia2-CEP)

El motor CEP que permite definir reglas en las que interviene el tiempo (por ejemplo que no ha llegado una cierta medida en 1 día). A los eventos generados por el motor CEP pueden suscribirse los clientes o servir como entrada al motor de Reglas.

Sofia2 Storage

Modulo de almacenamiento de la información de la plataforma.

Con el objetivo de garantizar que, para cada momento del **ciclo de vida de la información**, ésta se gestiona de la menor manera, la plataforma plantea el uso de tres repositorios distintos que se complementan y comunican componiendo una solución de almacenamiento completa:

Este módulo nos ofrece las siguientes **Capacidades**:

- Un repositorio adecuado para cada momento en el **ciclo de vida de la información**.
- Optimización de tiempos de acceso a la información.

- **Soporte a diferentes tecnologías** en función del patrón de accesos, altas y consultas de cada repositorio.
- **Escalabilidad horizontal** de todos los repositorios.
- Los repositiorios están integrados entre sí y con las demás capas de la plataforma.
- Soportan estándares y bases de datos **SQL y NO-SQL**.

Base de Datos Tiempo Real (BDTR)

Almacena la información recibida en tiempo real, como instancias de ontologías, siendo, por lo tanto el primer repositorio en el que se almacena la información recibida de:

- sensores y dispositivos integrados con la plataforma en un contexto IoT típico.
- Redes Sociales, en los casos en que la escucha de twitter, Facebook, Instagram... es un dato más en el universo de los datos configurados.
- Cualquier otra fuente cuya información sea requerida y/o reportada frecuentemente.

Esta información se valida automáticamente, garantizando corrección de la estructura según la definición previa de las ontologías (entidades o conceptos de negocio).

Por cada ontología se puede configurar una ventana de tiempo a partir de la cual la información ya no se considera 'información en tiempo real', de tal manera que será migrada automáticamente al repositorio de información histórica.

En función del patrón de accesos a la información de tiempo real, se puede implementar este módulo con tecnologías diferentes, que aseguren el acceso ágil a la misma (MongoDB, Hbase, BD relacionales...).

Gracias a Kudu e Impala se facilita el acceso en tiempo Real para la analítica de datos.

Podemos destacar las siguientes **capacidades** de este repositorio:

- **Acceso ágil** a la información.
- **Herramienta de consulta SQL** integrada en el panel de control Sofia2 incluso si la base de datos es NO-SQL.
- **Origen de datos para Analítica** de Datos en Tiempo Real.
- **Integración** con el motor de Reglas, Machine Learning y capas de integración.
- **Escalabilidad** horizontal.
- **Control sintáctico** de la información insertada de acuerdo a las ontologías definidas.

Base de Datos Histórica (BDH).

Almacena la información histórica para su posterior explotación analítica.

La información almacenada estará disponible como origen de datos para los distintos módulos de la plataforma: Integración, Machine Learning, API Manager...

La implementación de este repositorio está basada en Hadoop como repositorio, Hive como solución Datawarehouse y SparkSQL para facilitar el acceso homogéneo a los datos.

Como **características** más importantes de este repositorio podemos destacar las siguientes:

- **Almacenamiento temporal** de información heterogénea.
- **Herramienta de consulta SQL** integrada en el panel de control Sofia2.

- **Origen de datos para Analítica** de Datos Históricos
- **Integración** con el motor de Reglas, Machine Learning y capas de integración.
- **Escalabilidad** horizontal.
- Actúa como el corazón del **Data Lake** de la plataforma, almacenando información heterogénea con capacidad de procesamiento

Repositorio Staging

Almacena información en diferentes estados (estructurada, semi-estructurada y no estructurada) **temporalmente**, para facilitar procesos complejos de transformación, ingestión y exposición de datos que requieran la persistencia temporal de estados intermedios del proceso.

Este repositorio se implementa sobre **HDFS**, cuya arquitectura basada en ficheros de texto y nodos de procesamiento paralelo, proporcionan la flexibilidad y escalado horizontal necesarios.

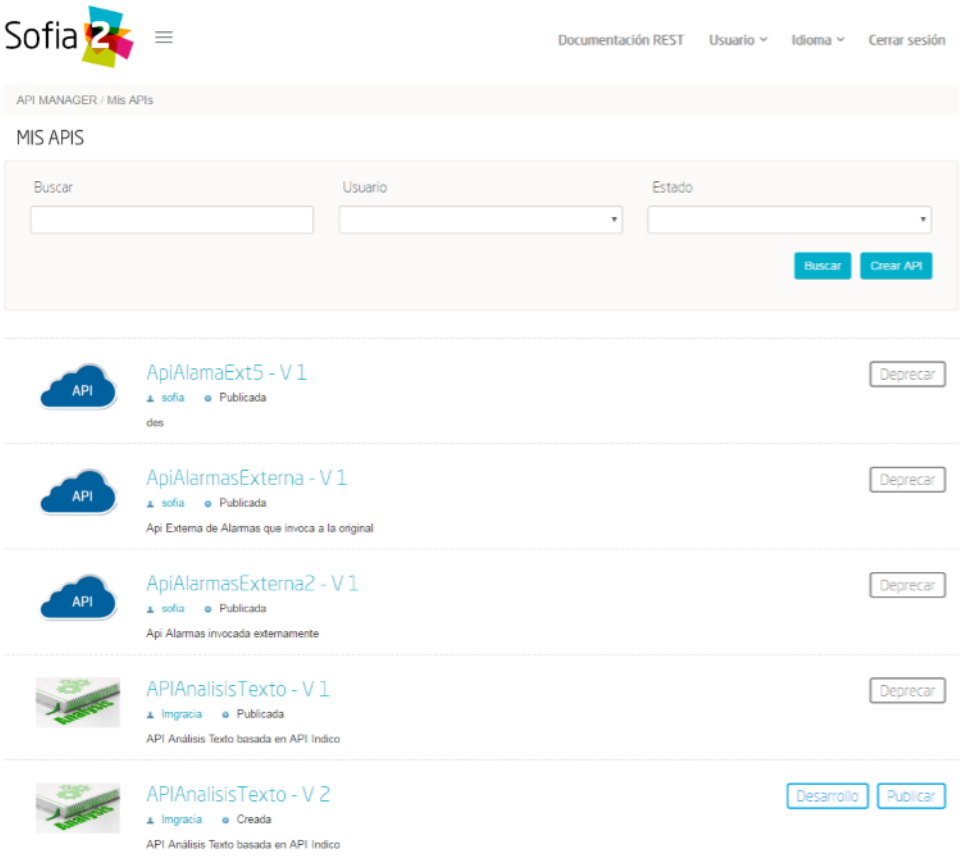
Podemos destacar las siguientes capacidades de este repositorio:

- **Almacenamiento temporal** de información heterogénea.
- **Usado para dar soporte a procesos analíticos** y de transformación de dato complejos.
- **Integración** con el motor de Reglas y Machine Learning.
- **Escalabilidad** horizontal.

API Manager

Este módulo permite acceder a la información recolectada y gestionada por la Plataforma.

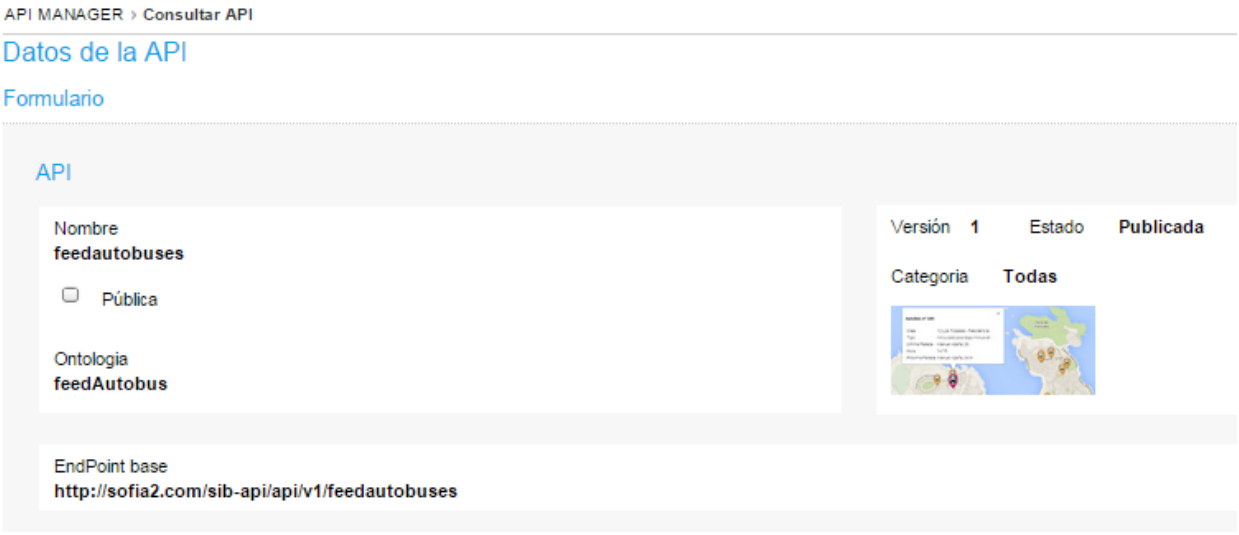
Para ello, publica la información gestionada por la plataforma como APIs REST. Esto permite poner toda información a disposición y uso directo de los distintos stakeholders involucrados en el desarrollo de la actividad diaria sin necesidad de conocer las APIs y protocolos de la Plataforma.


Sofia 2
Documentación REST
Usuario
Idioma
Cerrar sesión
API MANAGER / Mis APIs
MIS APIS
Buscar
Usuario
Estado
Buscar
Crear API
API
ApiAlamaExt5 - V 1
sofia
Publicada
des
Deprecar
API
ApiAlarmasExterna - V 1
sofia
Publicada
Api Externa de Alarmas que invoca a la original
Deprecar
API
ApiAlarmasExterna2 - V 1
sofia
Publicada
Api Alarmas invocada externamente
Deprecar
APIAnalisisTexto - V 1
lmgracia
Publicada
API Análisis Texto basada en API Indico
Deprecar
APIAnalisisTexto - V 2
lmgracia
Creada
API Análisis Texto basada en API Indico
Desarrollo
Publicar

Este módulo también permite disponibilizar Servicios REST externos a la Plataforma, lo que permite ofrecer un punto único de acceso a APIS internas y externas de la Plataforma.

Sus principales **capacidades** son:

- **Exposición de entidades (ontologías) como APIS REST**. Desde la consola de administración es posible exponer como API REST cualquier entidad (ontología) , indicando los métodos a exponer para su consulta y tratamiento.



- **Simplicidad en el acceso a la información de la plataforma** a través de APIs REST lo que permite que esta se pueda consumir sin conocer los detalles técnicos de la Plataforma.
- **Portal integrado en el Panel de Control** que permite la búsqueda, suscripción e invocación de las APIs.
- **Autenticación** mediante token en cabecera de cada petición HTTP. Desde la consola de administración, cada usuario, o en su caso un administrador, puede generar y regenerar sus token de autenticación. En cada petición se deberá incluir en la cabecera HTTP este token para autenticar la petición.
- **Seguridad en el acceso personalizado a las APIs**, a nivel de API y a nivel de entidad de información. Desde la consola de administración, cada usuario que exponga una entidad de información puede otorgar y revocar el permiso de operación sobre el API a otros usuarios.
- **Control de cuotas de consumo** en el acceso a la información para usuarios. Sofia2-API Manager gestiona el consumo realizado por cada usuario en función de distintas políticas configurables desde la consola de administración.
- **Proporcionar acceso a APIS externas** incluyendo sistemas de datos abiertos.
- **Ciclo de vida de las APIS expuestas**, gestionando a través de la consola de administración la fuente de los datos, versión del API, categoría y estado de exposición del API (Creada, en Desarrollo, Publicada, Deprecada, Eliminada).
- **Documentación web de APIS** expuestas mediante página descriptiva de los comentarios incluidos durante la creación del APIS y la definición de los métodos expuestos.
- **Cache de APIs configurable**, cacheando la respuesta de las peticiones durante un intervalo configurable el tiempo de respuesta para peticiones complejas sobre grandes volúmenes de datos es casi inmediato.

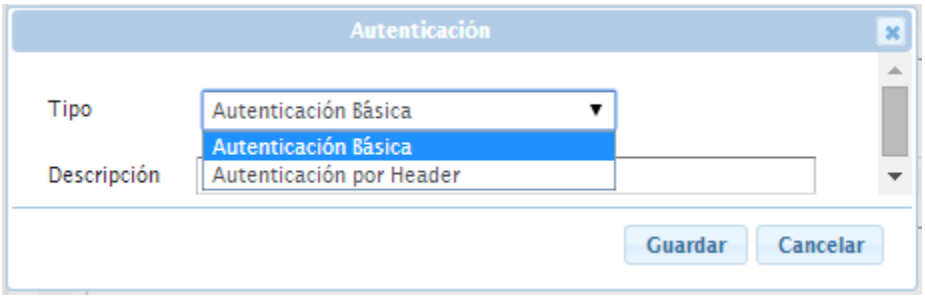
Operaciones

GET	/{id}	**Descripción** Obtiene el estado de un autobus de A Coruña a través de su identificador
GET(sql)	?$filter={query}&$targetdb={targetdb}	**Descripción** Obtiene el estado de un autobus de A Coruña a través de una consulta sql
POST	/	**Descripción** Permite insertar información relativa a los autobuses de A Coruña
PUT	/	**Descripción** Permite actualizar información relativa a los autobuses de A Coruña
DELETE	/	**Descripción** Permite eliminar información relativa a los autobuses de A Coruña
DELETE	/{id}	**Descripción** Permite eliminar información relativa a los autobuses de A Coruña a través de su identificador
GET	?$query={query}&$queryType={queryType}	**Descripción** Permite realizar operaciones SQLLIKE para insertar, eliminar y actualizar

SuscripciónAPI.jpg


API MANAGER › Mi API Key
Mi API Key
Token de Usuario
61a01008399749c98e664c693cbf819d
Regenerar


X-SOFIA2-APIKey
X-SOFIA2-APIKey


Autenticación
Tipo
Autenticación Básica
Autenticación Básica
Descripción
Autenticación por Header
Guardar
Cancelar


API INFO
Base path: http://sofia2.com/sib-api/api/v1/feedautobuses
Version: 1 - 2014-06-27
APIS
feedautobuses
OBJECTS
FEEDAUTOBUSES
Información relativa a los autobuses de A Coruña
/ PUT
/ POST
/{id} GET
?$query={query}&$queryType={queryType} GET
?$filter={query}&$targetdb={targetdb} GET
/{id} DELETE
/ DELETE


Cachear Resultados
Tiempo de Caché (minutos)
2

Holystic Viewer

Este módulo forma parte del ecosistema de la Plataforma, es desarrollado por una empresa partner de Indra y puede adquirirse o no junto a la plataforma.

Sofia2-HolisticViewer es el módulo de visualización avanzada de la Plataforma, se trata de un sistema integral de visualización avanzada e interactiva que permite una gestión de información geolocalizada asociándola a un entorno de visualización tridimensional y multimedia:

Proporciona geovisualización en tiempo real sobre el terreno

Módulos Big Data:

Sofia2 DataFlow

Permite hacer ingesta masiva de datos desde multitud de fuentes, transformaciones simples online sobre la información y ruteado hacia otro destino (módulo IoT Flow, BDTR, BDH,...). Es posible añadir plugins a la plataforma para incorporar nuevas fuentes, transformaciones y destinos.

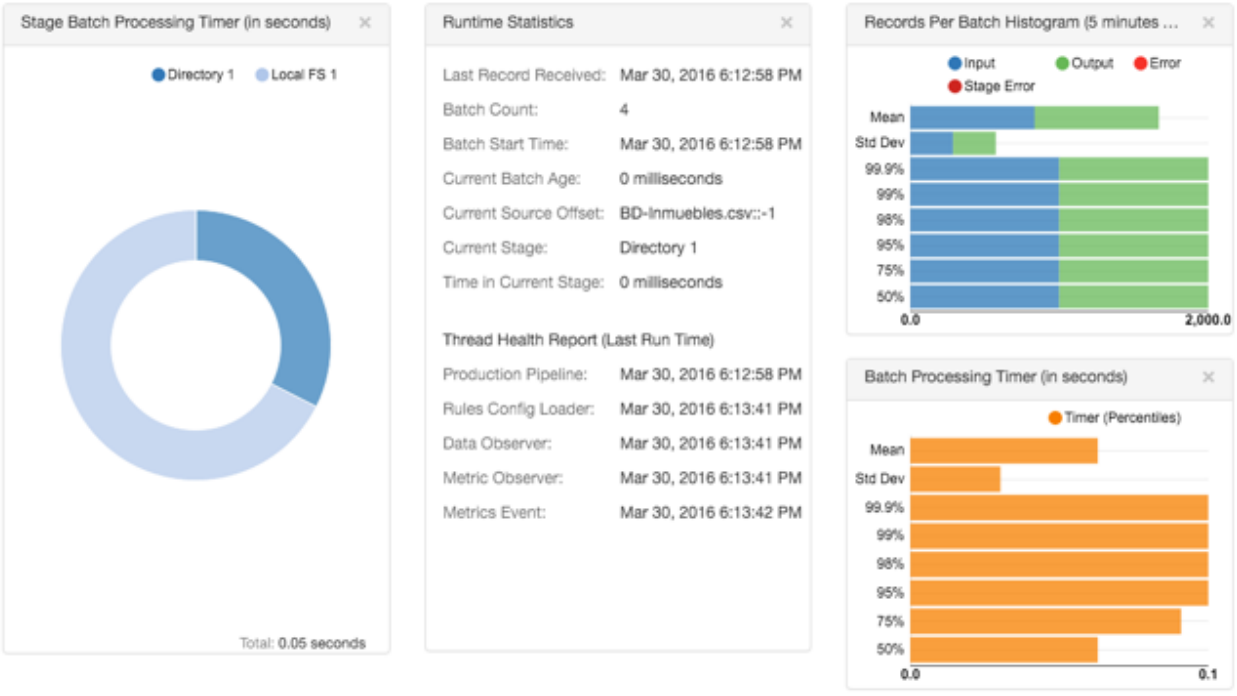
La composición del proceso ETL (Extracción, Transformación y Carga o Load), se realiza mediante el drag&drop de las tareas disponibles en la barra de herramientas.

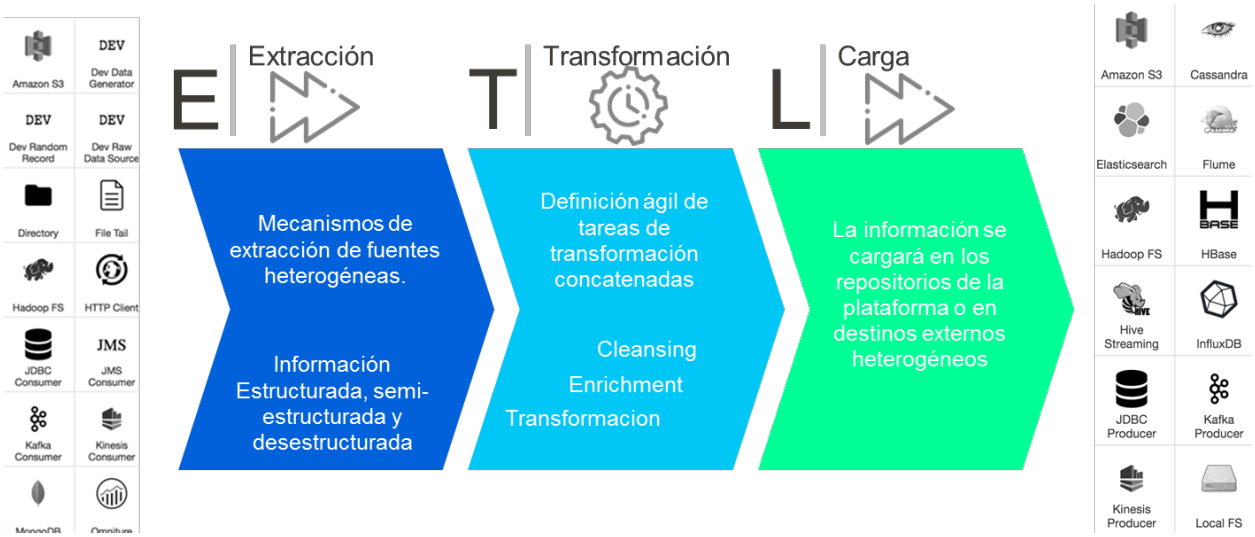
Monitorización: En tiempo de ejecución, se pueden configurar reglas para capturar y visualizar datos de un pipeline en ejecución. Además es posible consultar las estadísticas de ejecución de cualquier pipeline en tiempo real, los datos procesados y el historial del pipeline.

Alertas: La configuración de alertas y thresholds de normalidad posibilitan la ejecución de acciones automáticas como la comunicación de estos eventos y la visualización del detalle.

Haciendo foco en las **capacidades** ETL del módulo, podemos destacar las siguientes capacidades por cada fase del proceso:

- **Extracción**: Disponen de 18 los orígenes de datos integrados , entre los que se encuentran como orígenes disponibles: Sofia2 (que permite seleccionar la ontología, campos, query...), Excel, AmazonS3, HadoopFS, Kafka...


Stage Batch Processing Timer (in seconds)
Directory 1
Local FS 1
Total: 0.05 seconds
Runtime Statistics
Last Record Received: Mar 30, 2016 6:12:58 PM
Batch Count: 4
Batch Start Time: Mar 30, 2016 6:12:58 PM
Current Batch Age: 0 milliseconds
Current Source Offset: BD-Inmuebles.csv::-1
Current Stage: Directory 1
Time in Current Stage: 0 milliseconds
Thread Health Report (Last Run Time)
Production Pipeline: Mar 30, 2016 6:12:58 PM
Rules Config Loader: Mar 30, 2016 6:13:41 PM
Data Observer: Mar 30, 2016 6:13:41 PM
Metric Observer: Mar 30, 2016 6:13:41 PM
Metrics Event: Mar 30, 2016 6:13:42 PM
Records Per Batch Histogram (5 minutes ...
Input
Output
Error
Stage Error
Mean
Std Dev
99.9%
99%
98%
95%
75%
50%
0.0
2,000.0
Batch Processing Timer (in seconds)
Timer (Percentiles)
Mean
Std Dev
99.9%
99%
98%
95%
75%
50%
0.0
0.1


Amazon S3
Dev Data Generator
Dev Random Record
Dev Raw Data Source
Directory
File Tail
Hadoop FS
HTTP Client
JDBC Consumer
JMS Consumer
Kafka Consumer
Kinesis Consumer
E
Extracción
T
Transformación
L
Carga
Mecanismos de extracción de fuentes heterogéneas.
Información Estructurada, semi-estructurada y desestructurada
Definición ágil de tareas de transformación concatenadas
Cleansing
Enrichment
Transformacion
La información se cargará en los repositorios de la plataforma o en destinos externos heterogéneos
Amazon S3
Cassandra
Elasticsearch
Flume
Hadoop FS
HBase
Hive Streaming
InfluxDB
JDBC Producer
Kafka Producer
Kinesis Producer
Local FS

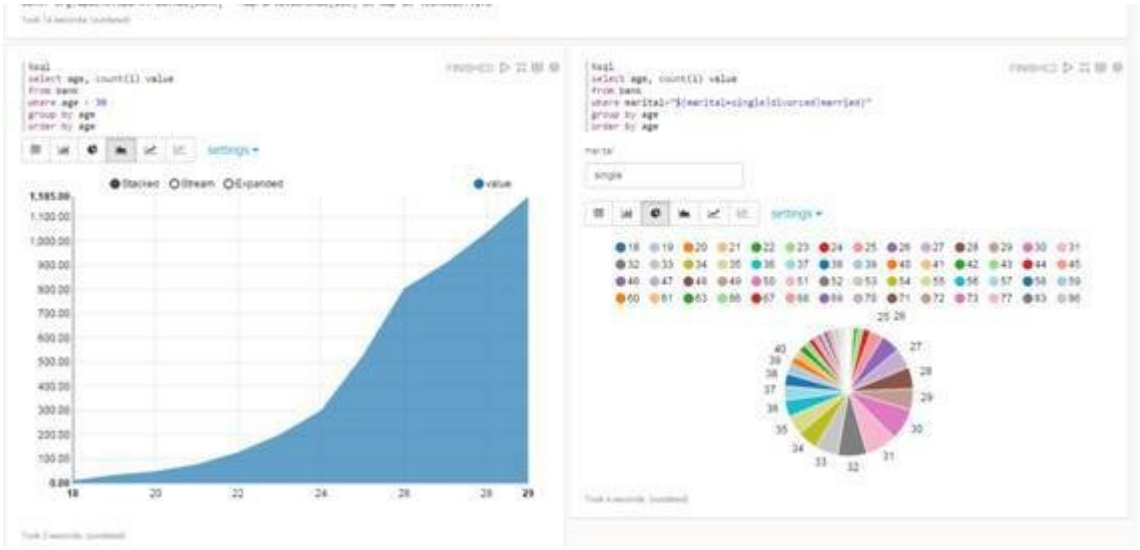
- **Transformación**: Se podrán concatenar sucesivas transformaciones y actuaciones sobre los datos hasta conseguir el proceso completo. Para ello se cuenta con 20 posibles tareas:
- **Evaluador de Expresiones**: Realiza comprobaciones y calculos que puede escribir campos nuevos o existentes.
- **Acciones sobre campos**: Diferentes acciones disponibles sobre los campos como: Converter, Merger, Masker, Hasher, Remover, Renamer. . . .
- **Parseadores de JSON, XML y logs**: Parsea información válida según los diferentes tipos de formato de logs, y esquemas XML y JSON.
- **Selector de Flujo**: Para seleccionar la siguiente actividad a ejecutar sobre el conjunto de datos, en función de condiciones de ejecución.

- **Carga**: Se disponen de más de veinte posibles destinos, a incorporar en el proceso mediante Drag&drop desde la barra de tareas. De ellos podemos destacar el componente Sofia2 (que permite seleccionar la ontología, campos y otros parámetros adicionales), AmazonS3, Cassandra, Hadoop, Kafka, Flume. . . .

Este módulo incluye herramientas de monitorización tanto para agilizar el desarrollo como para hacer seguimiento de la ejecución del proceso una vez activado y publicado.

Sofia2 Notebooks

Permite realizar de manera muy sencilla e interactiva, analítica sobre datos de fuentes muy variadas, incluidas las fuentes de datos de Sofia2. De esta manera se podría, por ejemplo, realizar cargas de archivos desde HDFS a spark, cargar de datos en tablas Hive, lanzar consultas o realizar un proceso complejo de machine learning mediante las librerías de MLlib de Spark. También el posible la utilización de código R así como las numerosas librerías del lenguaje, permitiendo por ejemplos visualizar mapas de leaflet.



Sofia2 Notebooks posee la capacidad de combinar código Scala, Spark, SparkSQL, Hive, R, Shell, o muchos otros con contenido html o directivas reactivas de angular, permitiendo interacciones en tiempo real con una potente interfaz y todo ello en un entorno compartido y multiusuario. Cada lenguaje soportado es gestionado por un intérprete, por lo que siempre que se quiera escribir código de un cierto lenguaje se tendrá que escribir un marcador propio en el párrafo. Además permite realizar **visualizaciones instantáneas de los datos**, pudiendo configurar de forma sencilla los gráficos y cambiar rápidamente el tipo de visualización de los mismos. También es posible la creación de gráficos avanzados gracias a librerías propias de cada lenguaje.

SparkSQL:

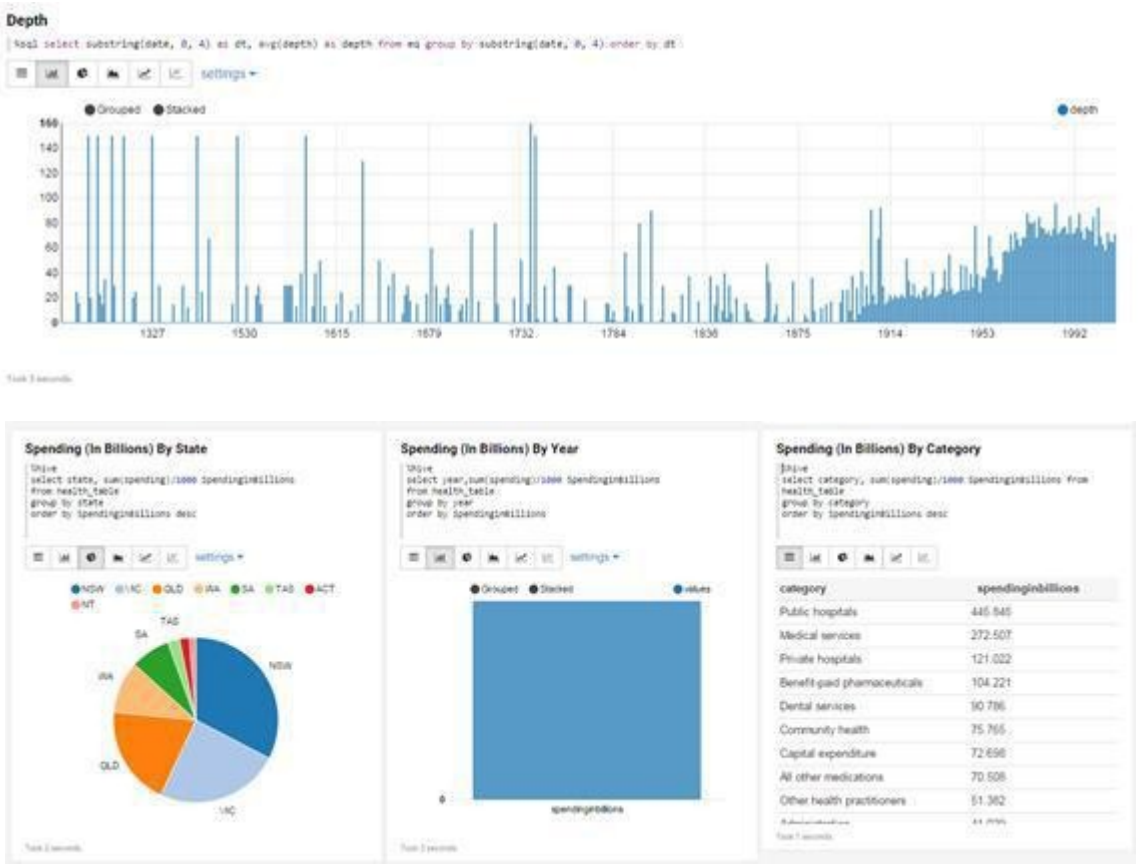
HIVE:

Python:

R:

Cada Notebook se compone de párrafos, que pueden tener diferentes lenguajes, pudiendo ejecutar individualmente los párrafos y visualizando la salida de los mismos, así como el estado de la ejecución.

Tanto los párrafos, como el notebook completo se pueden externalizar vía url, viendo en tiempo real en todos los casos, las ejecuciones de los notebooks o del párrafo en concreto.



Otra característica importante es la posibilidad de **planificar la ejecución de los notebooks** mediante un expresión CRON, pudiendo ejecutar notebook repetidamente y sin pérdida de contexto, pudiendo seleccionar un intervalo de ejecución de los prediseñados o escribir uno personalizado.

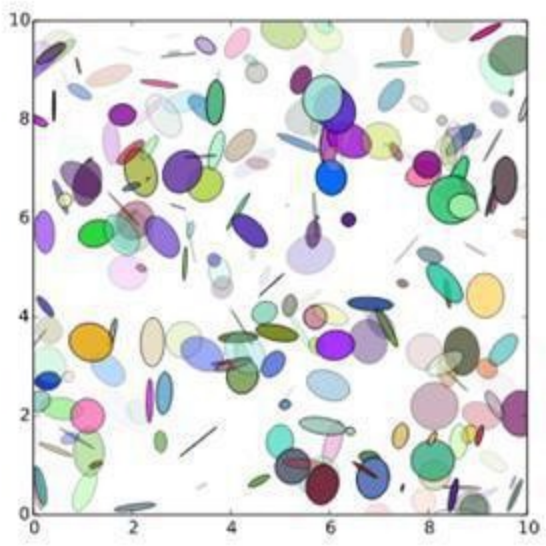
Con todas estas características tenemos una **herramienta web colaborativa**, que es capaz de realizar análisis complejos la información gestionada por la plataforma IoT (tanto en tiempo real como histórica), **combinando diferentes lenguajes y generando vistas gráficas** (u otras acciones), que se pueden planificar para su ejecución periódica, refrescando automáticamente el resultado de la analítica que queda expuesto en una URL.

Sofia2 ML

El modulo Machine Learning de la Plataforma permite aplicar y modelar de forma sencilla diversas técnicas de aprendizaje, entre las cuales podemos destacar las siguientes:

- **Regression**: Técnicas para estimar relaciones entre variables y determinar la importancia relativa de éstas en la predicción de nuevos valores.
- **Clustering**: Técnicas para segmentar los datos en grupos similares.
- **Classification**: Técnicas para identificar la pertenencia de un elemento a un grupo determinado.
- **Recommendation / Prediction**: Técnicas para predecir el valor o preferencia de una entidad nueva basado en históricos de preferencias o comportamientos.

```
NUM = 250
plt.clf()
ells = [Ellipse(xy=rand(2)*10, width=rand(), height=rand(), angle=rand()*360)
        for i in range(NUM)]

fig = figure()
ax = fig.add_subplot(111, aspect='equal')
for e in ells:
    ax.add_artist(e)
    e.set_clip_box(ax.bbox)
    e.set_alpha(rand())
    e.set_facecolor(rand(3))

ax.set_xlim(0, 10)
ax.set_ylim(0, 10)
show(plt)
```



```
%r

require(rCharts)

map3 <- Leaflet$new()
map3$setView(c(51.505, -0.09), zoom = 13)
map3$marker(c(51.5, -0.09), bindPopup = "<p> Hi. I am a popup </p>")
map3$marker(c(51.495, -0.083), bindPopup = "<p> Hi. I am another popup </p>")
map3$print("map3", include_assets=TRUE, cdn=TRUE)
```


```
sc.version
```

FINISHED

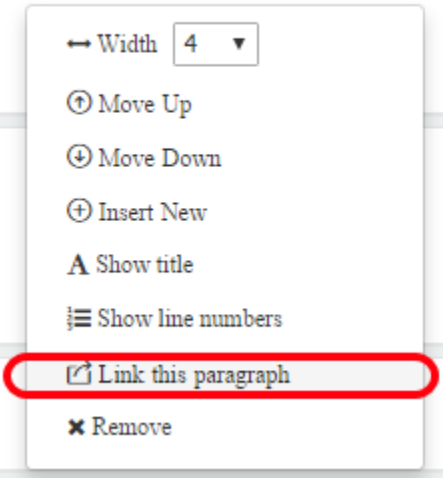
```
res0: String = 1.5.0
```

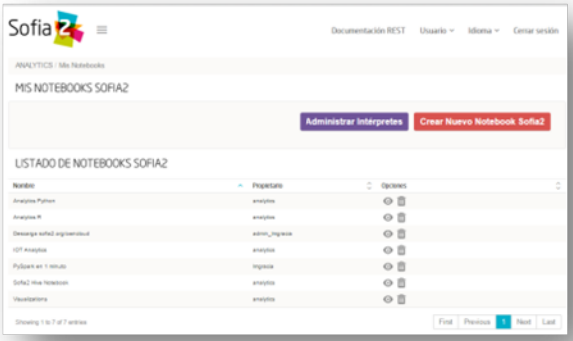
Took 30 seconds

```
%r
1+1
```

```
[1] 2
```

Took 3 seconds





A través del intérprete Sofia2 permite:

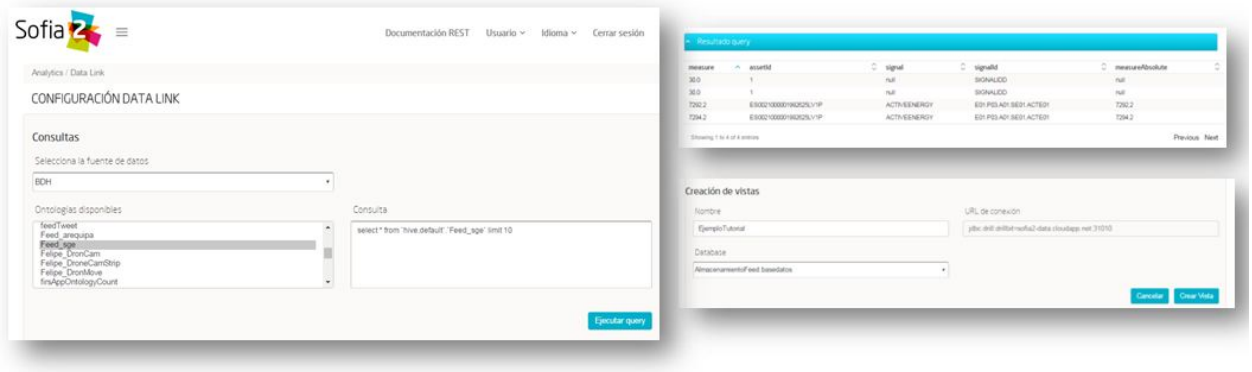
- Almacenar los modelos creados en la plataforma. A partir de esto será posible gestionarlos desde la consola web, desde la que también podremos invocarlos en base a parámetros y darles permisos.
- **Publicar** Scripts Sofia2Models que disponen de métodos para recuperar el **modelo**, guardarlo, invocarlo, evaluar su calidad..
- Generar APIs REST que permitan evaluar sets de datos de entrada a través de los modelos generados. Esto facilita su invocación a través de mecanismos estándar que cuentan además con la seguridad integrada de la plataforma.
- Permite **definir flujos de trabajo visualmente**, de modo que únicamente sea necesario introducir los parámetros de configuración y datos de entrada para definir procesos analíticos.
- **Carga** de ficheros locales.
- **Parseo** de datos en diversos formatos (ARFF, XLS, XLSX, CSV, SVMLight).
- **Algoritmos**: K-means, Generalized Linear Model, Distributed RF, Naïve Bayes, Principal Component Analysis, Gradient Boosting Machine y Deep Learning.

Sofia2 DataLink

Actúa de interfaz con productos de analítica, ofreciendo conectores estándar JDBC, ODBC y REST y una capa de abstracción que permite operar a través de SQL independientemente del origen de los datos. De esta manera, se facilita la integración tradicional a nivel de datos, con los repositorios BDTR y BDH indistintamente, pudiendo incluso realizar consultas en las que se combine información de ambos.

Por lo tanto, las **capacidades** que nos ofrece este módulos son las siguientes:

- Acceso **simultaneo** a múltiples fuentes de datos.
- Acceso a los datos a través de **SQL estándar**.
- Consultas sobre datos anidados en varios niveles.
- Creación de **vistas personalizadas**.
- **JOINS** entre repositorios.

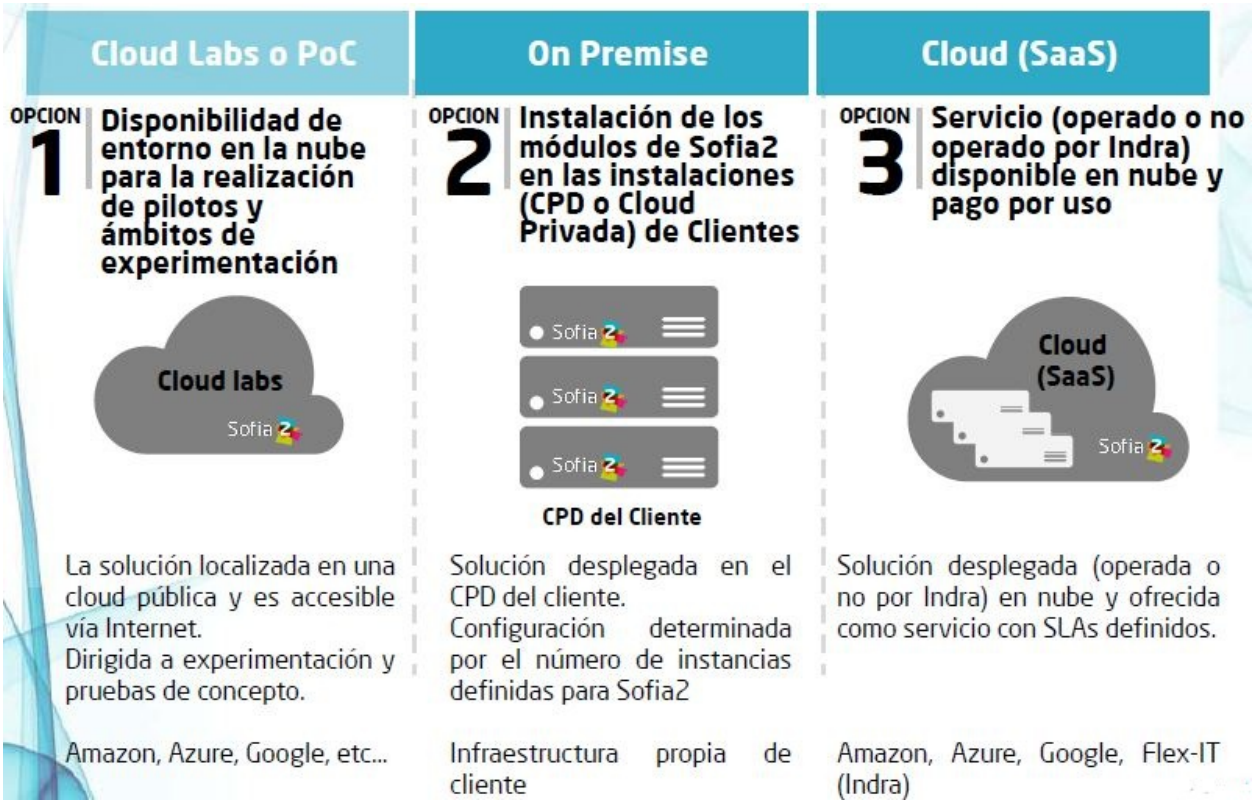


- Baja latencia.

CHAPTER 11

Despliegue

La plataforma está preparada para su despliegue según la conveniencia del proyecto o cliente, soportando:



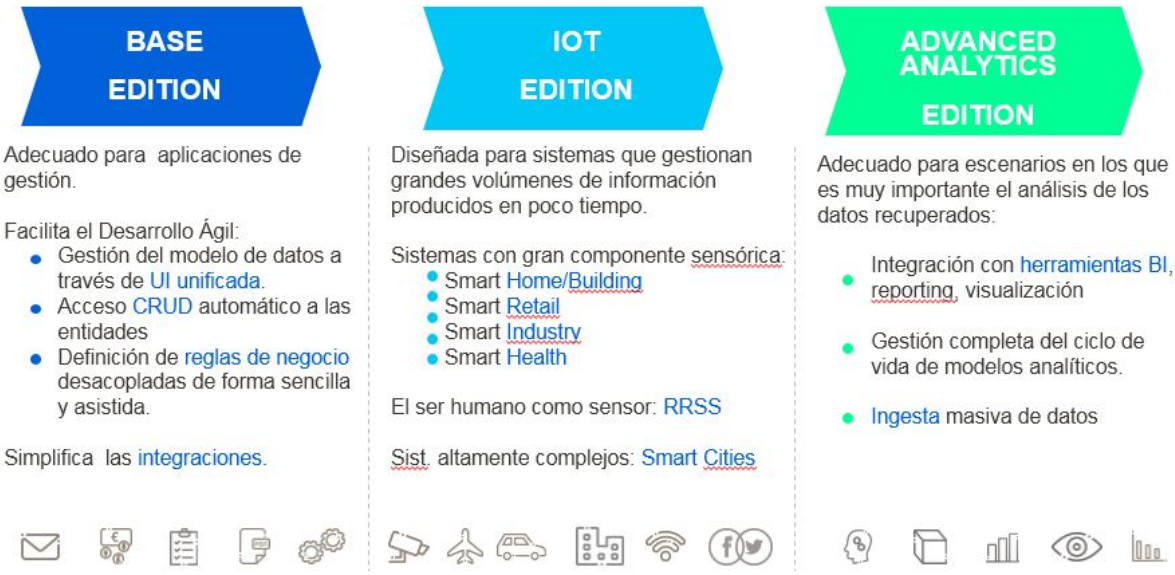
El número de nodos o VMs necesarias para desplegar la Plataforma depende del número de módulos que se usarán de la Plataforma y el uso que se les dará a estos.

Por ejemplo si la Plataforma va a soportar un gran volumen de dispositivos es importante que el IoT Gateway+Semantic Broker estén preparados para soportar esta carga, mientras que si la carga analítica de la plataforma es alta es importante dimensionar bien los módulos que soportan esto.

CHAPTER 12

Versiones de la Plataforma

A nivel comercial manejamos 3 versiones de la Plataforma, **Base Edition, IoT Edition y Advanced Analytics Edition**, que se posicionan así:

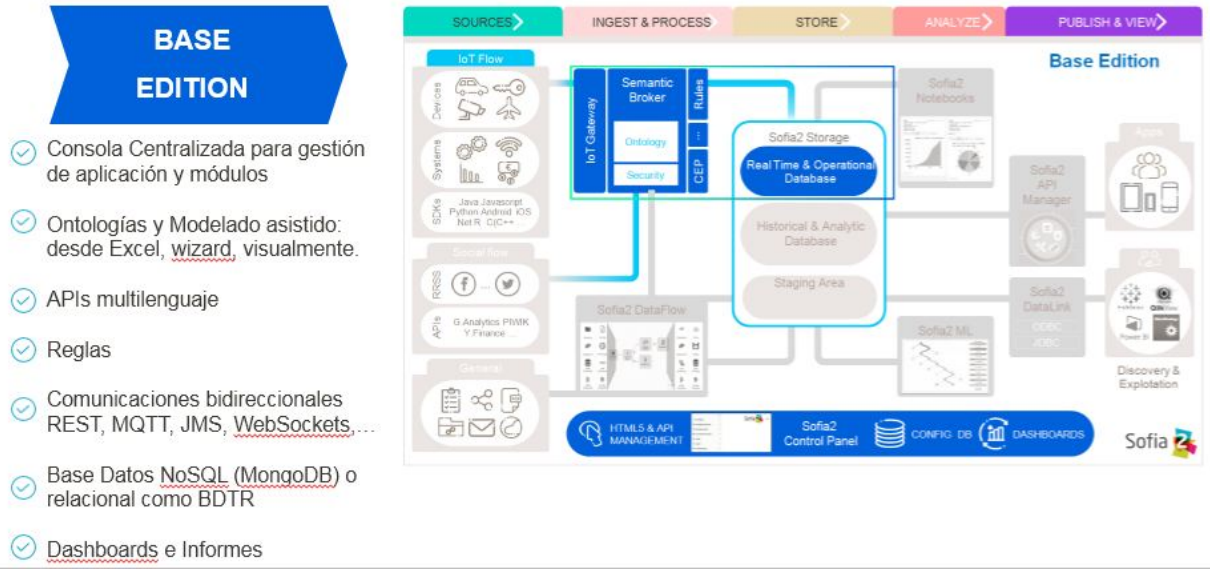


Base Edition

Incluye los siguientes módulos:

- IoT Gateway con conectores MQTT, HTTP REST, WebSockets.

- Semantic Broker.
- Process con Rules en Python, R y Groovy, CEP y Planner.
- Real Time Database (RTDB) sobre MongoDB.
- Sofia2 Control Panel incluyendo Dashboards y Reports.



IoT Edition

Además de los módulos de la versión Base incluye:

- Historical Database (HDB) sobre HIVE + Impala.
- Sofia2 API Manager.
- Sofia2 Holystic Viewer.
- Sofia2 Notebooks con soporte Spark, R y HIVE.

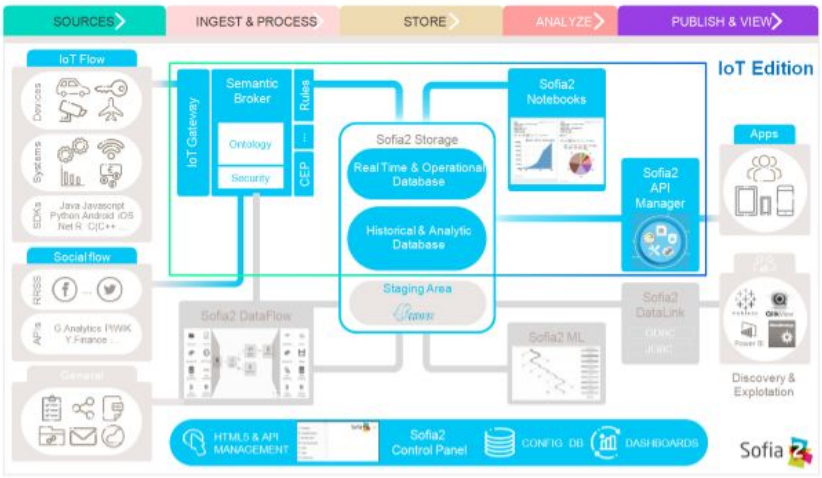
Advanced Analytics Edition

Esta versión añade a las capacidades para negocios IoT, capacidades analíticas Big Data que pueden complementar y ampliar el ámbito Internet of Things hacia el Analytics of Things.

Incluye todos los módulos de la plataforma.

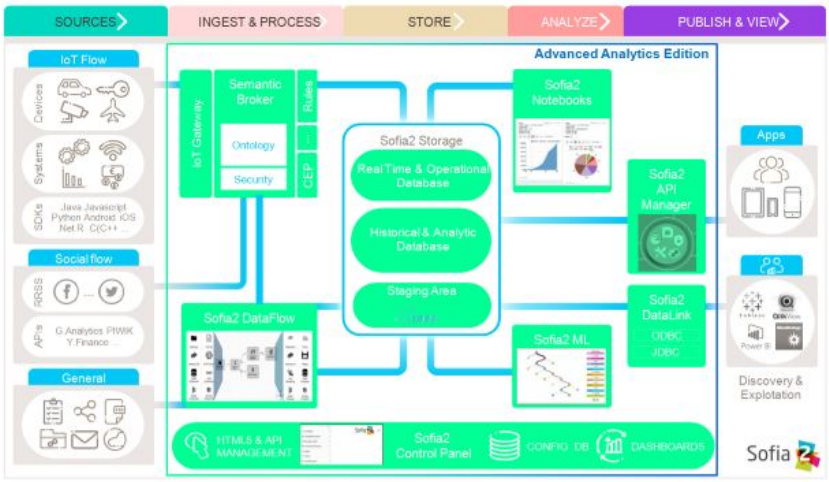
IOT EDITION

- Sobre Base Edition: Integración dispositivos y redes sociales
- Infraestructura Big Data sobre Hadoop como BDH
- API Manager para publicación de información como APIS REST
- Notebooks Web para hacer analítica interactiva
- Migración de BDTR a BDH automática y configurable



ADVANCED ANALYTICS EDITION

- Integración de cualquier fuente de datos de forma visual y asistida
- Módulo Machine Learning integrado que permite cargar datasets, lanzar algoritmos, crear modelos, publicarlos,…
- Módulo que permite acceder a la información de la Plataforma a través de interfaces ODBC y JDBC (herramientas BI,…)



Sofia 2

CHAPTER 13

Seguridad

La Plataforma ofrece diferentes mecanismos de seguridad que permiten garantizar la privacidad de los datos, el control de acceso a los mismos y el envío de información cifrada a través de sus comunicaciones.

La capa de seguridad es aplicada en diferentes niveles: **autenticación**, **autorización**, y **cifrado**, asegurando las siguientes características:

- **Comunicación segura y confidencial** en cualquier tipo de conexión, ya sea entre clientes y plataforma o en cualquiera de los puntos de acceso disponibles (UI, APIs. . .), a través de SSL y/o HTTPS.
- **Privacidad de los datos**: asegurando el acceso exclusivo de aquellos usuarios que disponen de las credenciales necesarias para el acceso a la información correspondiente.
- **Autenticación de los clientes de la Plataforma** en la comunicación con esta a través de diversos mecanismos: usuario+ password, token, certificado.
- **Autorización en el acceso a los datos**, permitiendo controlar a grano fino (por ejemplo quien puede insertar qué información y quien puede consultar qué información).
- **Capacidad de configurar y extender cada uno de los conceptos mencionados** a través de los mecanismos de extensión de la plataforma.
- **Gestión de usuarios y roles**: La plataforma consta de un sistema de roles gestionable desde la consola centralizada, permitiendo la asignación de roles a usuarios, asignación de permisos sobre información almacenada en la plataforma. Estos roles pueden almacenarse en diversos repositorios entre ellos LDAP.

CHAPTER 14

Escalabilidad

La robustez, escalabilidad y alta disponibilidad de la Plataforma se garantiza a diversos niveles:

- Volumen de almacenamiento.
- Procesado de datos.
- Velocidad de proceso de la información.
- Capacidad de respuesta en tiempo real.

La plataforma Sofia2 se construye sobre piezas que soportan y garantizan la robustez y escalabilidad horizontal.

Relacionado con esta escalabilidad horizontal está el hecho de que la plataforma esté concebida y pensada para correr sobre hardware commodity de modo que para aumentar la capacidad de procesamiento baste con incluir máquinas al clúster.

Escalabilidad Ingest + Process (IoT Gateway + Semantic Broker)

- Ofrece escalabilidad horizontal en el procesado de datos, de modo que incorporando nuevas máquinas a la Plataforma esta incrementa su capacidad de procesamiento de forma lineal.
- Ofrece robustez al estar basado en la Plataforma Java.
- Ofrece alta disponibilidad ya que se crea un clúster de instancias de modo que el trabajo se redistribuye automáticamente.

Escalabilidad almacenamiento (Sofia2 Storage)

El almacenamiento de la plataforma también cumple los criterios:

- Como BDH usa Hadoop: - vía su sistema de ficheros distribuido HDFS garantiza la alta disponibilidad. - Vía su Arquitectura YARN garantiza la escalabilidad horizontal.
- Como BDTR se usa MongoDB: - A través de sharding permite escalado horizontal. - A través de su Arquitectura garantiza robustez y alta disponibilidad con Arquitectura Maestro-esclavo.

Escalabilida Sofia2 PaaS

Permite el auto escalado de las siguientes formas:

- Escalabilidad Vertical: Está asociada al incremento de las capacidades de cómputo y/o memoria de la una instancia concreta.
- Escalabilidad Horizontal: Está asociada a la variación de número de instancias para mantener el nivel de servicio en base a la demanda real del sistema.
- Escalabilidad Híbrida: Modelo en el que se puede optar por ambas opciones dependiendo de las necesidades del proyecto.

Sofia2-PaaS ofrece capacidades de elasticidad Horizontal transparente, es decir, analiza la demanda del servicio y realiza automáticamente los procesos necesarios de provisión y des-provisión de la infraestructura necesaria para cubrir dicha demanda.

CHAPTER 15

Arquitectura de Referencia

Componentes de Despliegue

Los componentes de Sofia2 son:

- **SIB Runtime**: Es el core de la plataforma, encargado de procesar y almacenar todos los mensajes recibidos por Sofia2. Expone interfaces (MQTT, WebSockets, REST,...) para que los clientes puedan publicar y consultar información.
- **Web Console**: Consola Web + API Web para gestionar/administrar la plataforma. Facilita el modo de configurar Sofia2
- **API Manager**: Permite exponer la información de negocio del dominio de Sofia2 mediante APIs REST. Asimismo, puede actuar de punto centralizador de invocación a otras APIs REST externas.
- **SIB Tools (Script)**: Es el módulo que proporciona capacidad de ejecución de reglas Script y CEP.
- **SIB Tools (Proccess)**: Es el módulo encargado de realizar procesos en backgroud en la plataforma Sofia2, entre ellos, paso de información de BDTR a BDH, limpieza de registro de log de procesos. Borrado de información en BDTR que ha dejado de pertenecer a la ventana de tiempo real.
- **Contenedor KP's**: Proporciona un entorno para la ejecución de determinados KPs pertenecientes a usuarios privilegiados de la plataforma Sofia2.
- **BDC**: Almacena toda la información de onfiguración de la Plataforma. Puede ser cualquier BD relacional con driver JDBC. Actulamente Sofia2 está certificada sobre MySQL y Oracle.
- **BDTR**: Almacena los datos del tiempo real de las Apps que utilizan la plantaforma. La implementación de referencia está certificada sobre MongoDB, aunque con las limitaciones impuestas por las propias bases de datos, se pueden soportar cualquier BD relacional con driver JDBC.
- **BDH**: Almacena la información de la plataforma que ha dejado de pertenecer a la ventana de tiempo real y se considera como información histórica. Se soporta sobre infraestructura Hadoop.

En una instalación de Sofia2 no es necesario instalar todos los módulos, ya que cada módulo cubre un conjunto de funcionalidades bien diferenciadas. Dependerá de cada caso concreto que módulos se instalan en la instalación de Sofia2.

Cada módulo Sofia2 puede funcionar en instancia única o en cluster, dependiendo de la carga de trabajo a la que se vaya a someter.

Plugins

Los plugins son componentes que, añaden una funcionalidad adicional o una nueva característica a la platafoma, sin tener que modificar el core de la misma. Los plugins se empaquetan como fichersos JAR (Java Application Resource), cuyas cláses se incluirán en tiempo de ejecución, junto con el resto de clases del core de la plataforma, añadiendo de este modo nuevas funcionalidades..

Existen distintos tipos de plugins:

- **Seguridad**: Permiten personalizar el mecanismo de autenticación/autorización en los módulos de Sofia2, adaptándose al módelo existente en la organización donde se despliega Sofia2.
- **Plugin-Gateway**: Permiten añadir nuevos protocolos e interfaces de comunicación para el envio/recepción de información desde/hacia Sofia2
- **Plugins de pre/post procesado**: Permiten realizar ciertas acciones con un mensaje recibido por la plataforma antes y/o después de ser procesado
- **Plugins de plataforma**: Son plugins desarrollados a medida para Sofia2. Realizan funciones de integración entre componentes de la plataforma.

CHAPTER 16

BDH

Requisitos Previos

Instalación

CHAPTER 17

BDC

Requisitos Previos

Instalación

CHAPTER 18

BDTR

Requisitos Previos

Instalación

CHAPTER 19

Consola Web de Configuración

CHAPTER 20

SIB

CHAPTER 21

Script

CHAPTER 22

Documentación Básica

Las Guías para conocer temas básicos que aplican a Sofia2 son las siguientes:

Mecanismos de Seguridad

Esta guía describe los mecanismos de seguridad que aplican a Sofia2.

La plataforma Sofia2 es un entorno de interoperabilidad controlado, que implementa varios mecanismos de seguridad que operan a lo largo de las diferentes capas ofreciendo autorización, autenticación, consistencia y en su conjunto protección integral de los datos.

La seguridad en la plataforma Sofia2 es implementada a través del mecanismo de plugins que permiten realizar una personalización total de la Autenticación y la Autorización, permitiendo implementar mecanismo basados en diferentes estándares (LDAP, BD, Oauth. . .).

La Seguridad cubre las diferentes capas del aplicativo:

- **Las Comunicaciones**: Todos los Gateway implementados en la plataforma permiten la securización del canal a través del uso de Certificados SSL.
- **Los Clientes:** La plataforma solo permite la conexión de aquellos clientes que han sido autorizados en la plataforma.
- **Las Operaciones**: La plataforma solo permite realizar las operaciones para las que se tiene permiso.

- **La Información**: La plataforma solo te permite interactuar con la información para la que has sido autorizado, y valida que la información que se persiste en la plataforma cumple con el modelo de la información esperado.

Seguridad en las Comunicaciones

Sofia2 utiliza el canal seguro de comunicaciones https para todas las comunicaciones con el exterior. Este canal utiliza un protocolo SSL/TSL con un cifrado estándar RSA que necesita de un certificad de seguridad X.509. Es el estándar de cifrado más extendido y aceptado en el mundo del www.

La clave privada se instala en el SIB Sofia2 y su parte pública (conocida como certificado) deberá instalarse en los navegadores o ser incorporado en las apps que necesiten acceder a la plataforma desde la extranet. La seguridad mediante certificados prevendrá filtrado o manipulación de datos entre cliente y servidor mediante ataques del tipo man-in-the-middle.

Hay que destacar que el sistema de autenticación será mediante certificado de servidor, no de cliente-servidor (como podrían ser los certificados de la fnmt). Esta tipo de encriptado garantiza que el cliente se está conectando con el servidor requerido y no a un posible servicio impostor, pero no autentica al cliente.

Seguridad en los Clientes

Un usuario deberá registrar en la plataforma sus KPs (Cliente), de lo contrario, la plataforma rechazará la conexión de los mismos.

Para ello necesitará de un Token de Autenticación que el usuario podrá crear asociado al KP (según las restricciones de su rol). Los usuarios con rol Colaborador o rol Usuario únicamente podrán dar de alta tokens para KP's de los que sean propietarios.

Los KP's tienen también una clave de cifrado que sólo es necesaria si quiero usar XXTEA como protocolo de encriptación. Se utiliza en dispositivos que no soportan HTTPs, como por ejemplo Arduinos.

Un KP podrá hacer uso de una o varias ontologías, siendo esta la información que producirá o consumirá de la plataforma.

Una vez registrado en la plataforma, el KP ya podrá establecer conexiones con la misma.

Tras la creación de un KP podemos dejar definida una instancia KP a través de la Consola Web.

Una instancia KP identifica al cliente que se va a conectar a la plataforma Sofia2.

La conexión de un KP con la plataforma debe ser vista como dos tipos de conexión

- **Conexión Física**: Establecida por el protocolo de trasporte utilizado para la conexión por un KP (TCP/IP, MQTT, WebSockets, REST, WS, JMS, Ajax-Push. . .). La manera de realizar esta tipo de conexión depende en gran medida del API de KP utilizado (Java, Javascript, Arduino, C++...).
- **Conexión Lógica**: Establecida por el protocolo SSAP (Smart Space Access Protocol) de mensajería definido en SOFIA. Es común a todos los APIs de KP.

Nos vamos a centrar en la seguridad a nivel de conexión Lógica que debe mantener un KP con la plataforma, pues la seguridad a nivel Física es cubierta por la seguridad en la capa de Comunicaciones.

Para que un KP pueda conectarse a la plataforma y producir/consumir datos e interoperar con otros KP, es necesario que abra una sesión con un SIB de la plataforma.

El protocolo SSAP proporciona dos operaciones en este sentido:

- **JOIN**: Donde un KP informa a la plataforma el usuario y password de su propietario así sus datos de instancia, de manera que si todo es correcto, la plataforma autentica al KP y abre una sesión con el mismo (devolviendo una session key que podemos usar durante un tiempo preestablecido y configurable).

- **LEAVE**: Donde un KP informa a la plataforma que va a cerrar la sesión.

Mientras exista una sesión entre el KP y la plataforma, el KP podrá utilizar el resto de operaciones del protocolo SSAP para producir/consumir información.

Seguridad de Acceso

La plataforma está dividida en dos áreas con independencia en su modelo de seguridad.

La Administración.

En la plataforma se han definido tres tipos de Roles, que definen las funcionalidades que dispondrán los usuarios a nivel administrativo:

- Rol **Administrador**.
- Rol **Colaborador**: este rol permite operar con la información de la plataforma, volcando y consumiento información y crear nuevas estructuras de información además de realizar tareas de procesado de información (Reglas, Script, Informes).
- Rol **Usuario**: este rol permite operar con la información de la plataforma, volcando y consumiento información de estructuras de información existentes en las que ha sido autorizado.

La Operación.

Los permisos, para los que también existen 3 tipos, definen las funcionalidades de los usuarios a nivel operativo sobre la información.

Estos permisos se aplican a nivel de Ontología – Usuario.

Los **administradores** tienen Permiso Total sobre todas las Ontologías.

Los **Colaboradores** tienen Permiso Total sobre las ontologías de las que son Propietarios:

- **Permiso de Lectura**: Permite a un usuario o colaborador realizar únicamente operaciones de tipo Query sobre las ontologías para las que se le ha asignado este permiso.
- **Permiso de Inserción**: Permite a un usuario o colaborador realizar únicamente operaciones de tipo Insert sobre las ontologías para las que se le ha asignado este permiso.
- **Permiso Total**: Permite a un usuario o colaborador realizar todas las operaciones sobre las ontologías para las que se le ha asignado este permiso.

Seguridad en los Datos

Todas las operaciones son validadas a nivel de Autenticación, para lo que la plataforma comprueba si el Cliente se ha autenticado con la plataforma.

Una vez que se ha comprobado la Autenticación del Cliente se comprueba su autorización en dos niveles:

- Primero, se valida que el usuario puede operar con la Ontología para la que quiere realizar la operación.
- Segundo, se valida que la operación (Query, Insert, Delete, Update) que quiere realizar el usuario, la puede realizar sobre esa ontología (Tiene los permisos adecuados).

Si todos los pasos anteriores han sido correctamente comprobados y la operación es Insert o Update todavía se ha de realizar una tercera validación, que consisten en comprobar que la información que se inserta cumple escrupulosamente con el Esquema que se ha definido, a través de la validación del JSON Schema, casando la información que está insertando con la estructura de la Ontología.

Implementación de Referencia

La implementación de referencia de la Seguridad está basada en tres plugins:

plugin-sofia-user

Este plugin (Usado únicamente a nivel de Administración) es el encargado de recuperar la información de los usuarios. En la implementación de referencia la recupera de la base de datos de configuración.

Tiene la capacidad de trabajar con Password encriptada o en claro, permitiendo configurar el Algoritmo de encriptación.

public void persist(Usuario user) **throws** NotImplementedException;

public void remove(Usuario user) **throws** NotImplementedException;

public void merge(Usuario user) **throws** NotImplementedException;

public long countUser() **throws** NotImplementedException;

public List<Usuario> findAllUser() **throws** NotImplementedException;

public Usuario findUser(String idUsuario) **throws** NotImplementedException;

public List<Usuario> findUsers(String qlString, List<Object> parametros) **throws** NotImplementedException;

public Usuario findLoginUser(String identificador, String credential, String sourceInfo) **throws** EmptyResultDataAccessException;

public List<Usuario> findUserByCriteria(Usuario usuario) **throws** NotImplementedException;

public List<Usuario> findUserByIdentificacion(String identificacion) **throws** EmptyResultDataAccessException;

plugin-console-security

Este plugin es el encargado de gestionar la Autenticación y Autorización en la consola de Administración y se basa en Spring Security. En la implementación de referencia hace uso de la base de datos de configuración.

Hace uso de plugin-sofia-user para recuperar la información de los usuarios.

plugin-sib-security

Este plugin es el encargado de gestionar la Autenticación y Autorización a las operaciones del SIB y está basado en un mecanismo de Token – SessionKey.

Hace uso de plugin-sofia-user para recuperar la información de los usuarios.

Este plugin debe cumplir con el siguiente interface:

String authenticate(SSAPMessage message) **throws** AuthenticationException;

void checkSessionKeyActive(String sessionKey) **throws** AuthenticationException;

void closeSession(String sessionKey) **throws** AuthenticationException;

public void removeAuthenticationContextSessionkey(String sessionKey) **throws** AuthenticationException;

void checkAuthorization(SSAPMessageTypes operationType, String ontologyName, String sessionKey) **throws** AuthorizationServiceException;

void checkAuthorizationConfig(SSAPMessageTypes operationType, String tableName, String sessionKey) **throws** AuthorizationServiceException;

void checkAuthorization(SSAPMessageTypes operationType, String kpName, String instanceKpName, String token) **throws** AuthorizationServiceException;

TCO de Sofia2 vs desarrollo a medida sobre una base relacional

A continuación se describe el razonamiento empresarial para implementar Sofia2 en vez de un desarrollo tradicional con una base de datos relacional. Este informe hace una comparación entre el Coste Total de Propiedad de la Plataforma Sofia2 y el de un desarrollo a medida con una base de datos relacional, considerando los costes iniciales y corrientes (software, hardware y personal).

Coste Total de Propiedad (TCO)

Costes Iniciales

Los costes iniciales se componen de:

- **Esfuerzo de desarrollo inicial**: Coste de personal + Programación del desarrollador necesaria para la aplicación
- **Esfuerzo administrativo inicial**: Coste de personal + Administradores para instalar y configurar software, máquinas del clúster, particionado, ...
- **Licencias de software**
- **Hardware de servidores**. Servidores necesarios para ejecutar la base de datos (se excluye almacenamiento). Depende principalmente del número y tipo de procesadores y RAM. Otros costes incluyen recintos, conexiones de red, cableado y suministros de alimentación
- **Hardware de almacenamiento**. Almacenamiento necesario para almacenar los datos, varía en función de si se utiliza almacenamiento interno o compartido (SAN), de la cantidad de almacenamiento y de si se utilizan unidades de disco duro (HDD) o unidades de estado sólido (SSD).

Costes Corrientes

Los costes corrientes se componen de:

- **Esfuerzo de desarrollo corriente**: Personal + Programación necesaria para adaptar el almacén de datos a las necesidades del cliente, del mercado y empresariales

- **Esfuerzo administrativo corriente**: Personal + Esfuerzo administrativo necesario para mantener el funcionamiento y ejecución del almacén de datos
- **Mantenimiento y soporte técnico del software**: Mantenimiento: actualizaciones y soluciones de errores del software + Soporte técnico: asistencia para localizar y solucionar problemas técnicos en el software
- **Mantenimiento y soporte técnico del hardware**: Mantenimiento: actualizaciones y soluciones de errores del firmware y cualquier software que pueda incluir el hardware + Soporte técnico: asistencia telefónica para localizar y solucionar problemas técnicos en el hardware
- **Costes de despliegue diversos**: Otros costes necesarios para mantener la base de datos en funcionamiento. Incluye costes de nube/alojamiento/coubicación, costes de ancho de banda, tarifas eléctricas, etc.

Comparación del coste total de propiedad

A continuación veremos como Sofia2 reduce los diversos costes que componen el TCO de un sistema.

Esfuerzo de desarrollo inicial

El esfuerzo de desarrollo inicial se refiere al coste del tiempo dedicado por el desarrollador para conseguir que la aplicación y el almacén de datos trabajen juntos.

En el caso de un desarrollo sobre base de datos relacional, el esfuerzo de desarrollo inicial incluye tareas como definir el modelo de datos, crear una capa de mapeo objeto-relacional (ORM), escribir la lógica empresarial para la aplicación y hacer la capa de presentación para esta lógica.

Sofia2 está diseñado para que reducir los tiempos de desarrollo, de modo que un desarrollador en cualquier lenguaje pueda utilizar la Plataforma con facilidad.

Para eso a través de la Consola Sofia2 (Sofia2-Console) el desarrollador puede:

- Crear sus entidades (Ontologías en Sofia2, tablas en un SGBDR, colecciones en MongoDB)
- Definir sus reglas de negocio de forma sencilla y asistida
- Establecer seguridad en el acceso a sus entidades
- Acceso CRUD (consulta, inserción, borrado, actualización,...) a todas estas entidades a través de cualquier lenguaje (Java, Javascript, C, Android,...) lo que le permite desarrollar tanto aplicaciones Web MVC (API Java, Python, Node.js), aplicaciones HTML5 (API Javascript), aplicaciones móviles (API Android, iOS, Javascript...) o módulos de negocio (Java, Python, C,...)
- Capacidad de suscripción a eventos, consultas, reglas, ...de forma sencilla e independiente del protocolo de mensajería (JMS, MQTT, AMQP,...)
- Publicación asistida y web de APIS REST a partir de las entidades
- Capacidades GIS integradas
- Dashboards integrados
- Informes integrados
- Repositorio Big Data integrado

Por lo tanto, podemos decir que resulta mucho más rentable desarrollar con Sofia2 que hacer un desarrollo a medida sobre bases de datos relacionales.

Otra ventaja de productividad importante de Sofia2 es su diseño de Entidades (Ontologías) orientado a documentos y a los esquemas dinámicos. La forma en que almacena datos de la aplicación se corresponde con la tecnología y

prácticas de desarrollo actuales, que han evolucionado considerablemente desde los comienzos de la industria de las bases de datos relacionales hace 30 años.

Algunos motivos que respaldan las ventajas de productividad de Sofia2 son:

- **Facilidad de uso**: Sofia2 es compatible con las metodologías de desarrollo actuales, permite a los desarrolladores realizar iteraciones de forma rápida y continua sobre el modelo de datos y todo desde un interfaz Web. En contraposición un desarrollo tradicional modelo relacional impone un estricto conjunto de limitaciones al desarrollo, tanto a nivel de modelo de datos, de creación de reglas, cambios,...
- **Modelo de datos.** Con Sofia2, el desarrollador solo tiene que crear el modelo de datos en un lugar: la Consola Web del propio producto. En un desarrollo los desarrolladores necesitan crear y mantener el modelo de datos en tres lugares mediante el uso de diferentes interfaces: la aplicación, la propia base de datos y la capa ORM.
- **Flexibilidad de datos**. A diferencia de una SGBDR, Sofia2 permite a los desarrolladores almacenar con facilidad datos polimórficos, así como datos semiestructurados y estructurados, en un almacén de datos individual.
- **Soporte JSON**. El almacenamiento en JSON, pilar básico de numerosas aplicaciones actuales, se realiza sin dificultades y no requiere conversión. Con una SGBDR, los desarrolladores necesitan "aplanar" y transformar JSON para almacenarlo en tablas relacionales, y más tarde tienen que recuperar las capas al realizar la extracción de la base de datos.

Esfuerzo administrativo inicial

La instalación y configuración de Sofia2 es económica y sencilla.

La Plataforma se compone de :

- **BDC (Base Datos Configuración)** : puede ser cualquier base de datos relacional. Por defecto funciona sobre una BD embebida MySQL.
- **BDTR (Base Datos Tiempo Real)**: en la RI es un MongoDB lo que hace que el esfuerzo administrativo inicial sea bajo, un administrador solo debe tener en cuenta una variable: el número de nodos en el clúster. Solo existe un reducido conjunto de ajustes de configuración para poner el sistema en funcionamiento. Los administradores de MongoDB no necesitan integrar capas de memoria caché ni crear lógica de particionado horizontal personalizada para dirigir las consultas al nodo servidor correcto. En lugar de esto, el almacenamiento en memoria, caché y el particionado horizontal son capacidades centrales de MongoDB.
- **BDH** (**Base Datos Histórica**): puede funcionar sobre MongoDB o Hadoop en función de las necesidades o preferencias.
- **SIB + Consola + Tools + API Manager + Process**: todos los módulos de negocio de la Plataforma están construidos en Java, se despliegan como aplicaciones Web en cualquier servidor de aplicaciones JEE. El grueso de la configuración va en la BDC por lo que no es necesario crear ficheros de configuración complejos.

Licencias de software

Sofia2 es una Plataforma con una versión gratuita para la comunidad de código abierto (licencia Apache) y una edición para suscriptores comerciales que puede usarse en modo On Premise o en Modo Cloud.

Esta versión incluye soporte técnico en diferentes modalidades (desde 8x5 sin SLAS a 24x7 con SLAS estrictas), actualizaciones de software y soluciones de errores y algunas funciones adicionales.

La edición comercial de Sofia2 se factura de forma continua en lugar de puntualmente (esto es, una cuota anual por servidor).

Hardware de servidores

En general, los costes de servidores de Sofia2 son considerablemente inferiores a los de un desarrollo tradicional sobre BD relacional para cargas de trabajo y disponibilidad similar. Esto aplica a todos los componentes.

Sofia2 se diseña para utilizar hardware básico en arquitecturas escalables.

Los despliegues de Sofia2 normalmente utilizan servidores Linux básicos y económicos, que tienen un coste de tan solo 3.000 $; incluso un sistema de baja energía y alto rendimiento puede costar tan solo 4.000 $ (excluyendo almacenamiento).

Hardware de almacenamiento

La arquitectura escalable de Sofia2 permite reducir considerablemente los costes de almacenamiento.

Sofia2 puede utilizar el almacenamiento local económico y permite realizar un uso eficiente de las unidades de estado sólido (SSD).

Esfuerzo de desarrollo corriente

Las dinámicas del esfuerzo de desarrollo corriente son menores a las del esfuerzo de desarrollo inicial.

Con una desarrollo tradicional, el coste de realizar cambios en la aplicación es mayor, bien sean cambios en el esquema de una base de datos que ya se encuentre en producción (coste mayor que para una base de datos que aún no se ha entregado), como en el desarrollo de la lógica, reglas, seguridad, configuración).

Por ejemplo con Sofia2 resulta fácil para los desarrolladores agregar campos a las entidades, crear nuevas APIs, lo que se deriva en costes considerablemente inferiores y permite a los desarrolladores adaptar las aplicaciones a medida que evolucionen las demandas.

Esfuerzo administrativo corriente

El esfuerzo administrativo corriente incluye actividades que mantienen el sistema en buen estado de funcionamiento (por ejemplo, actualización del software y hardware, realización de copias de seguridad y recuperación de tiempos de interrupción inesperados).

Se requiere mucho menos tiempo y esfuerzo para administrar Sofia2 en comparación con un desarrollo tradicional.

La administración de un despliegue de Sofia2 implica principalmente administrar configuraciones de Linux y el propio hardware; solo es necesario conocer y administrar unos pocos parámetros.

Mantenimiento y soporte técnico

Las suscripciones de Sofia2 se facturan anualmente por core. Esto incluye el acceso al soporte técnico del producto, actualizaciones de software y soluciones de errores, así como ciertas funciones que solo se ofrecen en la edición de pago.

Otras ventajas de Sofia2

Resumiendo, además de los ahorros de costes tangibles, el modelo orientado a documentos y el esquema flexible de Sofia2 también aportan mayor agilidad y flexibilidad a las empresas, que a su vez proporcionan ventajas para generar ingresos.

Una vez implantada la Plataforma Sofia2 en una empresa esta puede utilizar la Plataforma (sin necesidad de montar nueva infraestructura) para hacer nuevos desarrollos y para integrar datos de otros sistemas de forma que los tenga centralizados en un repositorio común y con capacidades Big Data. Además puede desarrollar aplicaciones Sofia2 en cualquier tecnología y lenguaje.

Además se podrá descargar la siguiente documentación como primera toma de contacto con Sofia2:

Welcome Pack Sofia2.

Descargar PDF

Presentación Sofia2.

Descargar PDF

Soluciones Sofia2.

Descargar PDF

CHAPTER 23

Documentación para Usuarios.

Consola Web de Configuración

Descargar PDF

Modelado de Ontologías

A continuación se explica de forma clara y sencilla cómo se definen las ontologías en la Plataforma.

Conceptos Básicos

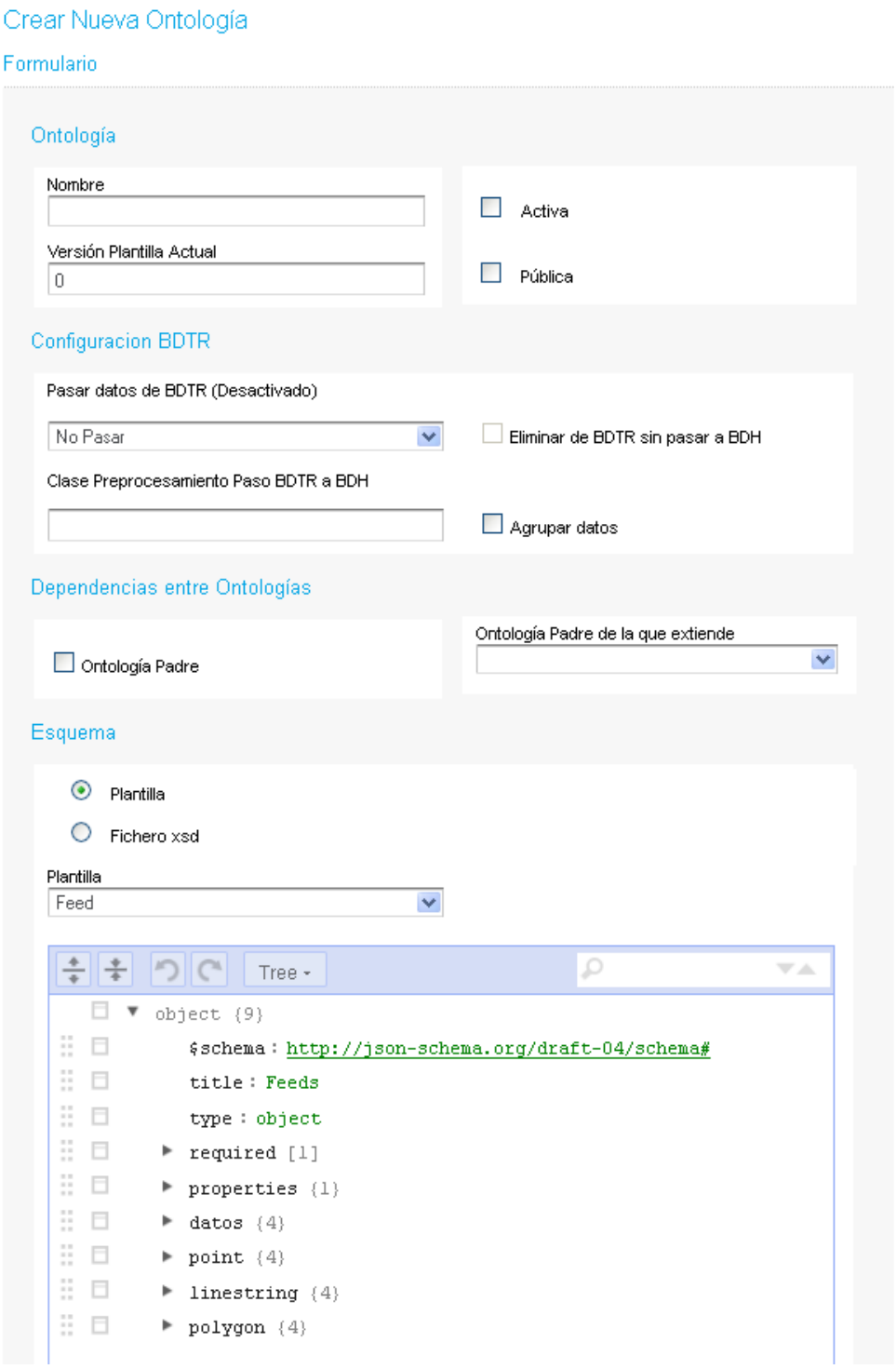
- **JSON**: es un formato ligero para el intercambio de datos (como XML pero menos verboso)
- **JSON-Schema:** un esquema JSON es un documento JSON que permite especificar cómo es un documento JSON al que se refiere (si hay atributos obligatorios, si son de tipo number, si pueden ser nulos). En la equivalencia XML correspondería con un esquema XML o con un DTD.
- **Ontología** define formalmente un conjunto común de términos que se usan para describir y representar un dominio.

- **Ontología SOFIA2:** En SOFIA2 un ontología es la definición del conjunto de clases y atributos de las mismas que van a compartir las distintas aplicaciones que interoperan dentro del SmartSpace. En SOFIA2 las ontologías de definen en JSON conforme a un esquema JSON.
- **Instancia de Ontología:** es un elemento concreto de una ontología

Un primer vistazo

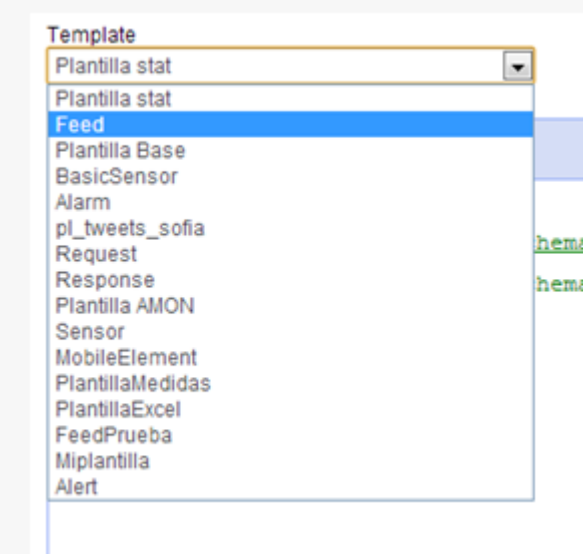
Como hemos dicho en SOFIA2 una **ontología** representa una entidad en mi sistema (SmartSpace), y esta se define en **JSON**.

La Plataforma ofrece una **Web (+API REST) de Configuración** en la que los usuarios con permisos (colaboradores y administradores) pueden crear sus ontologías.



Una ontología se define a partir de un **esquema JSON**.

Para simplificar la creación de Ontologías la Plataforma ofrece el concepto de Plantillas, que son esquemas JSON precargados que puede usar y ampliar el usuario para crear sus ontologías:

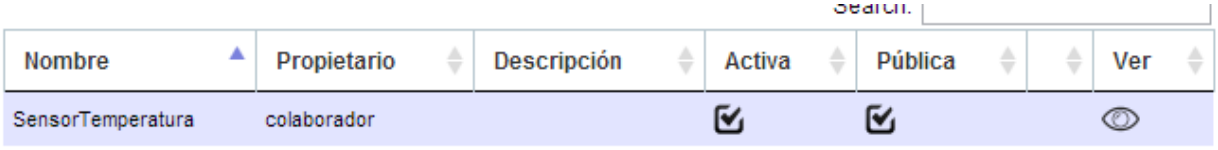


Comencemos con la definición de un Ontología sencilla como la que representa un Sensor de Temperatura que almacena: identificador, timestamp, medida, unidad y coordenadas GPS.

Una instancia de esta ontología sería algo como:

```
{
  "SensorTemperatura": {
  "identificador":"ST-TA3231-1",
  "timestamp":{"$date": "2014-01-27T11:14:00Z"},
  "medida":25.1,
  "unidad":"C",
  "geometry":
  {
   "type": "Point",
   "coordinates":[90,-10.1]
  }
 }
}
```

Puedo ver cómo se define el esquema JSON de esta ontología en SOFIA2:



Esta ontología es pública, lo que implica que cualquier persona puede consultar datos de esta.

Si pinchamos Ver  veremos el esquema JSON que describe esta Ontología (en posteriores apartados entraremos en detalle sobre la sintaxis de este esquema):

```
 {
  "$schema":"http://json-schema.org/draft-04/schema#",

  "title":"SensorTemperatura Schema",

  "type":"object",
```

```
 "required":["SensorTemperatura"],
 "properties":
 {
 "_id":
  {
   "type":"object",
   "$ref":"#/identificador"
  },
 "SensorTemperatura":
  {
   "type":"string",
   "$ref":"#/datos"
  }
 },
"additionalProperties":false,
"identificador":
{
 "title":"id",
 "description":"Id insertado del SensorTemperatura",
 "type":"object",
 "properties":
 {
  "$oid":
  {
   "type":"string"
  }
 },
"additionalProperties":false
},

"datos":
{
 "title":"datos",
 "description":"Info SensorTemperatura",
 "type":"object",
 "required":["identificador","timestamp","medida","unidad","coordenadaGps"],
 "properties":
 {
 "identificador":
 {
   "type":"string"
 },
 "timestamp":
 {
  "type":"object",
  "required":["$date"],
  "properties":
   {
    "$date":
     {
       "type":"string",
       "format":"date-time"
     }
   },
"additionalProperties":false
 },
 "medida":
 {
```

```
    "type":"number"
  },
 "unidad":
 {
   "type":"string"
 },
 "geometry":
 {
  "$ref":"#/gps"
 }
 },
 "additionalProperties":false
 },

 "gps":
 {
 "title":"gps",
 "description":"Gps SensorTemperatura",
 "type":"object",
 "required":["coordinates","type"],
 "properties":
 {
 "coordinates":
 {
 "type":"array",
 "items":[
 {
 "type":"number",
 "maximum":180,
 "mininum":-180
 },

 {
 "type":"number",
 "maximum":180,
 "mininum":-180
 }
 ],
 "minItems":2,
 "maxItems":2
 },
 "type":
 {
  "type":"string",
  "enum":["Point"]
 }
 },
 "additionalProperties":false
 }
}
```

NOTA

En el esquema podemos observar que las propiedades ***timestamp*** y ***geometry*** están definidas de una forma especial. Esto es así para permitir realizar búsquedas por fecha y geoespaciales en MongoDB. Veamos cada caso en particular:

- **timestamp**: MongoDB permite trabajar con fechas en formato ***ISO 8601*** date (**YYYY-MM-DDThh:mm:ss.fffZ).** Para que MongoDB interprete que un campo es de tipo fecha, espera recibir un JSON con una estructura similar a {"$date": "2014-01-27T11:14:00Z"} representados. El esquema que nos permite validar instancias de este tipo es el siguiente:

```
{ "timestamp":{
"type":"object",
"required":["$date"],
"properties":{
"$date":{
"type":"string",
"format":"date-time"
}
},
"additionalProperties": false
}
}
```

Esto nos permite realizar consultas como la siguiente :

```
db.SensorTemperatura.find({"Sensor.created":{"$lt": new ISODate()}});
```

- **geometry:** MongoDB permite realizar consultas geoespaciales y para ello requiere que los campos que hayan de tratarse con este fin sean definidos, con la siguiente estructura:

```
{ "geometry": {
"type": "object",
"required":["coordinates","type"],
"properties":{
"coordinates":{
"type":"array",
"items":[
{
"type":"number",
"maximum": 90,
"minimum": -90
},
{
"type":"number",
"maximum": 180,
"minimum": -180
}
],
"minItems":2,
"maxItems":2
},
"type":{
"type":"string",
"enum":["Point"]
}
},
"additionalProperties":false
}
```

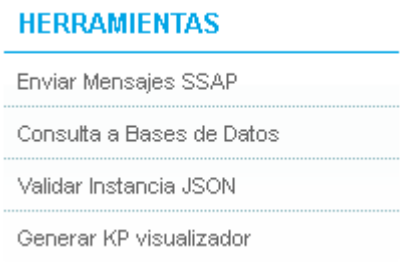
La propiedad *geometry*, está compuesto del tipo "Point" y de unas coordenadas, que representa un punto, dado por la latitud y longitud ("coordenates":[Latitud,Longitud]). El rango de valores que soporta MongoDB para este tipo de coordenadas está entre [90, -90] para las latitud y [180,-180] para la longitud. Si se intenta insertar un valor fuera del rango, MongoDB retornará error.

Una instancia que cumple con esta estructura: {"geometry ": {"type":"Point", "coordinates":[1.9, -3.9]}}

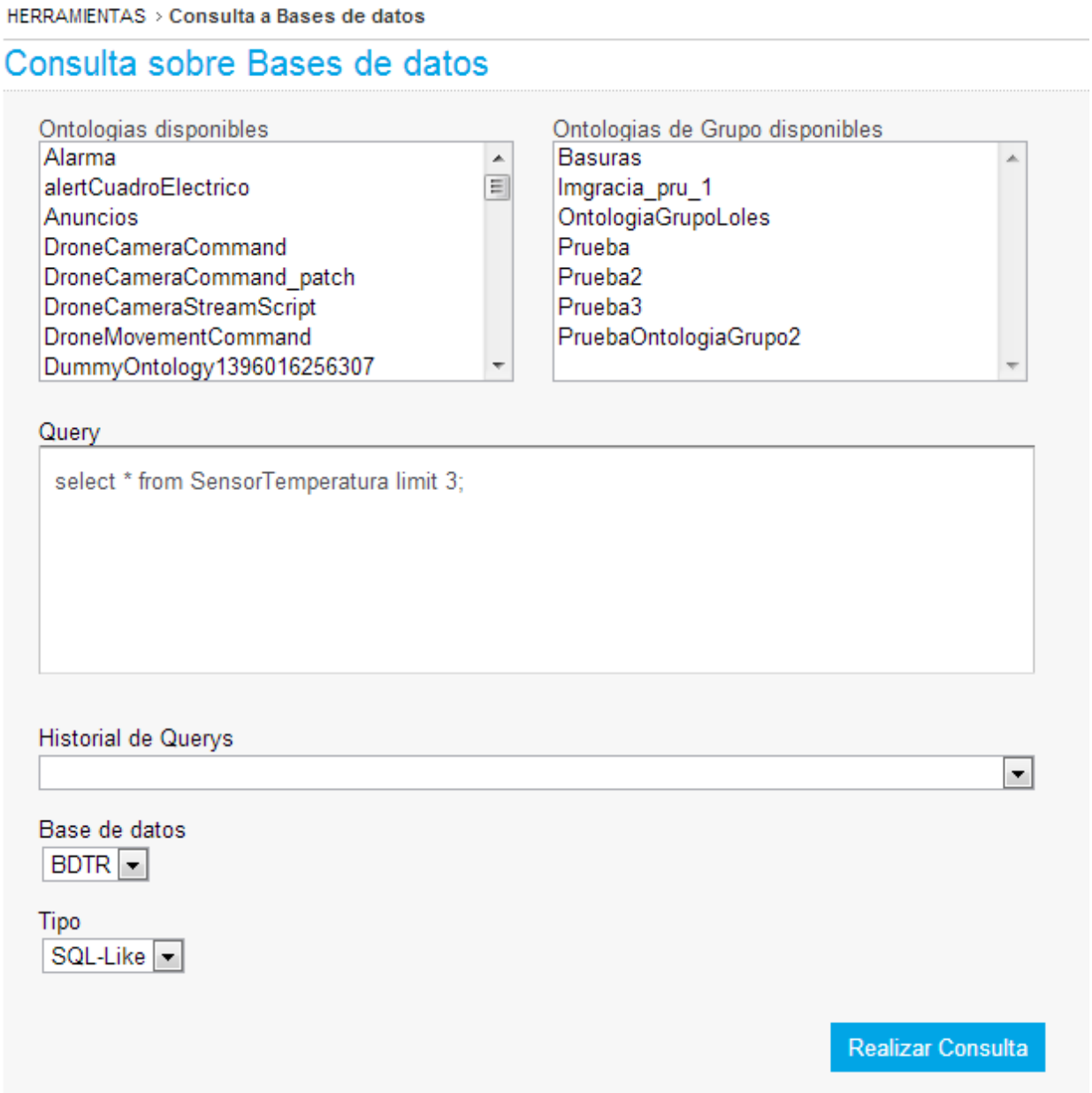
Podremos realizar búsquedas geoespaciales en MongoDB como la siguiente:

```
db.SensorTemperatura.find({"Sensor.geometry.coordinates":{$near:[12,12],$maxDistance:1}})
```

Puedo ver las **instancias de mis ontologías** desde la Web de Configuración a través de la opción Consulta a Base de Datos:



Si en esta consulta lanzo una consulta de este estilo:



Veré la información de la última instancia insertada en la BDTR de SOFIA2:

```
{
 "id":
 {
  "$oid": "51e3dbd465701fd8e0f69828"
 },
 "contextData":
 {
   "session_key": "08bf50c8-6ea6-41dc-99ac-5d12a6f517a3",
   "user_id": 1,
   "kp_id": 9,
   "kp_identificador": "gatewaysensores",
   "timestamp": {"$date": "2014-01-27T11:14:00Z"}
 },
 {
  "SensorTemperatura":
  {
    "identificador":"ST-TA3231-1",
    "timestamp":{"$date": "2014-01-27T11:14:00Z"},
    "medida":25.1,
    "unidad":"C",
    "geometry":
    {
      "type": "Point",
      "coordinates":[90,-10.1]
    }
  }
 }
}
```


Podemos observar que la información devuelta incluye:

- El **identificador** de esa instancia:


```
"_id": {
  "$oid": "52794144abc7d0b77e686c8b"
}
```


- **Información de contexto:** como el KP, instancia, usuario, sesión y fecha en la que se insertó.


```
"contextData": {
 "session_key": "08bf50c8-6ea6-41dc-99ac-5d12a6f517a3",
 "user_id": 1,
 "kp_id": 9,
 "kp_identificador": "gatewaysensores",
 "timestamp": {"$date": "2014-01-27T11:14:00Z"}
},
```


- **Instancia de la Ontología**

```
"SensorTemperatura": {
 "identificador":"ST-TA3231-1",
 "timestamp":{"$date": "2014-01-27T11:14:00Z"},
 "medida":25.1,
 "unidad":"C",
  "geometry":{
    "type": "Point",
    "coordinates":[110.2,1233.1]
  }
}
```

Tecnologías Implicadas

JSON

JSON es el acrónimo de JavaScript Object Notation.

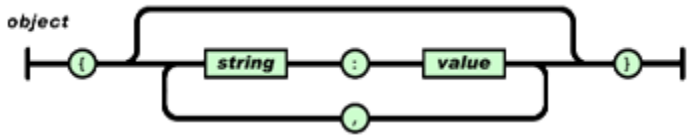
JSON es un formato ligero originalmente concebido para el intercambio de datos en Internet.

Tipos de datos JSON

- **string** : Cadena de texto
- **number**: Numérico
- **object**: Objeto
- **char**: Caracteres Unicode válidos
- **array**: Colección de valore
- **null**: Nulo
- **boolean**: Valores true o false

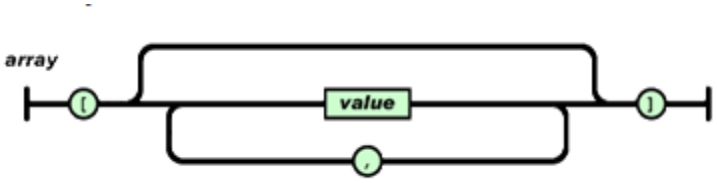
En JSON, se presentan de estas formas:

Un **objeto** es un conjunto sin ordenar de pares clave-valor. Comienza por "{" y termina con "}". Cada nombre estará seguido por ":", los pares clave-valor estarán separados por ",".



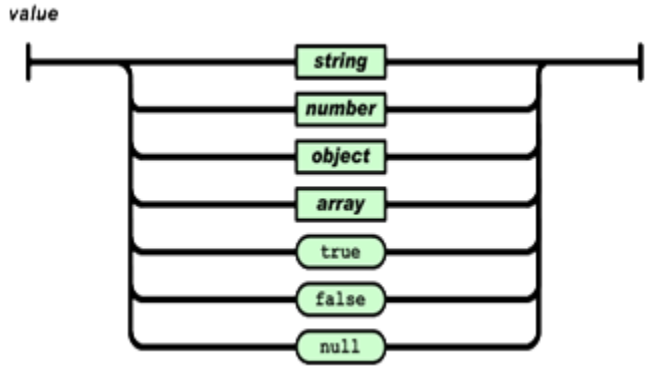

```
{
  "address" : {
    "line1" : "555 Main Street",
    "city" : "Denver",
    "stateOrProvince" : "CO",
    "zipOrPostalCode" : "80202",
    "country" : "USA"
  }
}
```

Un **array** es una colección de valores. Comienza por "[" y finaliza con "]". Los valores se separan por ",".

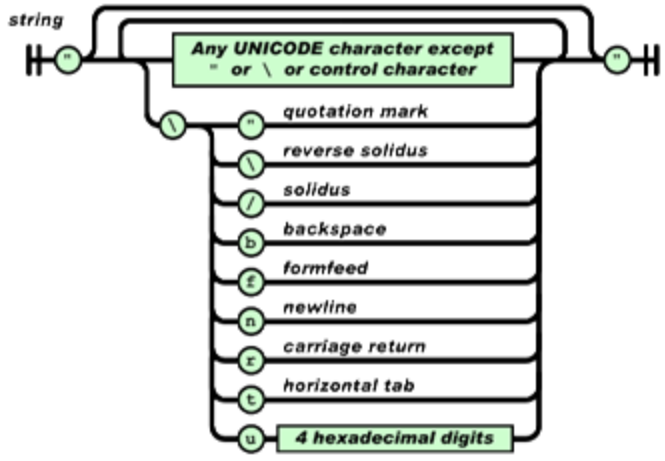



```
{
  "people" : [
    { "firstName": "John", "lastName": "Smith", "age": 35 },
    { "firstName": "Jane", "lastName": "Smith", "age": 32 }
  ]
}
```

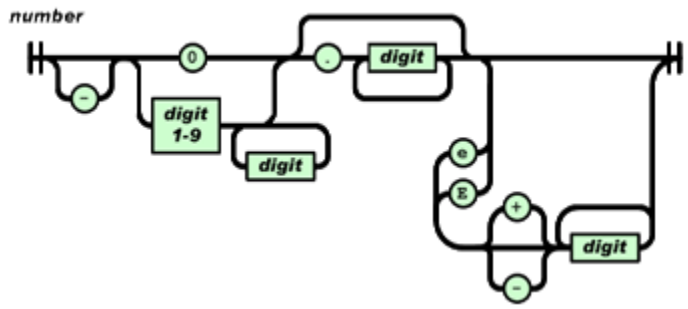
Un **valor** puede ser una cadena de caracteres con comillas doble, un número, true, false, null, un objeto o un array. Esta estructuras pueden anidarse:



Un **string** es una secuencia de cero o más caracteres Unicode, encerrados entre comillas dobles (“)



Un **número** es como un número decimal en Java.



Referencias

http://cdn.dzone.com/sites/all/files/refcardz/rc173-010d-JSON_2.pdf

Esquemas JSON (JSON-SChema)

JSON-Schema (http://json-schema.org) es un formato JSON para describir datos en JSON. Es en JSON lo que XSD a XML. Ofrece un contrato para definir los datos requeridos para una aplicación dada y la forma de interactuar con él.

Ejemplo

Para hacernos una idea veamos un ejemplo de un esquema JSON sencillo:

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "title": "Product",
  "description": "A product from Acme's catalog",
  "type":"object",
  "properties":
  {
    "id":
    {
      "description": "The unique identifier for a product",
      "type": "integer"
    },
  "name":
  {
    "description": "Name of the product",
    "type": "string"
  },
  "price":
  {
    "type": "number",
    "minimum": 0,
    "exclusiveMinimum": true
  }
 },
 "required": ["id","name","price"]
}
```

Que validaría como válidos JSONs como este:

```
{
  "id": 1,
  "name": "A green door",
  "price": 12.50,
  "tags": ["home", "green"]
}
```

Y como inválido este por no tener el atributo price:

```
{
     "id": 1,
     "name": "A green door",
     "tags": ["home", "green"]
}
```

Atributos de un esquema JSON

Podemos ver la referencia completa de la especificación JSON aquí: http://json-schema.org/latest/json-schema-core.html

Los atributos más utilizados en un esquema JON son:

- **"$schema"**: Nos permite indicar la versión del Schema JSON que queremos usar: 0.4 o 0.3, SOFIA2 se apoya en la versión 0.4 (http://json-schema.org/draft-04/schema#).
- **"title"**: indicar un título con el que identificar el esquema.
- **"description"**: Se puede utilizar este atributo para incluir una descripción de lo que va a representar el esquema JSON.
- **"type"**: Para indicar el tipo que va a representar el esquema.
- **"properties"**: Este atributo es un objeto con las definiciones de propiedades que definen los valores estáticos de una instancia de objeto. Es una lista no ordenada de propiedades. Los nombres de las propiedades se deben cumplir y el valor de las propiedades se definen a partir de un esquema, que debe cumplirse también.
- **"patternProperties"**: Este atributo es un objeto con las definiciones de propiedades que definen los valores de una instancia de objeto. Es una lista desordenada de propiedades. Los nombres de las propiedades son patrones de expresiones regulares, las instancias de las propiedades deben cumplir con el patrón definido y el valor de la propiedad con el esquema que define esa propiedad.
- **"additionalProperties"**: Permite indicar si la instancia JSON puede contener propiedades que no hayan sido definidas en el esquema. Tiene dos posibles valores (true o false), para indicar si se admite cualquier propiedad o no. Si no se añade la propiedad, se podrá incluir cualquier otra propiedad.
- **"required"**: Permite indicar todas las propiedades que son obligatorias para una instancia JSON y que como mínimo debe incluir. Las propiedades se incluirán entre corchetes y separadas por el carácter ",".

  (Este propiedad es obligatoria incluirla en el esquema).
- **"$ref"**: Define una URI de un esquema que contienen la completa representación para esa propiedad.

Veamos en este extracto de esquema un ejemplo para los atributos definidos:

```
{
    "$schema":"http://json-schema.org/draft-04/schema#",
    "title":"SensorTemperatura Schema",
    "type":"object",
    "required":["SensorTemperatura"],
```

```
    "properties":
    {
        "_id":
        {
            "type":"object",
            "$ref":"#/identificador"
        },
        "SensorTemperatura":
        {
            "type":"string",
            "$ref":"#/datos"
        }
     },
     "additionalProperties":false
}
```

En este ejemplo podemos ver que hay una propiedad que es obligatoria "**SensorTemperatura**" y que hay dos propiedades **"_id**" y "**SensorTemperatura**", que incluyen una referencia a un elemento que es el que contiene la representación completa de esa propiedad.

```
"identificador":
{
            "title":"id",
            "description":"Id insertado del SensorTemperatura",
            "type":"object",
            "properties":{
                "$oid":{
                    "type":"string"
                }
            },
            "additionalProperties":false
   },
   "datos":
   {
        "title":"datos",
        "description":"Info SensorTemperatura",
        "type":"object",
        "required":["identificador","timestamp","medida","unidad","coordenadaGps"],
        "properties":{
            "identificador":{
                "type":"string"
            },
            "timestamp":{
                    "type":"object",
                    "required":["$date"],
                    "properties":{
                        "$date":{
                            "type":"string",
                            "format":"date-time"
                        }
                    },
                    "additionalProperties":false
                    },
            "medida":{
                "type":"number"
            },
            "unidad":{
                "type":"string"},
```

```
            "geometry":{
                "$ref":"#/gps"
            }
        },
     "additionalProperties":false
    }
```

Como podemos ver tanto “identificador” como en “datos” son esquemas que definen su representación. Podemos ver también que no se admiten ningún tipo de propiedad que no sean las definidas (se ha incluido “additionalProperties”).

- **Enumerados**: Los enumerados los representaremos a como una lista entre corchetes y separados entre el carácter “,”. Los enumerados siempre son de tipo “string”. Por ejemplo si queremos definir una propiedad llamada “tipo” que sólo pueda tener uno de los dos valores “latitud” o “longitud”, quedaría del siguiente modo:

```
"tipo":
{
    "type": "string",
    "enum": ["latitud","longitud"]
}
```

Para instanciarlo, “tipo”: “latitud”*

- **“ítems”:** Define los elementos permitidos en un array, debe ser un esquema o un conjunto de esquemas.
- **“additonalItems”:** Para indicar si se admiten elementos en el array, además de los definidos en el esquema.
- **“minItems”:** Número mínimo de elementos que puede tener el array.
- **“maxItems”:** Número máximo de elementos que puede tener el array.

En el siguiente ejemplo podemos ver cómo es el esquema para un array, “coordinates”, que debe ser de tipo numérico y que sólo puede tener dos elementos. También vemos que la propiedad “type”, es un enumerado con un único valor posible “Point”.

```
"geometry":
{
    "type": "object",
    "required":["coordinates","type"],
    "properties":{
        "coordinates":{
            "type":"array",
            "items":{
                "type":"number"
            },
            "minItems":2,
            "maxItems":2
         },
        "type":{
            "type":"string",
            "enum":["Point"]
          }
    },
      "additionalProperties":false
}
```

Una instancia para este objeto sería como el siguiente:

```
"geometry":
 {
```

```
    "type": "Point",
    "coordinates":[110.2,1233.1]
}
```

Podemos encontrar más información y ejemplos en el siguiente enlace: http://json-schema.org/

Gobierno de Ontologías

La Plataforma Sofia2 ofrece todos los mecanismos necesarios para interconectar cualquier tipo de red de sensores ofreciendo diversos mecanismos para ello.

El presente documento, en aras de establecer una base común de trabajo para todas las aplicaciones y verticales, establece una serie de **recomendaciones** sobre reglas y procedimientos para la utilización de los mismos.

El objetivo es que **a priori** los desarrolladores y sistemas para explotar la información conozcan las estructuras de datos y tipos que maneja la Plataforma y **puedan hacer un uso homogéneo de los mismos independiente de la implementación concreta y particular de los desarrollos de integración**.

Se recomienda encarecidamente la lectura previa de la documentación de referencia para entender todos los conceptos manejados en el presente documento.

Plataforma obtiene, registra y gestiona toda la información de carácter sensorial (medidas, comandos, alertas, eventos, etc…) usando **ontologías json** para la representación de los datos.

Dado que los mecanismos ofrecidos por Sofia2 para la gestión de ontologías son muy flexibles y adaptables, para **garantizar que toda la información en la Plataforma tiene una estructura homogénea entre los diferentes aplicaciones y verticales**, y que permite **hacer un uso horizontal de toda la información generada**, se han establecido las reglas de gobernanza fijadas en el presente documento.

Entidades Sofia2

La Plataforma sofia2 gestiona dos tipos de entidades:

- **Assets**. Un asset es todo elemento (físico o virtual) capaz de generar o consumir información de carácter sensorial y gestionarla a través de la Plataforma SOFIA2
- **Ontologías**. Los assets generan información y dicha información se modela por medio de ontologías (JSON) en la Plataforma.

Assets

Dependiendo de su naturaleza y de cómo se conectan dichos assets a la Plataforma se establece una clasificación de los mismos en:

- **Assets del Inventario** de la Plataforma. Son Assets directamente gestionados por la Plataforma por lo cual se debe de conocer a priori toda la información sobre los mismos que permita conectarse a ellos directamente. **Todos los assets de inventario tienen que estar dados de alta en el Inventario de Assets de la Plataforma**. Se dividen a su vez en dos tipos:
  - **Managed.** Se trata de los Assets conectados a la Plataforma es decir, gestionados por los KPs. Su característica principal es que los KPs que los gestionan necesitan conocer toda la información de acceso físico a los mismos.
  - **Unmanaged.** Se trata de Assets conocidos en el inventario de la Plataforma, que generan y consumen información de la misma, pero para los cuales no es responsabilidad del módulo de interconexión su gestión final, es decir, son gestionados por otros sistemas de información que se integran en la Plataforma.
- **Assets Virtuales**. Los assets virtuales son a priori no conocidos, generan información y esta llega a la Plataforma a través de integraciones, normalmente contra otros servicios de información en la nube, como por ejemplo las redes sociales.

Ontologías

Los assets conectados generan y consumen información sensorial en forma de ontologías json.

A la hora de crear una ontología se pueden usar los catálogos (plantillas) siguientes:

- **Feed** (medidas). Se trata de la información de medida emitida por los assets conectados. Pueden ser medidas instantáneas, agregaciones por períodos de tiempo, cálculos, etc... Un feed puede recoger varias medidas simultáneamente y **todos los feeds deben de estar georreferenciados** (puntos, líneas o áreas) pudiendo ser éstos móviles.
- **Command** (comandos u órdenes). Se trata de las **órdenes enviadas a los assets** con capacidades de actuación. Dichas órdenes son asíncronas (no existe respuesta inmediata) por lo que existen dos tipos de comandos:
  - **Comandos request.** Las instrucciones enviadas a un Asset.
  - **Comandos response**. Las respuestas o ACKs enviadas por los Assets a dichos comandos. Las respuestas a los comandos únicamente indican la confirmación de la recepción de dicha instrucción.
- **Alert** (alertas). Las alertas son mensajes de notificación de situaciones (alertas, incidencias, mensajes, notificaciones, etc...) y tienen la particularidad de que **su estado así como su nivel de criticidad cambia a lo largo del tiempo** desde un mensaje inicial que genera la alerta hasta el cierre de la misma. Las alertas no se conocen a priori, es decir, se generan en el momento en el que suceden.
- **Schedule** (eventos). Los eventos reflejan situaciones conocidas en el tiempo y en el espacio, es decir, ocurren durante un periodo de tiempo en un lugar concreto y, normalmente, se conocen a priori.
- **Audit** (log). Los mensajes de auditoría o log son internos a la Plataforma y se utilizan para hacer explícitas y dejar gestionadas situaciones concretas particulares de los Assets. Por ejemplo, el encendido de una farola, el mal funcionamiento de un sensor, etc...
- **KPI** (indicador). Los indicadores son un tipo especial de medida y su característica principal es que no son generados por un único Asset. Se trata de cálculos realizados sobre múltiples datos, series históricas e incluso capas de datos espaciales que se almacenan como medidas en Plataforma.

Reglas especiales a tener en cuenta:

- Cada vez que se conecta un nuevo conjunto de assets (del mismo tipo) a la Plataforma deben de darse de alta al menos las ontologías **feed** y **audit** y si corresponde **command** para el mismo.
- Inmediatamente a la recepción de un comando y su correcta ejecución el Asset (el KP que lo gestiona) debe de generar una nueva medida para reflejar en la Plataforma su nuevo estado.

Nomenclatura

A continuación se establecen las reglas básicas de nomenclatura:

- Se utilizarán nombres en inglés y se seguirá el estándar Java (aka "camel").
- Para la definición de plantillas:
  - Nombres cortos, autoexplicativos. Primera letra en mayúsculas. Las primeras letras identifican el tipo de ontología:
    - Feed: **Feed**
    - Command: **Cmd**
    - Alert: **Alrt**
    - Schedule: **Schdl**
    - Audit: **Adt**
    - KPI: **Kpi**
- Para la definición de **ontologías**:
  - Primera letra en minúscula, empiezan siempre por el tipo de ontología. Ejemplo para una Espira: feedEspira, cmdEspira, adtEspira ...
- Para la definición de **atributos** de las ontologías:
  - Primera letra en minúsculas, sin espacios, sin caracteres especiales.
- Para la definición de **constantes** utilizadas como valores posibles para los atributos de las ontologías: todas las letras en mayúsculas. Por ejemplo: MOBILE, FIXED, VIRTUAL.

Tipado y Formatos

Para la definición de ontologías se utilizarán cadenas de texto UTF-8 siguiendo el esquema json establecido por la correspondiente plantilla (actualmente siguiendo JSON Schema 0.4 http://json-schema.org/draft-04/schema#).

A continuación se establecen las reglas de tipado y formato para los diferentes tipos soportados:

- **UUIDs**:
  - Cadena de texto. Standard Universally Unique Identifier.
- **Números enteros**:
  - Entero Largo de 64 bits
  - Ejemplo: {'contador' : 10}
- **Números flotantes**:
  - Notación simple. Decimal con punto. 64 bits
  - Ejemplo: {'valor' : 10.5}
- **Cadenas de texto**:
  - Cadena de texto. UTF-8. Caracteres especiales escapados
  - Ejemplo: {'comment' : 'next station'}
- **URLs y URIs**:
  - Cadena de texto. Codificadas siguiendo estándar RFC-1738

  - Ejemplo: {'url' : 'http%3A%2F%2Fwww.coruna.es%2Fmedioambiente%2F'}

- **Timestamps**:
  - Fecha. Cadena de texto siguiendo formato ISO-8601. RFC 3339
  - Objeto conteniendo atributo "$date"
  - Ejemplo: {"timestamp":{"$date":"2014-01-27T11:14:00Z"}}
- **Fechas e intervalos de fechas**:
  - Cadena de texto siguiendo formato ISO-8601.RFC 3339
  - Objeto con atributo "$date"
  - Ejemplo de fecha: {"created":{"$date":"2014-01-27T11:14:00Z"}}
  - Ejemplo de intervalo entre dos fechas: {"period":{"$date" : "2010-07-02T11:44:09Z/2010-07-02T11:47:00Z"}}
- **Direcciones**:
  - Notificación simplificada para facilitar las tareas de integración:

```
{"address":
  {
    "location": "cadena de texto",
    "number": "cadena de texto"
  }
}
```

- **Unidades de medidas**
  - Cadena de texto string siguendo notación **JScience library** (http://jscience.org/api/javax/measure/unit/SI.html)(http://jscience.org/api/javax/measure/unit/NonSI.html)
  - Ejemplo {'unit': 'A'} # Amperios
- **Coordenadas geográficas**:
  - Siguen la definición **OGC GeoJson**. Esquema de coordenadas **WGS84**. No hay coordenada Z. Orden [longitud, latitud]
  - **Puntos**:
    - GeoJson Point
    - Ejemplo:

```
 {"geometry":
  {
   "type": "Point",
   "coordinates": [-8.410161625142807, 43.360463863501934]
  }
}
```

- **Líneas**:
  - GeoJson LineString
  - Ejemplo:

```
 {"geometry":
  {
   "type": "LineString",
   "coordinates":[
     [-8.410161625142807, 43.360463863501934],
     [-8.410161625142807, 43.360463863501978]
   ]
 }
}
```

- **Áreas**:
  - GeoJson Polygon
  - Ejemplo:

```
  {"geometry":
   {
    "type": "Polygon",
    "coordinates":[
       [[-8.410161625142807, 43.360463863501934],
        [-8.410161625142807, 43.360463863501978],
        [-8.41016162514290, 43.360463863501978],
        [-8.410161625142807, 43.360463863501934]
       ]
    ]
  }
}
```

[NOTA]: Para cerrar el polígono el primer y el último valor de cada anillo deben de ser idénticos.

[NOTA]: Un polígono puede tener 2 anillos (el exterior y el interior).

Plantillas Predefinidas

Feeds (Medidas)

Para la definición de la plantilla de medidas se utiliza una simplificación del estándar de datos AMON (**http://amee.github.io/AMON/**):

```
{"Feed":
 {
  "asset":
  {
   "assetId" : string, (required)
    "assetType": string, (required)
    "assetSource": string, (required)
    "assetName": string (optional)
  },

 "type": string, (required) [FIXED, MOBILE, VIRTUAL]
 "timestamp": (required)
 {
  "$date": "RFC 3339 DATETIME"
 },
 "attribs": (optional)
 [
```

```
  { "name": "value" }
 ],
 "geometry": geojson [Point, LineString, Polygon], (optional)
 "measures": (required)
 {
  "timestamp" : (required)
   {
    "$date": "RFC 3339 DATETIME"
   },
  "type" : string, (required) [INSTANT, CUMULATIVE, PULSE]
  "period" : number, (optional)
  "periodUnit": string, (optional) [m, s, h, d]
  "values" : (required)
    [
     {
      "name": string, (optional)
      "desc": string, (optional)
      "unit": string, (required)
      "measure": string, (required)
      "method": string,(required)
      "modifications": (optional)
      [
       {
        "oldMeasure": string, (required)
        "changeTimestamp": (required)
        {
          "$date": "RFC 3339 DATETIME"
        }
        "changeDesc": string, (optional)
       }
     ]
    }
   ]
  }
 }
}
```

El objeto **asset** hace referencia al activo que emite la medida:

- **assetId**: identificador del activo en el sistema de referencia que lo gestiona (establecido en el campo assetSource).
- **assetType**: tipo de asset (farola, sensor de humedad, etc. . .)
- **assetSource**: sistema de información que gestiona el activo.
- **assetName**: atributo opcional para asociar un nombre al activo si se considera necesario.

El tipo de sensor (**type**) hace referencia a su naturaleza, los tipos validos son:

- **FIXED**. Sensores a priori conocidos (gestionados por un inventario conocido) posicionados geograficamente en una posición fija conocida.
- **MOBILE**. Sensores a priori conocidos que se mueven y su posición se actualiza en cada medida.
- **VIRTUAL**. Sensores a priori no conocidos (por ejemplo redes sociales).

El **timestamp** referencia la **fecha y hora de captura del feed**.

NOTA1: no confundir con el timestamp que se genera automáticamente al enviar la ontología al módulo de interconexión

NOTA2: no confundir con el timestamp de las medidas que referencia el momento de recogida de las mismas).

El objeto de **atributos** (attribs) tiene por objeto recoger una lista arbitraria de atributos modelados en forma clave:valor. Su utilidad puede ir desde recoger claves secundarias hasta almacenar cualquier atributo adicional necesario.

El objeto **geometry** recoge la posicion (punto, linea o poligono) a la que referencia el feed. **En todo caso, siempre que la posicion del feed sea conocida debe de figurar en el feed** independientemente de que la misma se encuentre dada de alta en el inventario. En caso de no conocerse la posición el atributo no debe de figurar en el feed.

El objeto **measures** hace referencia a las características comunes de referencia de todas las medidas capturadas y lista todas las medidas realizadas:

- **timestamp**: fecha de referencia de realización de las medidas
- **type**: tipo de medida realizada: medidad instantánea, acumulado, pulso
- **period**: si procede, período de tiempo utilizado para el cálculo de las medidas.
- **periodUnit**: unidad de tiempo ('s', 'm', 'h', 'd') utilizada para definir el período de tiempo.
- **values**: lista de medidas realizadas
  - **name**: si procede, nombre representativo de la medida
  - **desc**: si procede, descripción de la medida realizada
  - **unit**: unidad de medida
  - **measure**: valor de la medida en su versión más actualizada. Es decir, el atributo measure contendrá siempre la medida válida. En caso de realizarse modificaciones de la medida los valores históricos serán almacenados en la lista del atributo modifications.
  - **method**: método utilizado para obtener la medida (media, min, max, etc...)
  - **modifications**: lista de modificaciones realizadas sobre la medida originalmente capturada

Commands (Comandos)

Para la definición de los comandos se utiliza:

```
{"Command":

 {
   "commandId": string, (required)
   "asset":
    {
     "assetId" : string, (required)
     "assetType": string, (required)
     "assetSource": string, (required)
     "assetName": string (optional)
    },
   "timestamp": (required)
    {
      "$date": "ISO 3339 DATETIME"
    },
   "desc": string, (optional),
   "type": string, (required) [REQUEST, RESPONSE],
   "command":
```

```
      {
        "type": string, (required) [SWITCH, DIM, SET, EXECUTE, SEND]
        "value1": string, (optional)
        "value2": string, (optional)
        "value3": string, (optional)
        "msg": string (optional)
      }
    "rule":
     {
       "type": string, (required) [ASAP, DATE]
       "date": (optional)
       {
         "$date": "ISO 3339 DATETIME"
       }
     }
   }
}
```

Alertas

Para la definición de las alertas se utiliza una simplificación del estándar **CAP 1.2 (http://docs.oasis-open.org/emergency/cap/v1.2/CAP-v1.2-os.html)**

```
  {"Alert":
   {
    "id" : {
    "alertId": string, (required)
    "alertSource": string (required)
   },
   "timestamp":
    {
     "$date": "ISO 3339", (required)
    },
   "asset":
    {
      "assetId" : string, (required)
      "assetType": string, (required)
      "assetSource": string, (required)
      "assetName": string (optional)
    },
  "alert":
   {
     "sourceAlertId": identifier,
     "subject": string required,
     "description": string optional,
     "source": string required,
     "type": [ALARM, WARNING, MESSAGE, NOTIFICATION, INFO],
     "status": [OPEN, CLOSED, UNKNOWN],
     "affectedLocations": (optional)
     [
      {
        "desc": string, (optional)
        "geometry": geojson,(optional) [Point, Line, Polygon],
        "locationUri": string, (optional)
      }
```

```
    ]
  }
 "info":
  {
    "action": [CREATE, CLOSE, UPDATE, ACK,FOLLOW, SCALATION, REMINDER, CANCEL],
    "sender": string, (required)
    "contact": string, (optional)
    "description": string, (optional)
    "parameters": string, (optional)
    "urgency": [EXPECTED, FUTURE, INMEDIATE, PAST, UNKNOWN],
    "severity": [EXTREME, MINOR, MODERATE, SEVERE, UNKNOWN],
    "certainty": [LIKELY, OBSERVED, POSSIBLE, UNLIKELY, UNKNOWN],
    "resources": optional
     [
      {
       "name": string, (required)
       "description": string, (optional)
       "uri": string, (required)
       "mimeType": string (optional)
      }
     ]
  }
}
```

}

Eventos

La definición de eventos sigue el siguiente esquema.

```
  {"Event":
   {
    "id":
     {
       "eventId": string, (required)
       "eventSource": string (required)
     },
    "timestamp":
     {
       "$date": "ISO 3339", (required)
     },
    "asset":
     {
       "assetId" : string, (required)
       "assetType": string, (required)
       "assetSource": string, (required)
       "assetName": string (optional)
     },
    "eventInfo":
     {
       "subject": string, (required)
       "description": string, (optional)
       "type": string,[INFO, PROGRAM, EVENT]
       "affectedLocations": (optional)
         [
```

```
          {
            "desc": string, (required)
            "geometry": geojson optional [Point, Line, Polygon],
            "locationURI": string optional
          }
        ],
      "resources": (optional)
       [
        {
            "name": string, (required)
            "description": string, (optional)
            "uri": string, (required)
            "mimeType": string (optional)
        }
       ]
      }
    "eventRule":
     {
       "type": [SINGLE, PERIOD, RULE],
        "period": (required)
        {
          $date: "RFC 3339 INTERVAL"
        },
       "repeatEach": entero, (opcional)
       "repeatUnit": string (opcional) [s, m, h, d, w, m]
     }
 }
}
```

Audit

La definición de mensajes de auditoria sigue el siguiente esquema.

```
{"Adt":
 {
  "id" : {
  "auditId": string, (required)
  "auditSource": string (required)
 },
 "timestamp": (required)
 {
   "$date": "ISO 3339 DATETIME"
 },
 "asset":
  {
   "assetId" : string, (required)
   "assetType": string, (required)
   "assetProvider": string, (required)
   "assetName": string (optional)
  },
 "message":
  {
   "source": string required,
   "sender": string optional,
   "subject": string required,
```

```
    "body": string optional,
    "level": [INFO, WARNING, ERROR, DEBUG]
   }
  }
}
```

CHAPTER 24

Documentación para Desarrolladores.

Se recomienda previamente leer la documentación Básica para adquirir los conceptos principales.

Si quieres desarrollar tu APP para interactuar con la plataforma Sofia2 las siguientes Guías para desarrolladores te serán de gran utilidad.

Primeros Pasos con Sofia2

Esta guía pretende introducir la plataforma SOFIA2, de modo que de una forma muy rápida un usuario pueda estar comunicando con la plataforma enviando y recibiendo datos a través de un KP.

Para eso se parte de:

- **SOFIA2 On Cloud**: es una versión de la Plataforma pública pensada para hacer pruebas de integración en la Plataforma. Es la instancia del SIB sobre la que los KPs ofrecidos de Ejemplo enviarán y recibirán información.
- El **SDK Sofia2** que nos permite desarrollar APPs Sofia2 de forma sencilla. No es imprescindible pero simplifica su desarrollo.
- El **API Java** que además contiene una APP Sofia2 de ejemplo (KP) que envía datos simulados de Temperatura y Humedad.
- El **API Javascript** que contiene una APP Sofia2 HTML de ejemplo que representa visualmente los datos introducidos por el KP Java.

Descarga e Instalación del SDK de Sofia2

Para descargar el SDK de Sofia2 iré al menú de Desarrolladores de Sofia2: http://sofia2.com/desarrollador.html

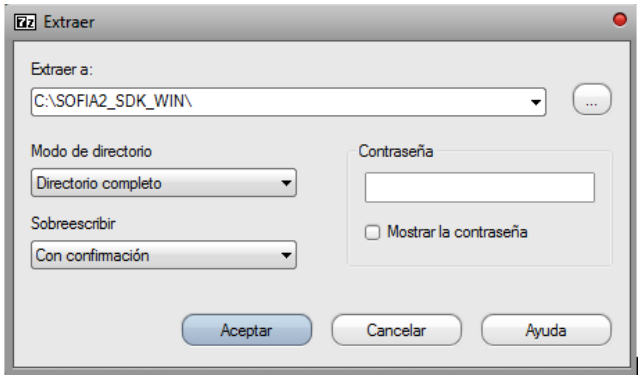
En la sección **DESCARGAS** accederé al SDK para mi Sistema Operativo, para el ejemplo usare el SDK Windows:

El SDK de Windows se suministra en formato ZIP y contiene todo lo necesario para desarrollar APPS Sofia2 sin necesidad de instalar nada en la máquina, esto es:

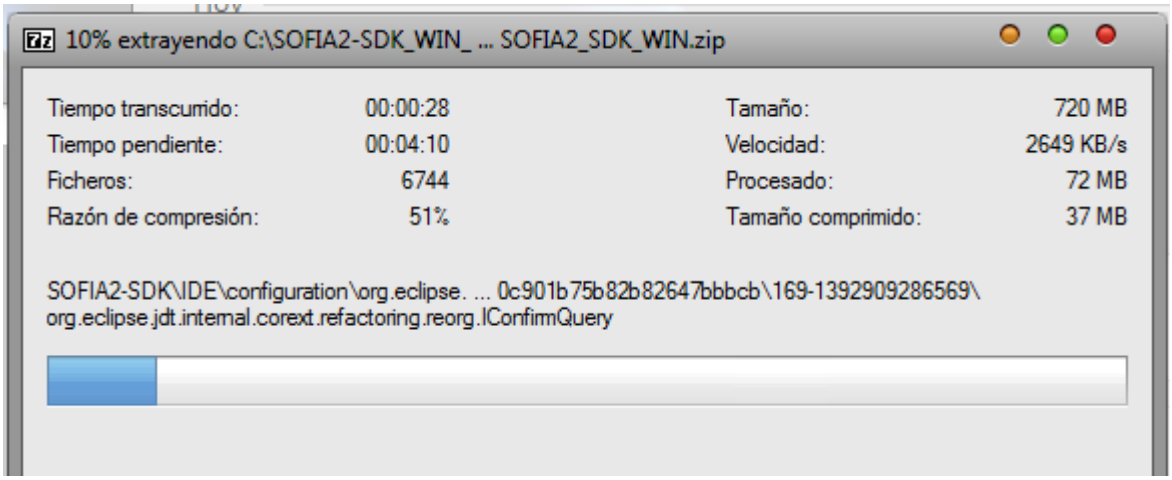
- IDE Eclipse configurado
- JVM Java 7
- Maven 3
- Scripts ejecutables

Los pasos para instalarlo son:

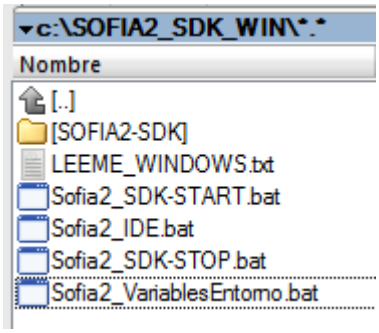
1. Descargar SDK Sofia2 (Windows) en http://sofia2.org/sdk/SOFIA2_SDK_WIN.zip
2. Una vez descargado el ZIP lo descomprimo en una carpeta, nosotros usaremos **c:\SOFIA2_SDK_WIN**



Tras descomprimirse:



Se habrá generado la carpeta con este contenido:



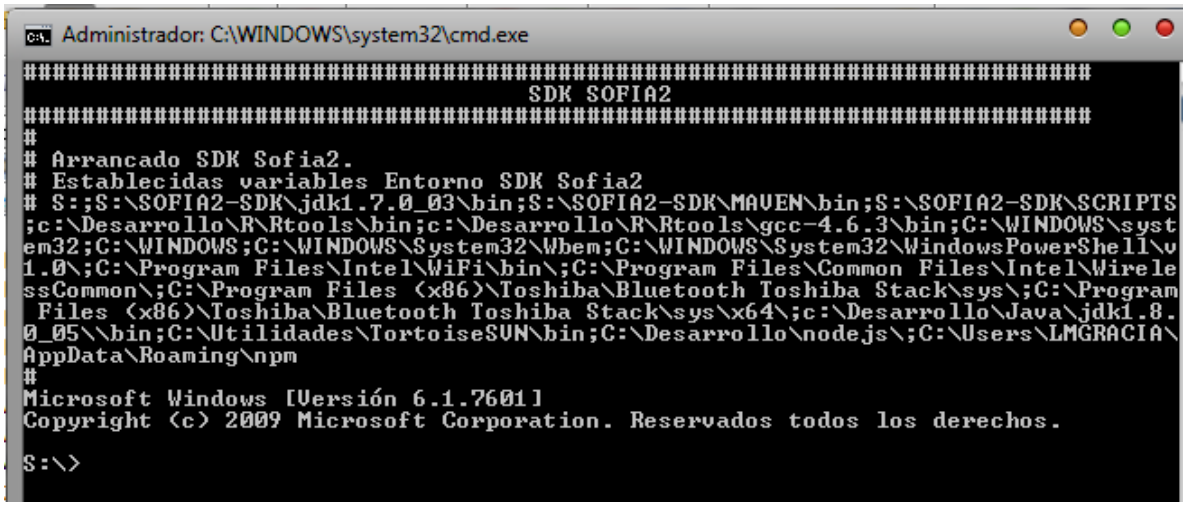
El fichero LEEME_WINDOWS.txt describe los pasos a seguir para configurar y lanzar el SDK y ejecutar los ejemplos.

3. Dentro del directorio SOFIA2-SDK encontramos todos los componentes que componen el SDK:

- En los directorios **MAVEN** y **M2_REPRO** se encuentran la Herramienta Maven las dependeencias manejadas por esta para el desarrollo sobre la Plataforma Sofia.
- En el directorio **API** encontramos las API Java y Javascript para el desarrollo de KP.
- En el directorio **WORKSPACE** encontramos los directorios de configuración de los IDE.

Arranque del SDK de SOFIA2

Una vez descomprimido voy a la carpeta (c:\SOFIA2_SDK_WIN\) y ejecuto el script **Sofia2_SDK-START.bat.** Esto debe abrir una línea de comandos sobre **S:** con esta información:



Elementos que componen el SDK de SOFIA2

La Plataforma SOFIA se compone por estos elementos:

- **IDE**
- **Consola de Arquitectura Técnica iTR de Indra**
- **Repositorio Local Maven3 con todas las dependencias**

IDE

Es el Entrono de Desarrollo de la Plataforma.

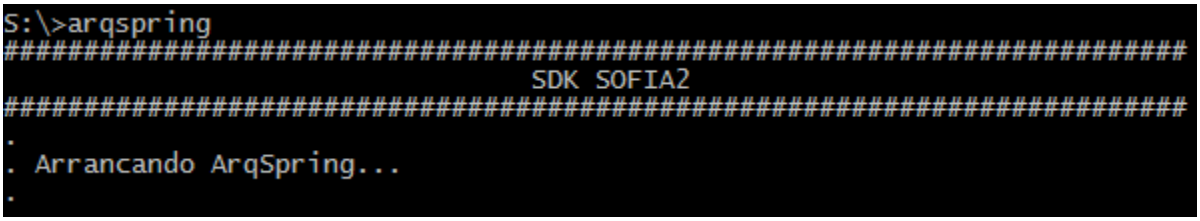
SOFIA2 distribuye 2 IDE:

- IDE para el desarrollo sobre Java y Javascript basado en Eclipse
- IDE para el desarrollo sobre Arduino.

Consola de Arquitectura

Ofrece herramientas de productividad para crear proyectos SOFIA2 de forma sencilla.

Ejecutando en la Consola el comando **S:\>arqspring**



Se abrirá la **consola de Arquitectura iTR**

Donde dispondremos de los comandos **ArqSpring>sofia create(Gateway, KP, Plugin)**

```
Welcome to Sofia2. For assistance press TAB or type "hint" then hit ENTER.
sofia2> sofia2 c

sofia2 configurarKp      sofia2 crearAppModelo    sofia2 crearKp
```

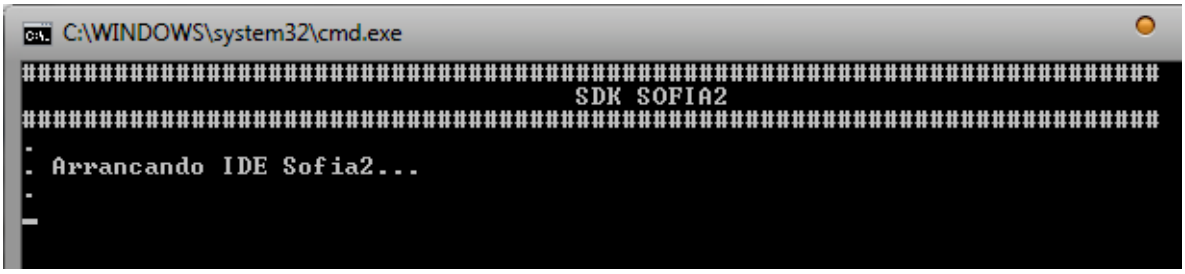
Que creará un esqueleto básico de un proyecto java para desarrollar uno de estos componentes.

Abrir IDE Sofia2

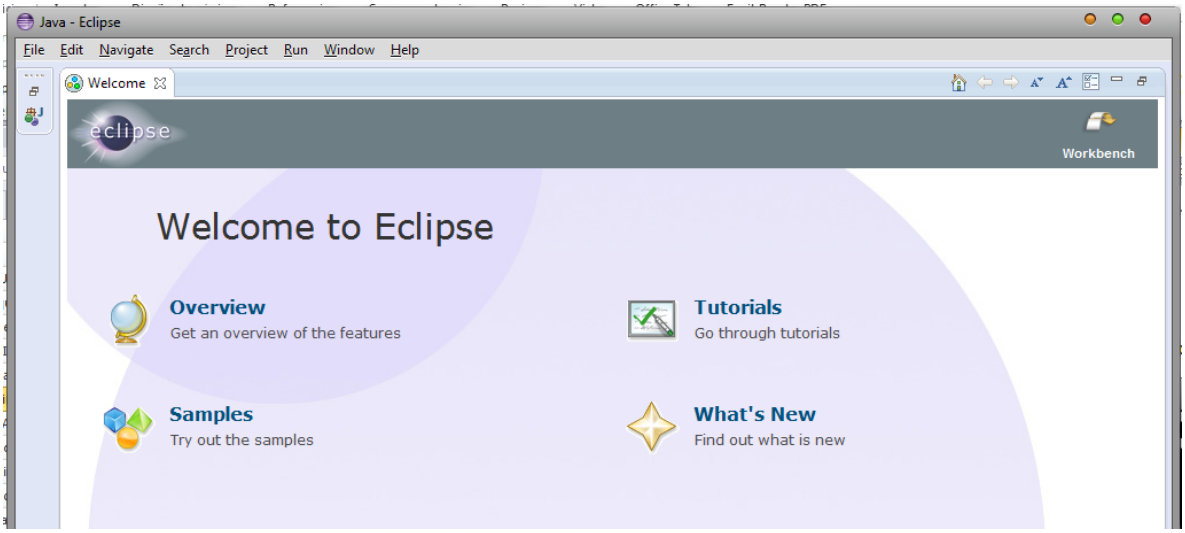
Una vez instalado y arrancado el SDK ya puedo lanzar el IDE Eclipse incluido.

Para eso:

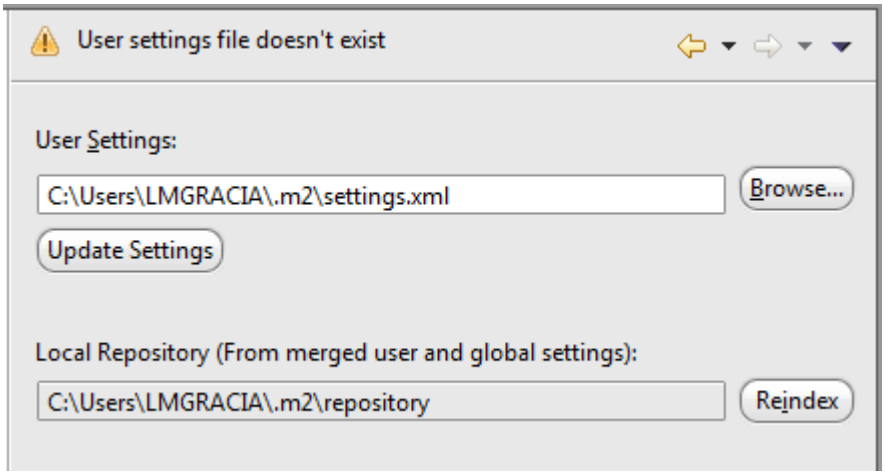
1. Desde la unidad S:\ ejecuto **S:\Sofia2_IDE.bat,** esto lanzará una ventana de comandos:



Y finalmente se abrirá un Eclipse.



1. Ahora voy a configurar el repositorio de Maven de Eclipse. Para eso voy a **Window>Preferences>Maven>User Settings**



y en User Settings selecciono: **s:\SOFIA2-SDK\MAVEN\conf\settings.xml** y selecciono **Update Settings.** Luego pincho OK.

Lanzar ejemplo de API Java Sofia2

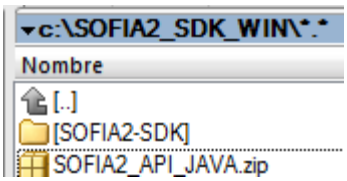
Ahora vamos a cargar en el IDE un ejemplo Java incluido en el API Java Sofia2.

Para eso:

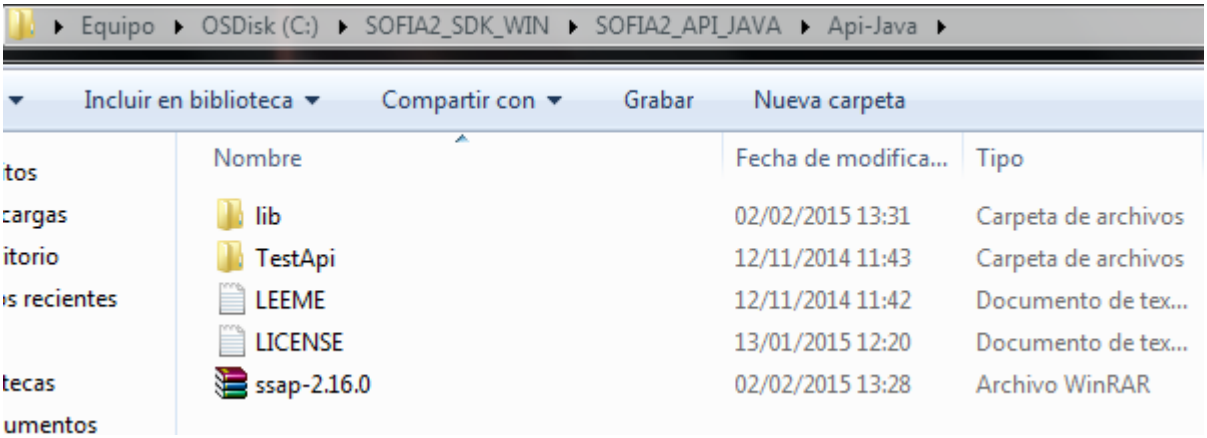
1. Desde el menú de Desarrollador de la web de Sofia2 http://sofia2.com/desarrollador.html en la sección Descargas selecciono el API Java:

   http://sofia2.org/apis/SOFIA2_API_JAVA/SOFIA2_API_JAVA.zip

1. Descargo el API Java en el directorio donde previamente instalé el SDK (en nuestro caso **c:\SOFIA2_SDK_WIN\)**

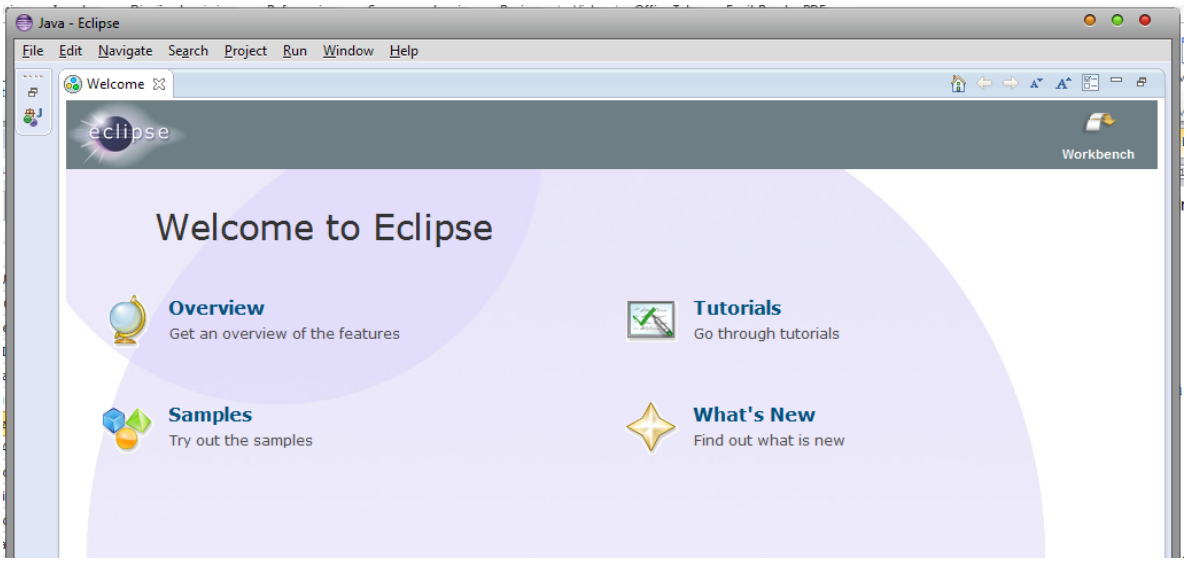


Y lo descomprimo directamente en esa carpeta. Tendré al final una carpeta c:\SOFIA2_SDK_WIN\SOFIA2_API_JAVA\Api-Java\ :

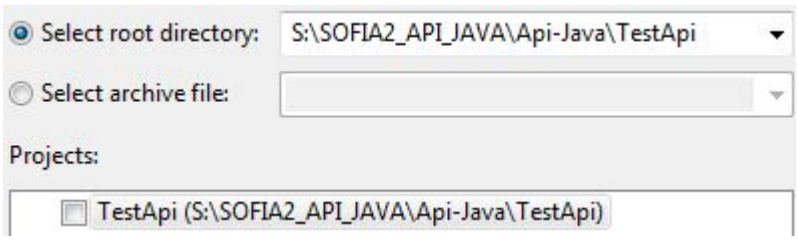


1. Si no lo tenía ya lanzado, lanzo el IDE que contiene el SDK de Sofia2 con el comando **s:\Sofia2_IDE.bat**.

   Al cabo de un tiempo debe lanzarse un Eclipse.

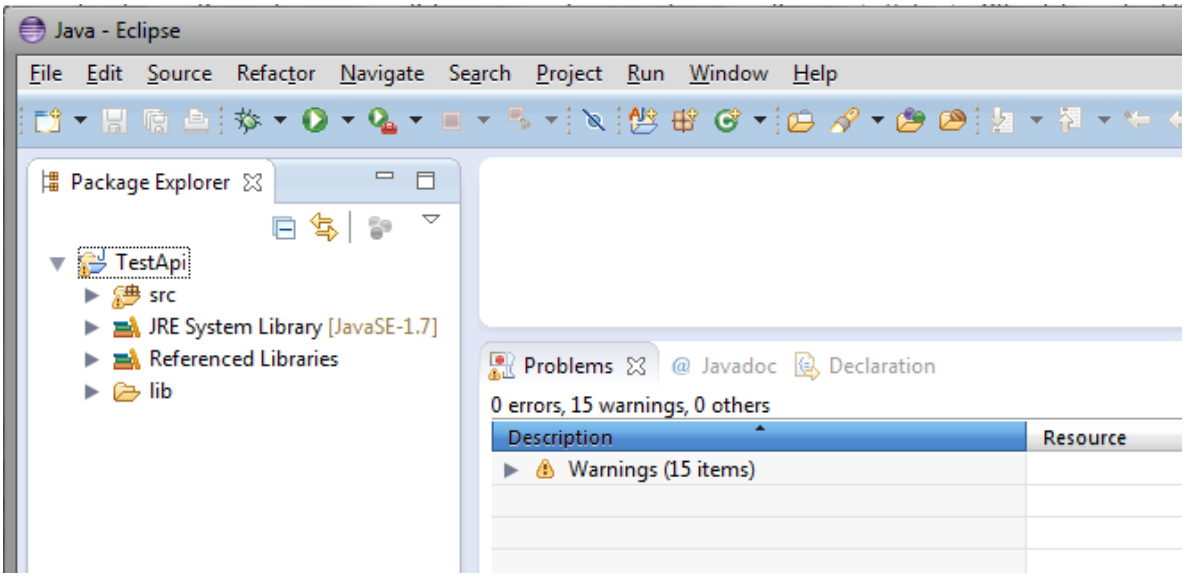


1. Lo siguiente es cargar en el IDE el ejemplo más básico incluido en el API Java. Para eso selecciono en Eclipse **File>Import>General>Existing Projects into Workspace>**" "**s:\SOFIA2_API_JAVA\Api-Java\TestApi**"

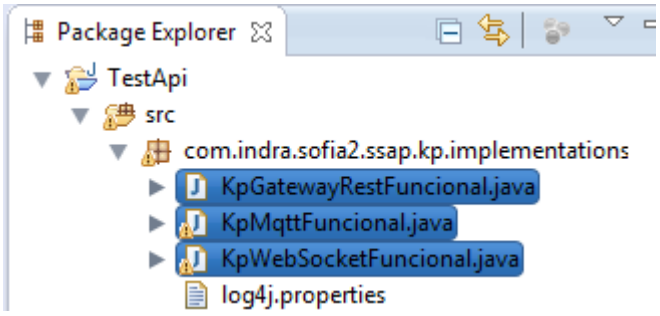


Selecciono el proyecto y pulso Finish.

Veré el proyecto cargado y compilado sin problemas:



1. En el proyecto tengo 3 clases:

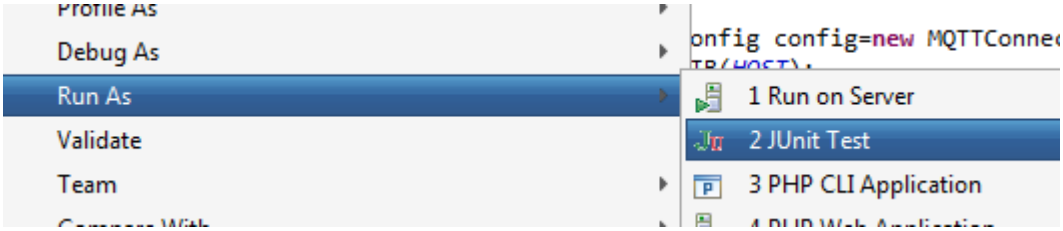


KpGatewayRestFuncional conecta vía REST con la instancia de Sofia2 en la nube.

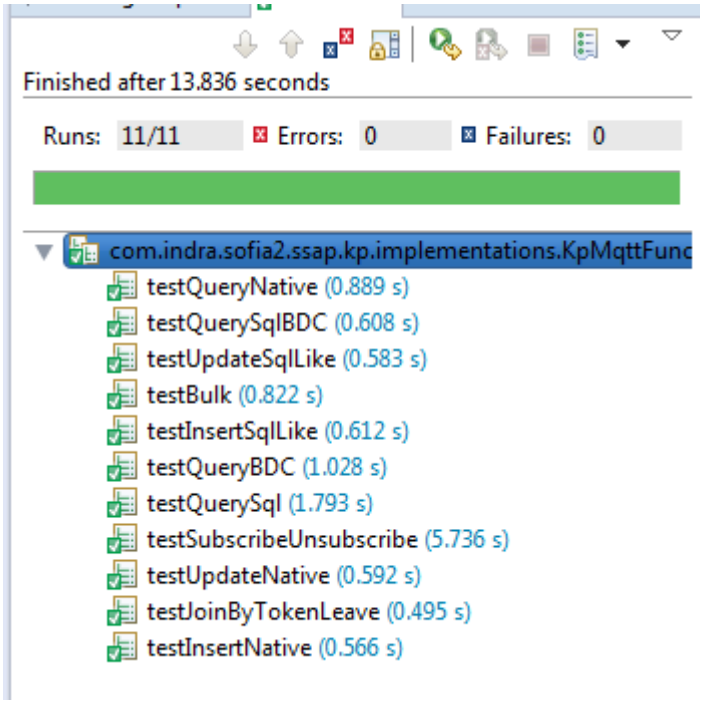
KpMqttFuncional conecta vía MQTT con la instancia de Sofia2 en la nube.

KpWebSocketFuncional conecta vía MQTT con la instancia de Sofia2 en la nube.

1. Estas 3 clases están implementadas como Tests JUnit y tienen configuradas los Tokens necesarios para insertar en la instancia de Sofia2 en la nube. Para probarlas seleccionaré botón derecho sobre cualquiera de ellas y **Run As>JUnit Test**



Al ejecutarlas obtendré:



Revisando el contenido de las clases puedo ver que se están lanzando contra el SIB de Sofia2 las siguientes operaciones (las imágenes son de la clase de MQTT):

- JOIN para autenticar
- Insert en modo nativo (insertando la ontología):

```
//Genera un mensaje de INSERT
SSAPMessage msgInsert=SSAPMessageGenerator.getInstance().generateInsertMessage(sessionKey, ONTOLOGY_NAME, ONTOLOGY_INSTANCE);
log.info("Envia mensaje INSERT al SIB: "+msgInsert.toJson());
SSAPMessage responseInsert=kp.send(msgInsert);
```

Con

```
private final static String ONTOLOGY_INSTANCE =
        "{ \"Sensor\": { \"geometry\": { \"coordinates\": [ 40.512967, -3.67495 ], \"type\": \"Point\" },"
        + " \"assetId\": \"S_Temperatura_00066\", \"measure\": 25, \"timestamp\": { \"$date\": \"2014-04-29T08:24:54.005Z\"}}}";
```

- Insert a través de SQL

```
private final static String ONTOLOGY_INSERT_SQLLIKE =
        "insert into TestSensorTemperatura(geometry, assetId, measure, timestamp) "
        + "values (\"{ 'coordinates': [ 40.512967, -3.67495 ], 'type': 'Point' }\", "
        + "\"S_Temperatura_00066\", 15, \"{ '$date': '2014-04-29T08:24:54.005Z'}\")";
```

- Update nativo y SQL
- Query en SQL, Query a la BDC (Assets)

Cómo Desarrollar una APP SOFIA2

En el punto anterior hemos descargado el SDK y las APIS y hemos ejecutado los ejemplos básicos que vienen incluidos.

En este punto explicaremos los pasos necesarios para desarrollar una **APP Sofia2** sobre la instancia de Sofia2 disponibilizada en http://sofia2.com.

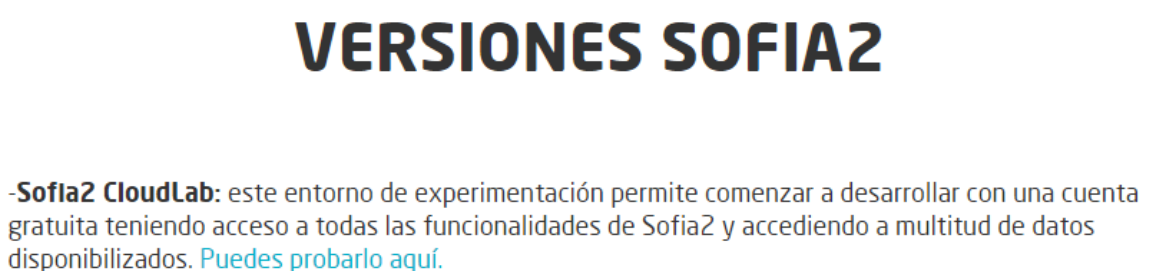
Esta instancia de Sofia2: **Sofia2 CloudLab** ofrece un entorno de experimentación que permite comenzar a desarrollar con la plataforma con una cuenta gratuita teniendo acceso a todas las funcionalidades de Sofia2 y accediendo a multitud de datos disponibilizados.

El proceso para desarrollar una APP Sofia2 consiste básicamente en:

- Darse de alta en la instancia Sofia2 CloudLab
- Acceder a la Consola Web de configuración de Sofia2 CloudLab
  - Búsqueda y suscripción a ontologías
  - Creación de APP (KP y Token)
  - Consultar información de estas ontologías desde Consola acceso BDTR
- Desarrollo de APP Sofia2 de consulta sobre las ontologías en Javascript y Java
- Desarrollo de APP Sofia2 que inserta y en Java que publica medidas de Temperatura y Humedad

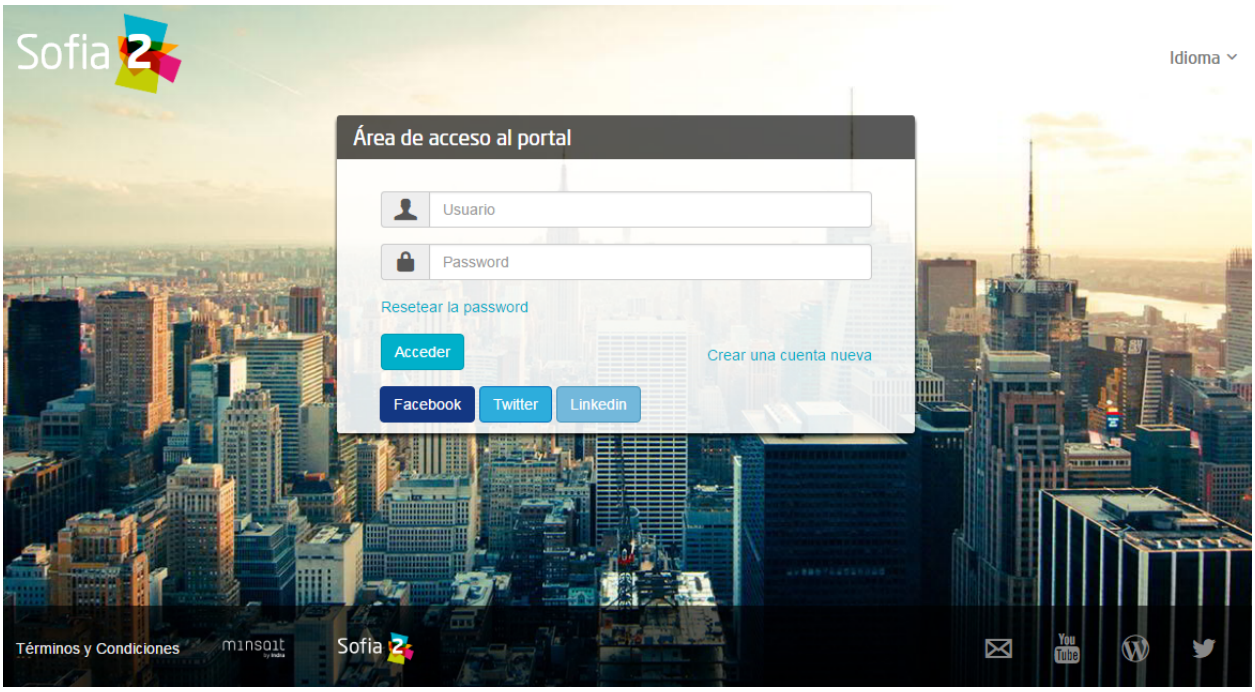
Darse de alta en la instancia Sofia2 CloudLab

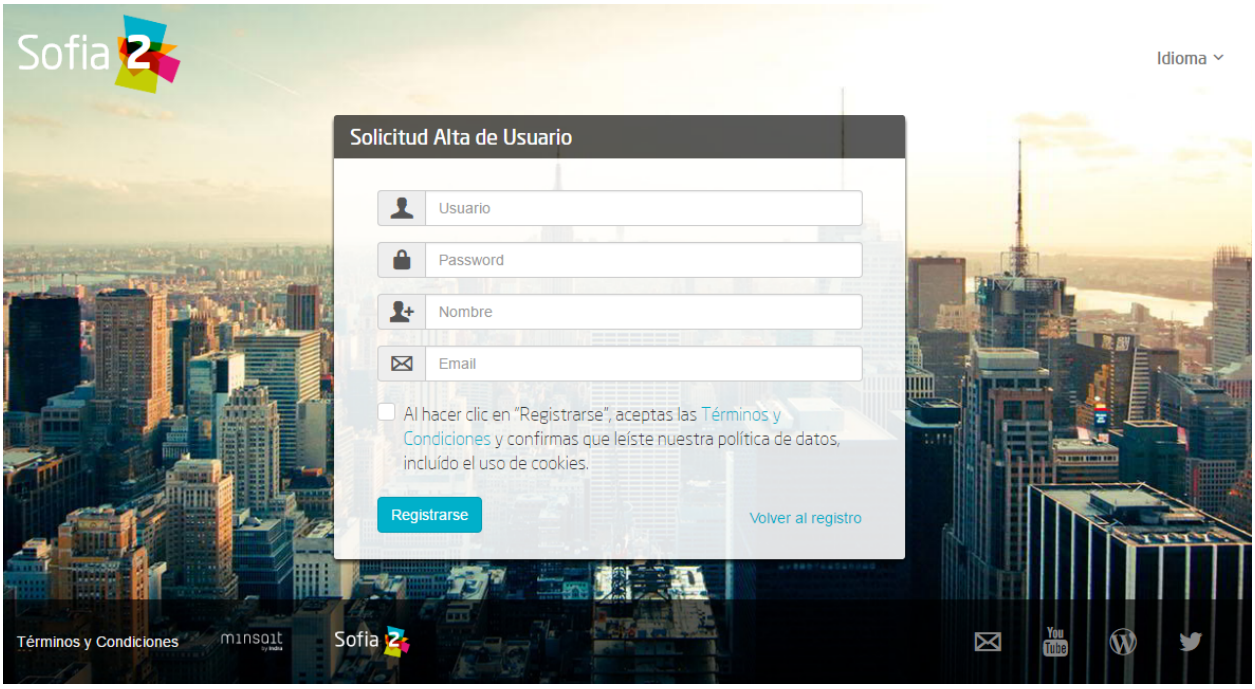
En la sección Productos de la Web de Sofia2 (http://sofia2.com/sofia2incloud.html) se puede acceder a **Sofia2 CloudLab:**



También se puede acceder vía: http://sofia2.com/console/login

En esta URL podemos registrarnos pulsando sobre el link **Crear una Nueva Cuenta**:





A la hora de crear el usuario es importante seguir las reglas que se indican, entre ellas que la password debe contener letras, números y letras mayúsculas y minúsculas.

Una vez que lo hagamos se nos asignará el rol **USUARIO**, **este rol permite consumir información pública de la plataforma, pero no volcar información en esta.**

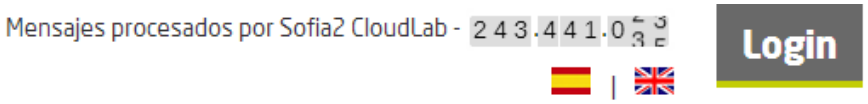
Con este rol puedo crear APPs Sofia2 que consuman (consulten, se suscriban) a las ontologías que otros usuarios hayan definido como públicas.

NOTA: Más tarde veremos como solicitar el rol **COLABORADOR**, que permite **volcar información en la**

plataforma.

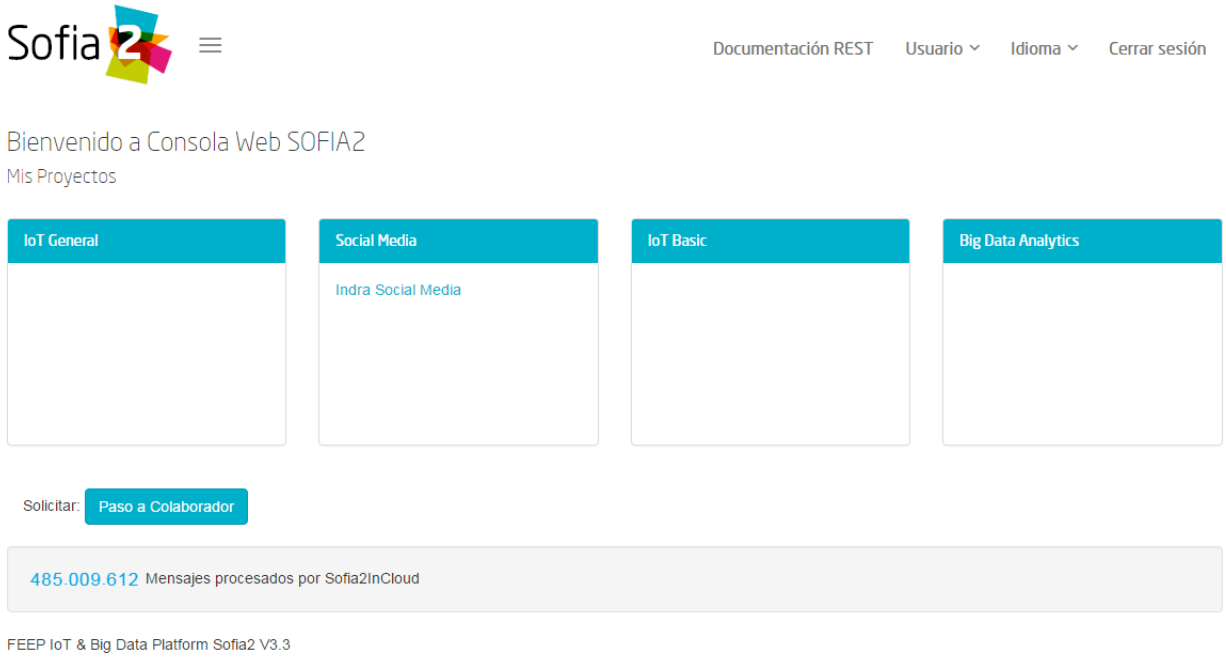
Acceder a la Consola Web de Sofia2 CloudLab

Una vez que tengo mi usuario de Sofia2 CloudLab desde la página principal de Sofia2 puedo hacer Login:



O bien a través de este link: http://sofia2.com/console/login

Nos logaremos con el usuario creado en el punto anterior, la pantalla principal tiene este aspecto:



Búsqueda y Suscripción a Ontologías

Una vez dentro de la consola web el próximo paso es buscar **ontologías públicas** que pueda consumir desde mi APP y suscribirme a ellas para poder usarlas.

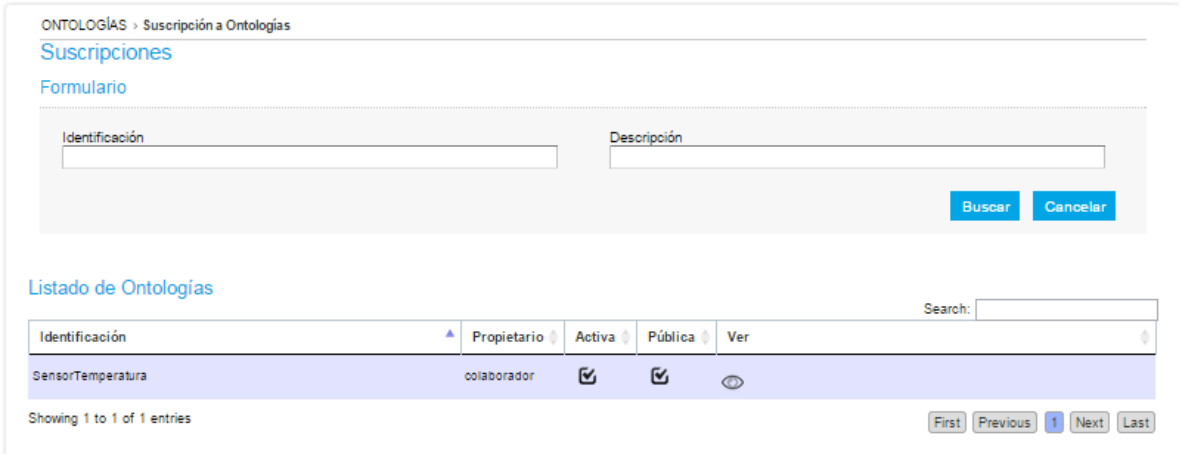
Una **Ontología** representa una Entidad viva en la Plataforma:

- Un usuario **COLABORADOR puede crear Ontologías**
- El **propietario de una Ontología puede hacer CRUDS** (INSERTAR/ ACTUALIZAR/ BORRAR/ CONSULTAR) sobre la Ontología
- El propietario de una Ontología **puede dar permisos a otros usuarios** para bien consultar, bien crear/modificar/borrar.

- El propietario de una Ontología puede hacer **PÚBLICA** una Ontología, en cuyo caso cualquier usuario de la Plataforma puede consultar esa Ontología.

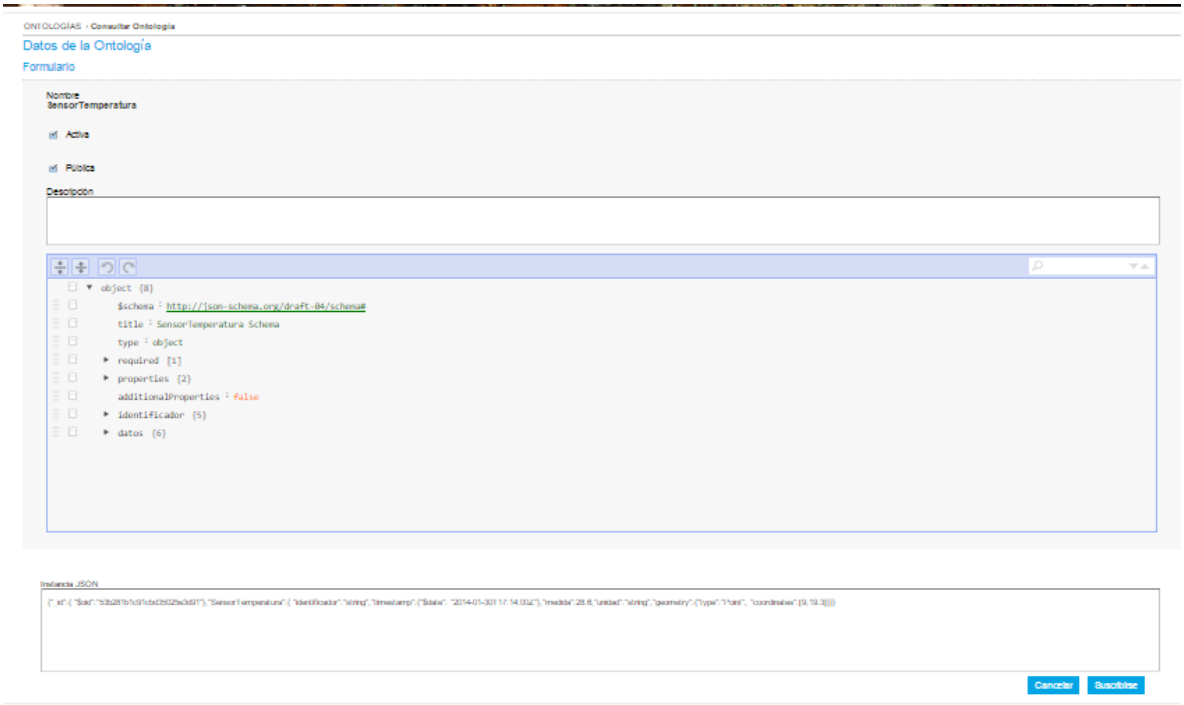
Para buscar y suscribirme a las Ontologías iré a la opción de Menú **ONTOLOGÍAS>Mis suscripciones:**

Y poner un criterio de búsqueda, buscaremos primero ***SensorTemperatura:***



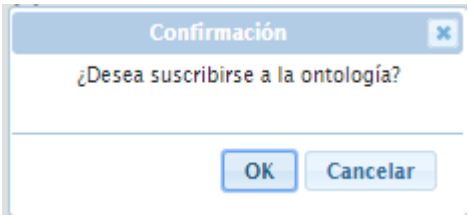
Una vez encontrada seleccionaré el icono de **Ver** 

En la pantalla de detalle puedo ver si una instancia está **Activa, su esquema y un ejemplo de Instancia de esa Ontología:**



Seleccionaré 

, la **suscripción** me permitirá usar esta Ontología (consultarla) desde mi APP.



En la pantalla principal podré verla:

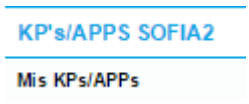
A continuación haré lo mismo sobre la Ontología ***SensorHumedad.***

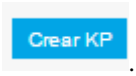
Creación de un KP

Tras suscribirme a las Ontologías debo crear un KP a través de la Consola Web.

Un KP representa una aplicación Sofia2, o de una forma más precisa **representa una conexión de una aplicación Sofia2 hacia la Plataforma.**

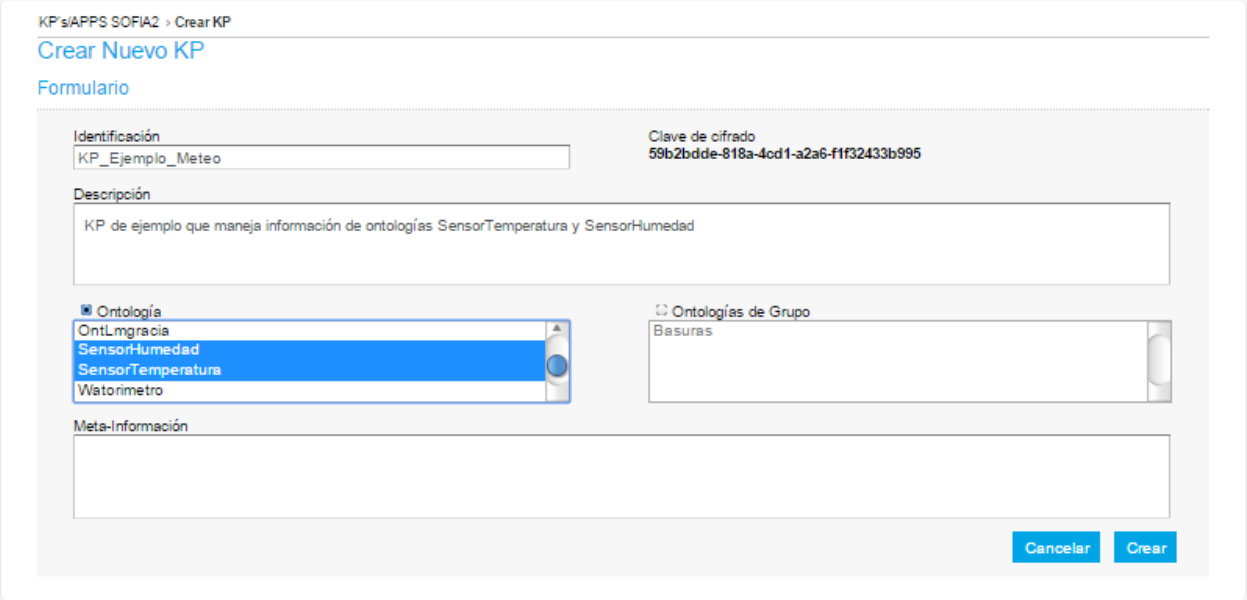
El primero paso es **Crear un KP,** para eso selecciono la opción de menú **KPs/APPs SOFIA2>Mis KPs/APPs:**



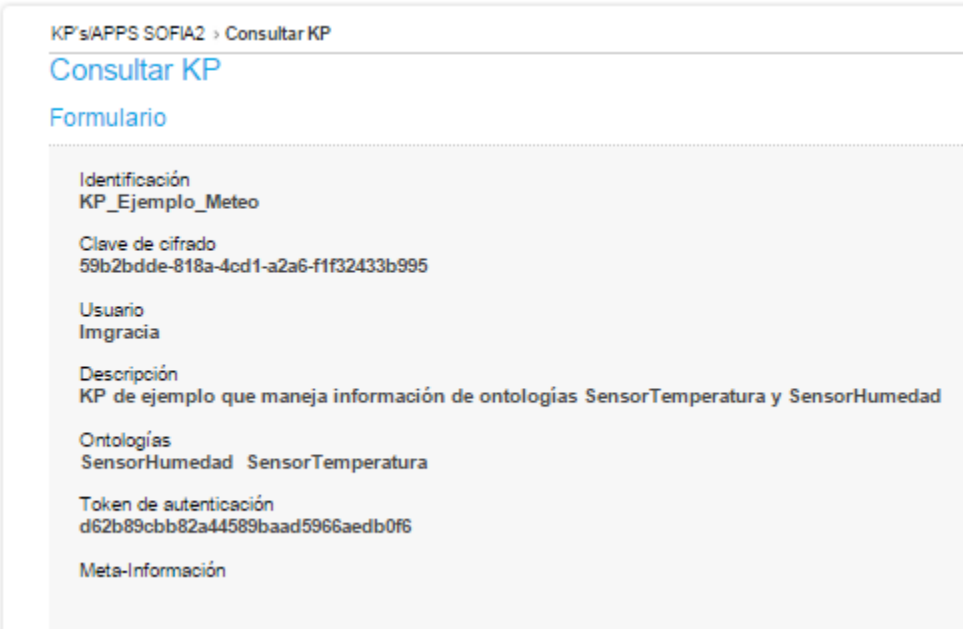
Desde esta pantalla selecciono 

.

Cuando creo un KP debo darle un nombre, descripción y seleccionar las ontologías que se manejarán desde ese KP:



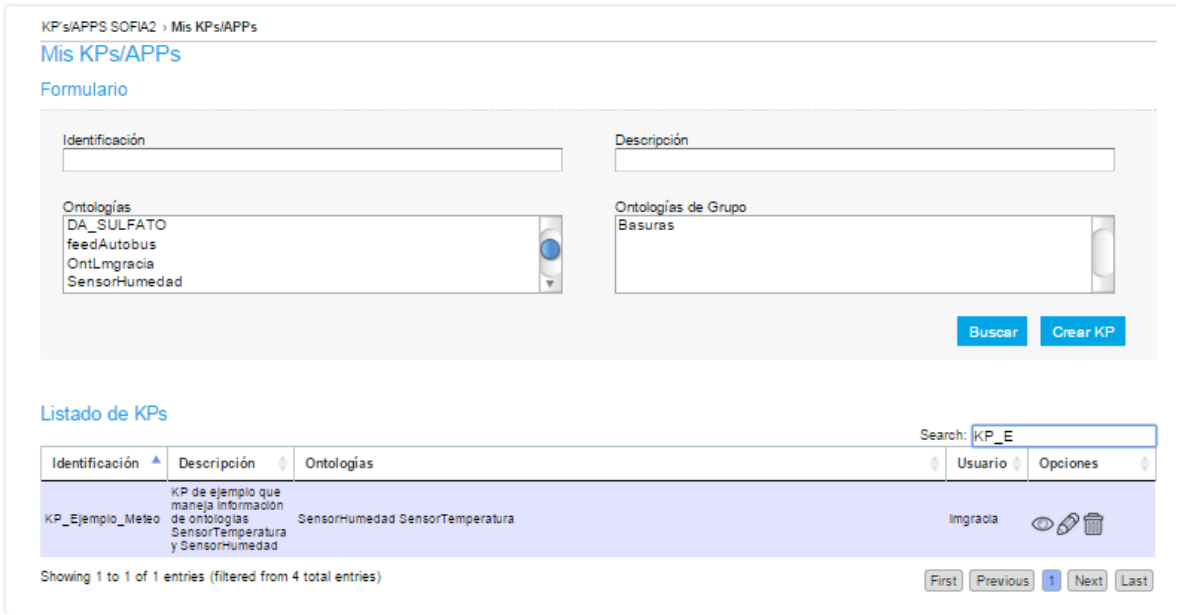
Al crearlo me mostrará esta información:



Sobre estos datos:

- El nombre del KP: **KP_Ejemplo_Meteo**
- El **Token de autenticación** (d62b89cbb82a44589baad5966aedb0f6) es la información que se necesita para conectar desde una APP Sofia2 con la Plataforma.
  - Se pueden gestionar los Tokens desde la opción **Mis Tokens** que permite desactivar el token creado, crear nuevos,. . .
- La **clave de cifrado** sólo es necesaria si quiero usar XXTEA como protocolo de encriptación, se utiliza en dispositivos que no soportan HTTPS como Arduinos.

Tras esto podré ver el KP creado en Mis KPs, desde esta ventana puedo verlo  , editarlo  o borrarlo  .



Creación de una instancia KP

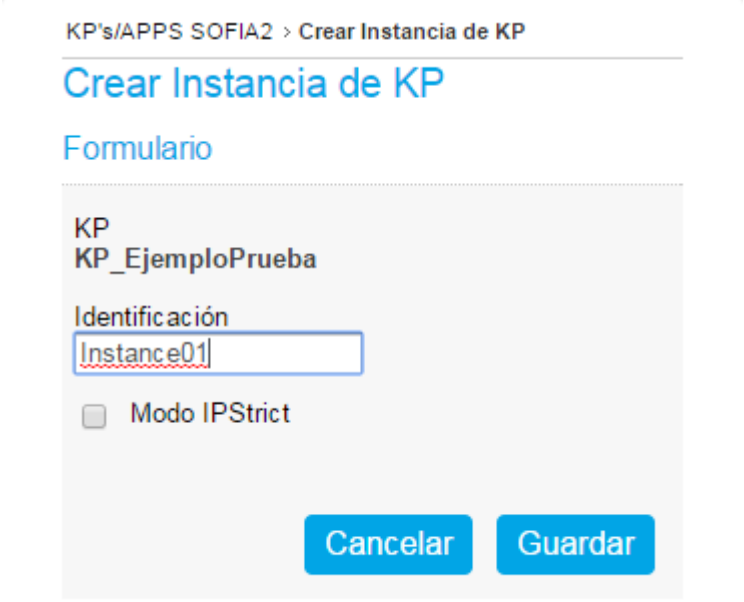
Tras la creación de nuestro KP podemos dejar definida una instancia KP a través de la Consola Web.

Una instancia KP identifica al cliente que se va a conectar a la plataforma Sofia2.

Para continuar con la creación de la instancia, seleccionamos la opción de menú **KPs/APPs SOFIA2 > Mis Instancias KP/APP:**

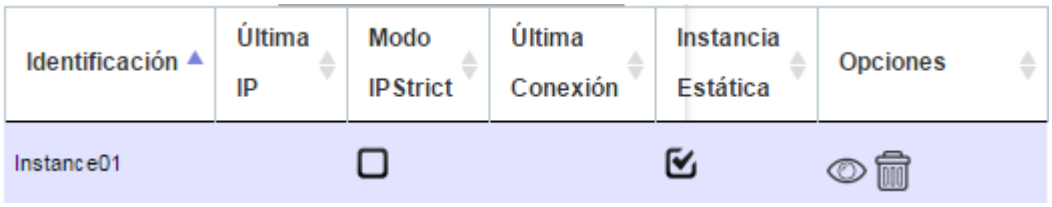
Desde esta pantalla pinchamos en las opciones de nuestro KP en el símbolo  .

Donde nombraremos la instancia que queremos crear.



Al pulsar en el botón 

. Obtendremos la siguiente información:



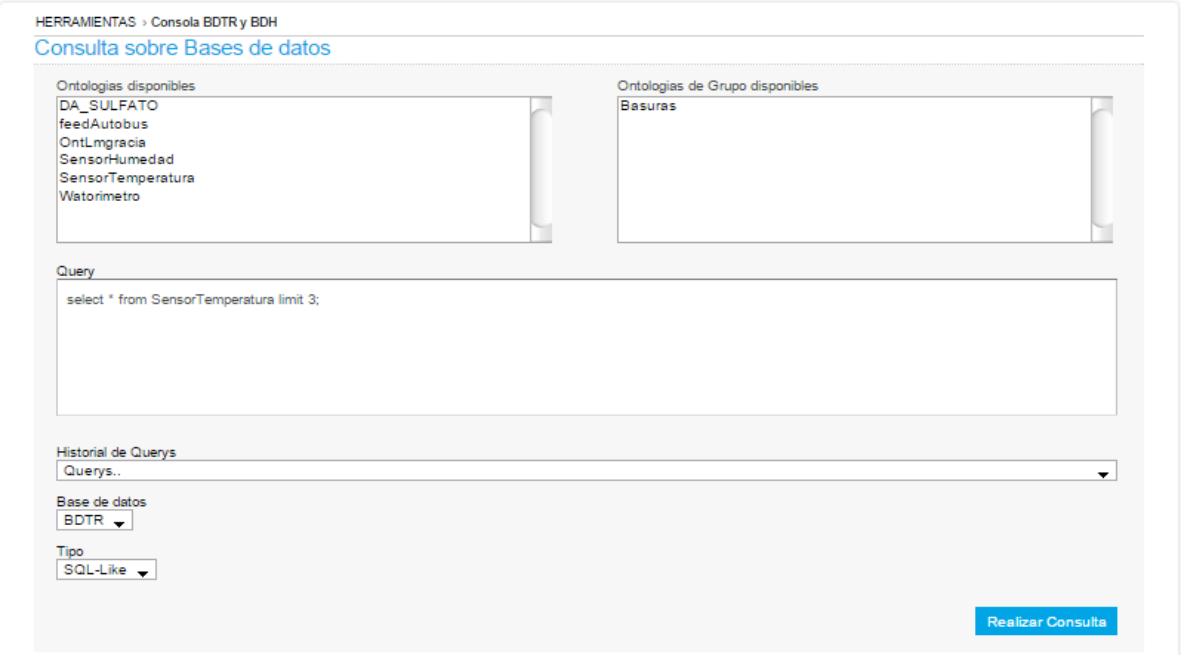
Tras esto podremos ver la instancia KP creada en Mis instancias KPs, desde donde podremos borrarla si fuese necesario  .

Si se necesita profundizar en los campos de las instancias KP se puede obtener mas información en la siguiente guía Consola Web Configuración (Pdf)

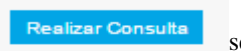
Consultar información desde Consola acceso BDTR

Desde la Consola accederá a la opción de menú **HERRAMIENTAS>Consola BDTR y BDH.**

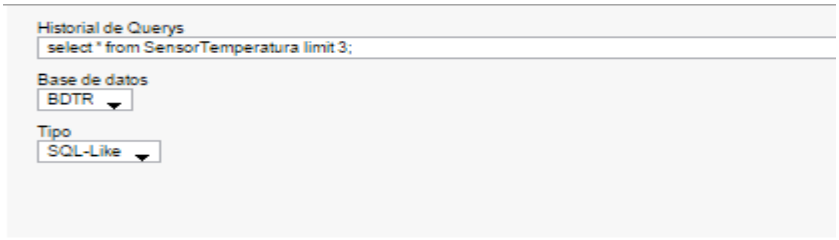
Desde esta consola puedo lanzar queries a las ontologías sobre las que tengo permiso:



Las consultas pueden hacerse hacia la BDTR (almacena información tiempo real) o BDH (información histórica) y puede hacerse en SQL (recomendado) o en lenguaje nativo.

Si selecciono una Ontología y pincho 2 veces se me carga en el textfield **Query,** pinchando 

se ejecuta la query:



Por cada registro se muestra la información de contexto (KP de inserción, usuario de inserción,. . .)

Y la información de la medida, en nuestro caso coordenadas geográficas donde se registró la medida, el identificador del sensor, la medida y unidad y el momento en el que se realizó.

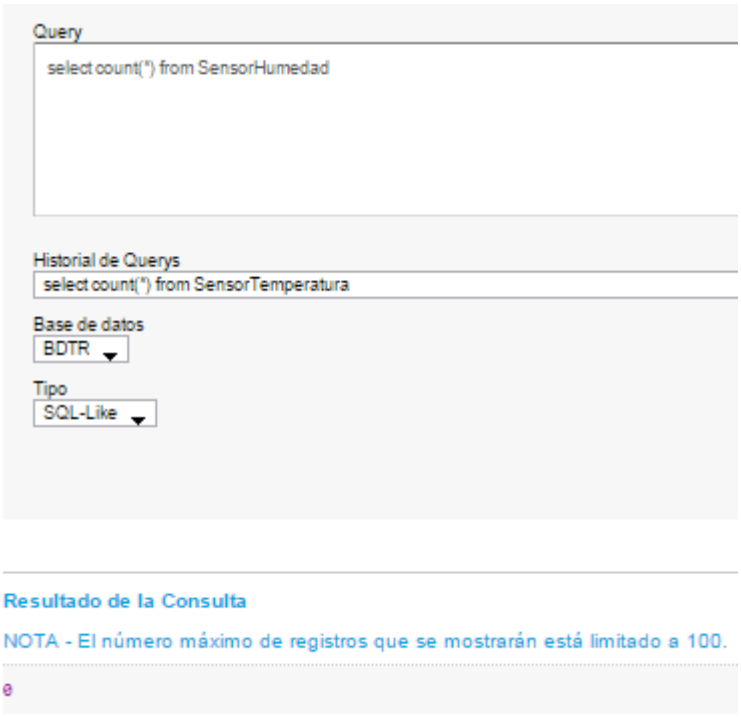
También puedo lanzar consultas con filtros:


Query
select count(*) from SensorTemperatura where SensorTemperatura.medida>16
Historial de Querys
select * from SensorTemperatura limit 3;
Base de datos
BDTR
Tipo
SQL-Like
Resultado de la Consulta
NOTA - El número máximo de registros que se mostrarán está limitado a 100.
3996

como esta:


Query
select count(*) from SensorTemperatura
Historial de Querys
select * from SensorTemperatura limit 3;
Base de datos
BDTR
Tipo
SQL-Like
Resultado de la Consulta
NOTA - El número máximo de registros que se mostrarán está limitado a 100.
9202

Y:



Hasta el momento:

- Nos hemos suscrito a 2 ontologías públicas lo que me permite consultarlas
- Hemos creado un KP con un Token
- Hemos consultado la información que existe en la BDTR para las ontologías seleccionadas.

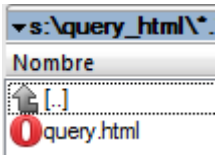
En este paso vamos a crear una APP Sofia2 tanto en Javascript como en Java que me permita consultar la información de estas ontologías, como su objetivo es ser didáctica la construiremos al estilo de la Consola de consultas a la BDTR.

Desarrollo de una APP Sofia2 de consulta en Javascript

Para el ejemplo de APP Javascript partiremos de un ejemplo muy sencillo que podemos descargar desde: http://sofia2.com/docs/query_html.zip

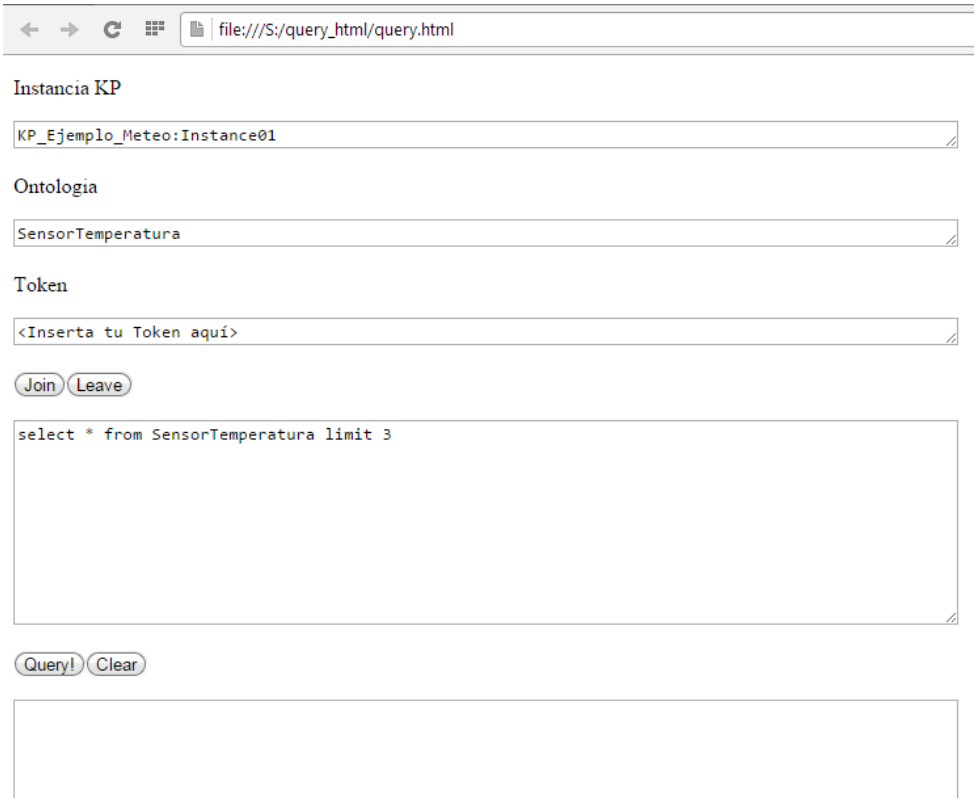
Lo descargaré a mi unidad s:\ en la que monté el SDK.

Una vez allí lo descomprimo en una carpeta **s:\query_html.**



El ejemplo se compone de un único fichero HTML **query.html** que utiliza las librerías Javascript de Sofia2 colgadas en http://sofia2.com.

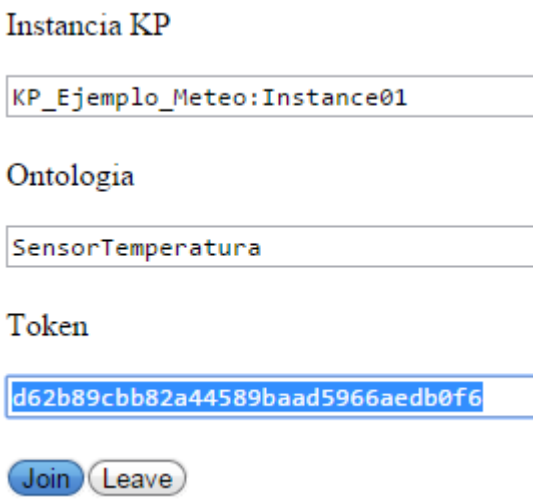
Si abrimos el ejemplo encontraremos un interfaz muy básico



En el que aparece:

- **Instancia de KP**: se compone del nombre del KP creado en el punto 4.2. + ":" + un nombre de instancia. En nuestro caso se llama: **KP_Ejemplo_Meteo:Instance01 **
- **Ontología:** es el nombre de la ontología sobre la que realizaré la consulta, debo ser propietario de esta o estar suscrito a ella como es mi caso
- **Token:** introduciré aquí el Token que cree en el paso 4.2.2: d62b89cbb82a44589baad5966aedb0f6
- **Query:** representa la query en SQL que lanzaré a la instancia CloudLab de Sofia2.
- **Botón Join:** este botón se encarga de logar en la instancia con el Token y la instancia de KP.

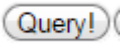
Introduciré mi Token en el campo y pulsaré el botón Join:



Esto me devolverá una SessionKey indicando que estoy conectado a la instancia de Sofia2 CloudLab con esta **Session Key.**

```
session key a7a7fa4b-fd34-4058-bdb0-e3340f74ed56
```

Esta Session Key tiene una fecha de caducidad, por lo que si no usamos esta sesión en 24 horas, caducará.

Una vez tenemos la sesión podemos pulsar el botón 

que nos permite lanzar la query sobre la ontología y pintarla formateada en el campo de texto:

Query! Clear


```
{
  "_id": {
    "$oid": "535f45b7d947103a7029bc92"
  },
  "contextData": {
    "session_key": "be09247a-782f-4b5b-a0aa-bc55c35a5ea7",
    "user_id": "1",
    "kp_id": "60",
    "kp_identificador": "KPvisualizacionHT01",
    "timestamp": {
      "$date": "2014-04-29T08:24:55.695Z"
    }
  },
  "SensorTemperatura": {
    "geometry": {
      "coordinates": [
        40.512967,
        -3.67495
      ],
      "type": "Point"
    },
    "identificador": "S_Temperatura_00001",
    "medida": 17,
    "timestamp": {
      "$date": "2014-04-29T08:24:54.005Z"
    },
    "unidad": "C"
  }
}
```

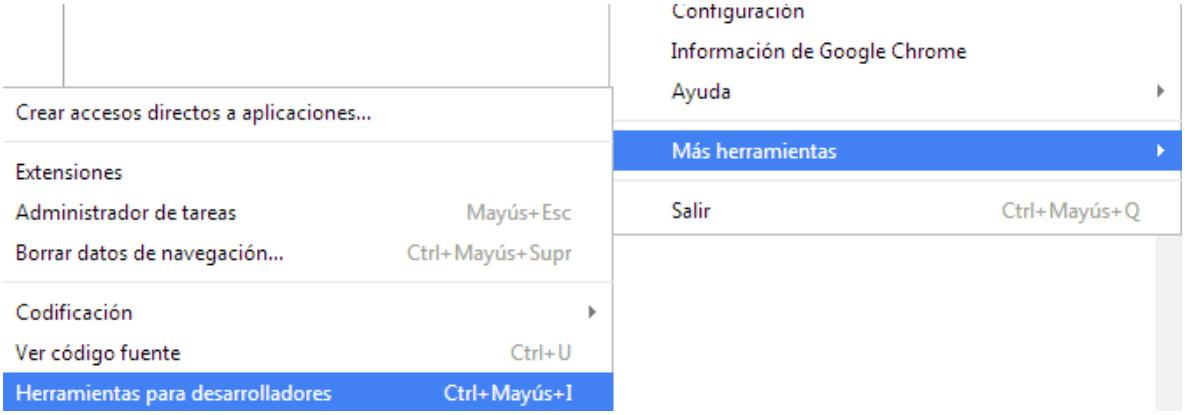

Puedo lanzar otras queries como:

```
select * from SensorTemperatura where SensorTemperatura.medida>17
```

Query! Clear


```
    "kp_id": "60",
    "kp_identificador": "KPvisualizacionHT01",
    "timestamp": {
      "$date": "2014-04-29T08:25:07.684Z"
    }
  },
  "SensorTemperatura": {
    "geometry": {
      "coordinates": [
        40.512274,
        -3.675679
      ],
      "type": "Point"
    },
    "identificador": "S_Temperatura_00001",
    "medida": 29,
    "timestamp": {
      "$date": "2014-04-29T08:25:06.002Z"
    },
    "unidad": "C"
  }
}
```

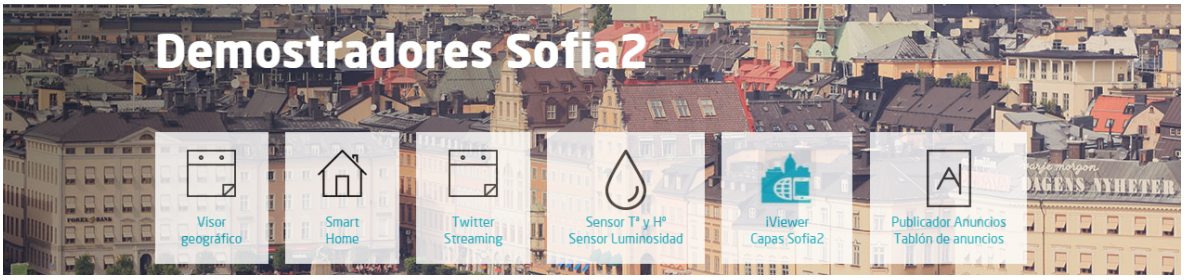

Podemos ver que el código Javascript es muy sencillo y podemos depurarlo a través de las herramientas de desarrollo de cualquier navegador, por ejemplo en Chrome en



Que me permite ir viendo por donde va pasando el código.

Nota: si no se muestra información puede ser debido a que no se encuentre información en la base de datos.

A partir de aquí con un poco de conocimiento en Javascript puedo crear aplicaciones mucho más complejas como pueden verse en los demostradores Sofia2: http://sofia2.com/demostradores.html

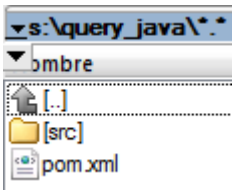


Desarrollo de una APP Sofia2 de consulta en Java

Siguiendo el mismo procedimiento que hemos seguido con la APP Javascript ahora descargaré el ejemplo base Java desde http://sofia2.com/docs/query_java.zip

Lo descargaré a mi unidad **s:** en la que monté el SDK.

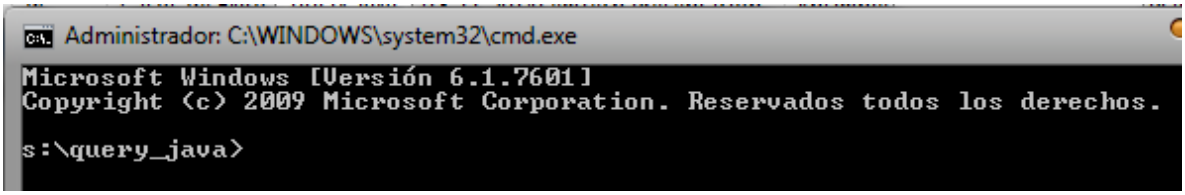
Una vez allí lo descomprimo en una carpeta **s:\query_java.**



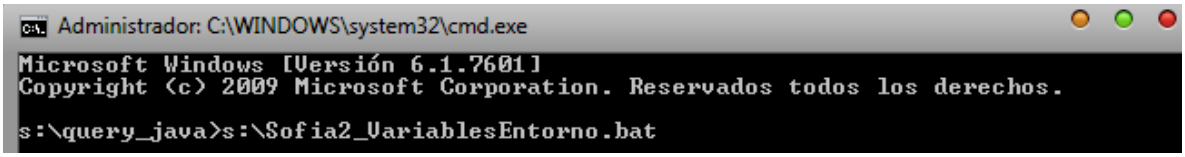
El ejemplo se compone de un proyecto Maven (pom.xml) que contiene una única clase **AppQuerySofia.java**

Al ser un proyecto Maven puedo ejecutarlo a través del comando **mvn,** para eso:

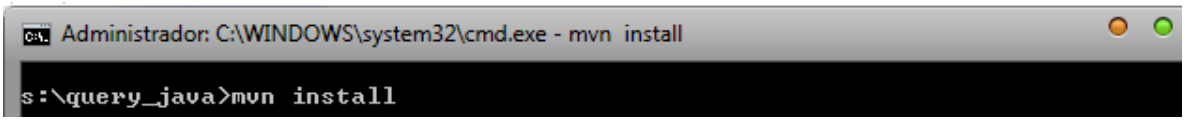
1. Abro una línea de comandos a **s:\query_java**



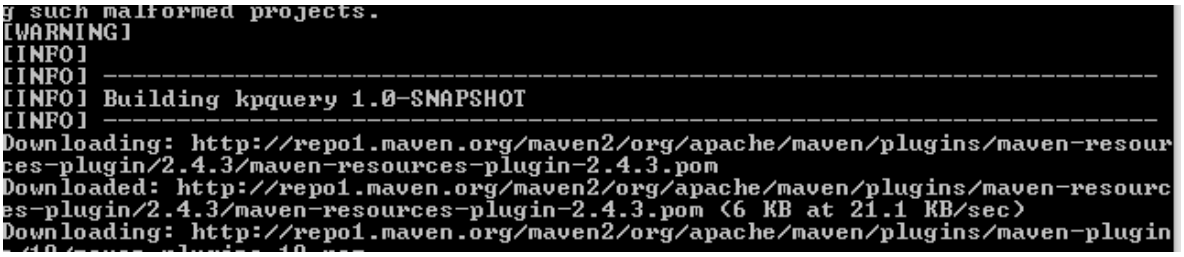
1. Desde esa línea de comandos ejecuto **s:\Sofia2_VariablesEntorno.bat** que establece las variables de entorno, entre ellas JAVA_HOME, MAVEN_HOME. . .



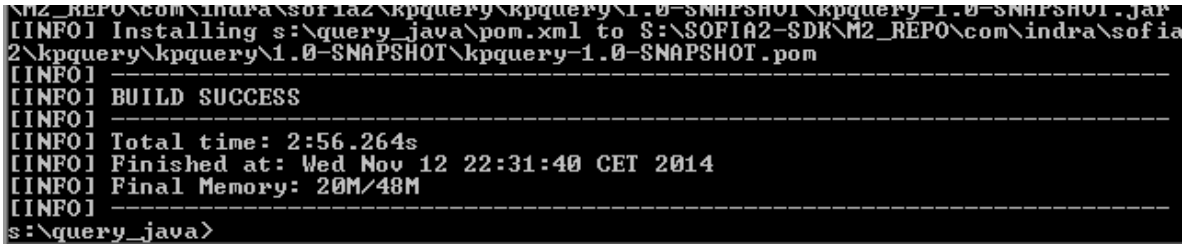
1. Tras esto ya puedo lanzar Maven, lo primero será **construir el proyecto** a través de Maven, esto se hace con el comando **mvn install** que lo primero que hace es descargar todas las dependencias necesarias desde su repositorio:



La descarga esta primera vez puede tardar varios minutos.



Si todo ha ido bien finalmente obtendremos un:

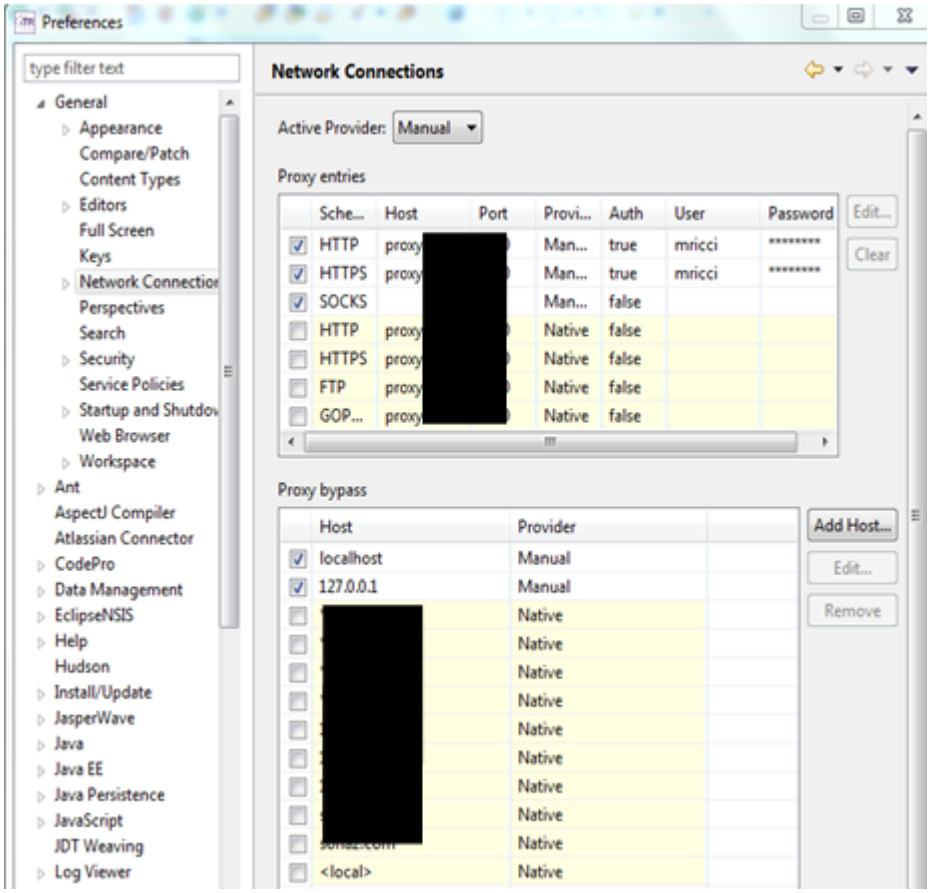


NOTA: Si se produce un error lo más normal es que se deba a que tengo un **proxy** para el acceso a Internet, en ese caso tengo que editar el fichero **s:\SOFIA2-SDK\MAVEN\conf\settings.xml** añadiendo una entrada <proxy> en <proxies>

```
<proxies>
  <!-- proxy
   | Specification for one proxy, to be used in connecting to the network.
   |
  <proxy>
    <id>optional</id>
    <active>true</active>
    <protocol>http</protocol>
    <username>proxyuser</username>
    <password>proxypass</password>
    <host>proxy.host.net</host>
    <port>80</port>
    <nonProxyHosts>local.net|some.host.com</nonProxyHosts>
  </proxy>
  -->

</proxies>
```

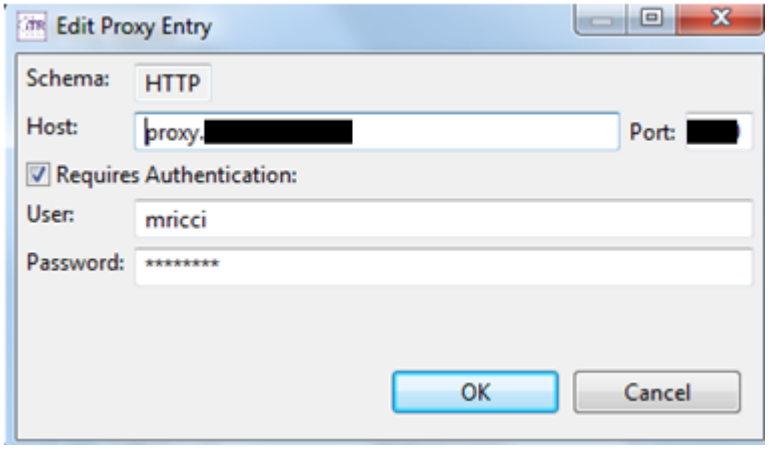
NOTA 2: Si realizando el paso anterior continua produciendose un error durante la descarga,deberemos entrar en las opciones del entorno, mostradas a continuación y realizar los siguientes cambios.

Entrando en el menú - Window > Preferences > General > Network Connection.



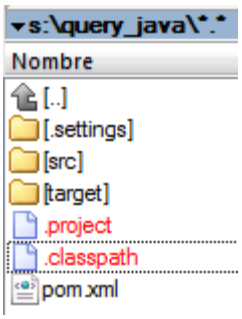
En el campo Active Provider cambiaremos el valor a **Manual.**

Y tendremos que editar **HTTP** y **HTTPS** introduciendo nuestras credenciales.



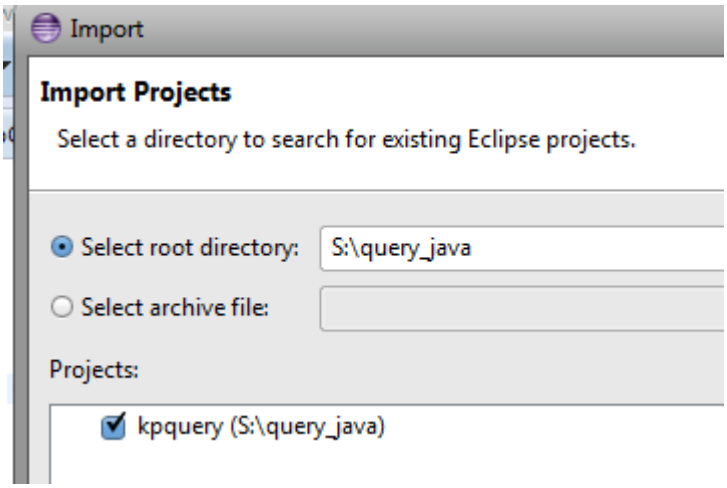
1. Tras esto lo que haremos será generar el proyecto de Eclipse y cargarlo en el IDE de Sofia2. Para compilar Ejecuto desde la línea de comandos abierta ejecuto **mvn eclipse:eclipse.**

   Esto al finalizar habrá creado los ficheros **.classpath y .project** de Eclipse.

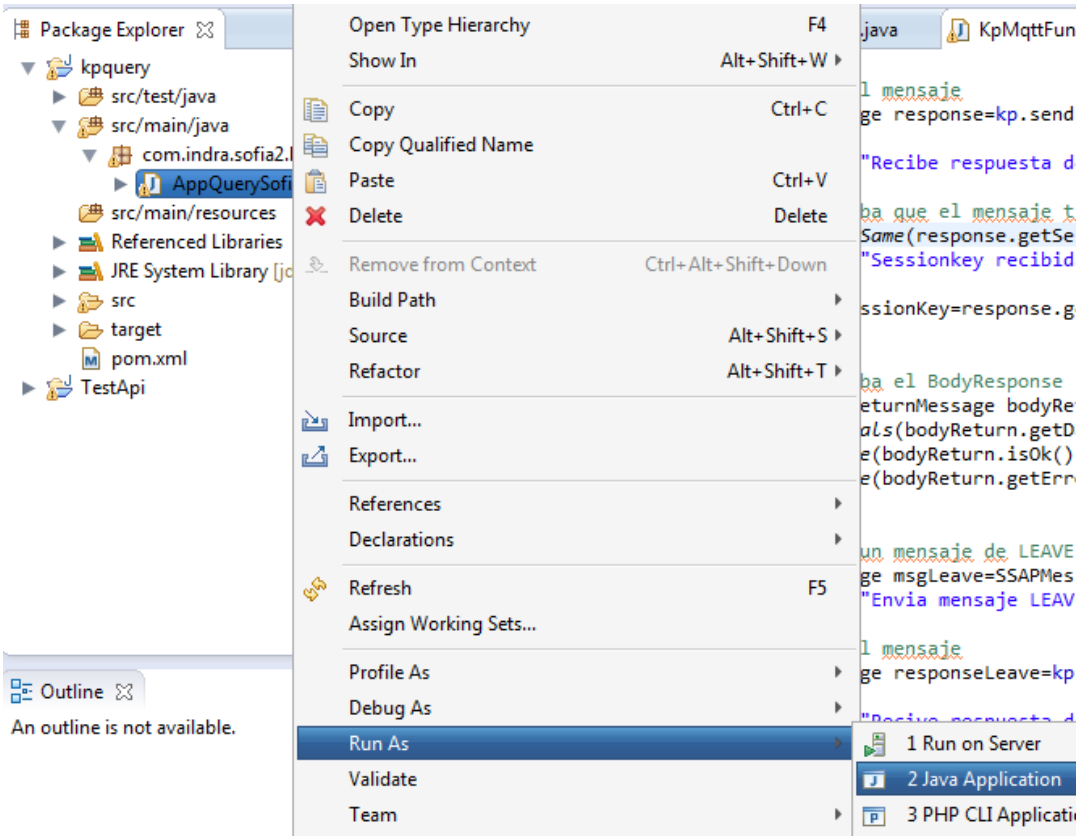


1. Si no lo tengo abierto, abriré el IDE Sofia2 con el comando **S:\Sofia2_IDE.bat** y cardaré el proyecto como se indica en el punto 3.3: Lanzar Ejemplo:

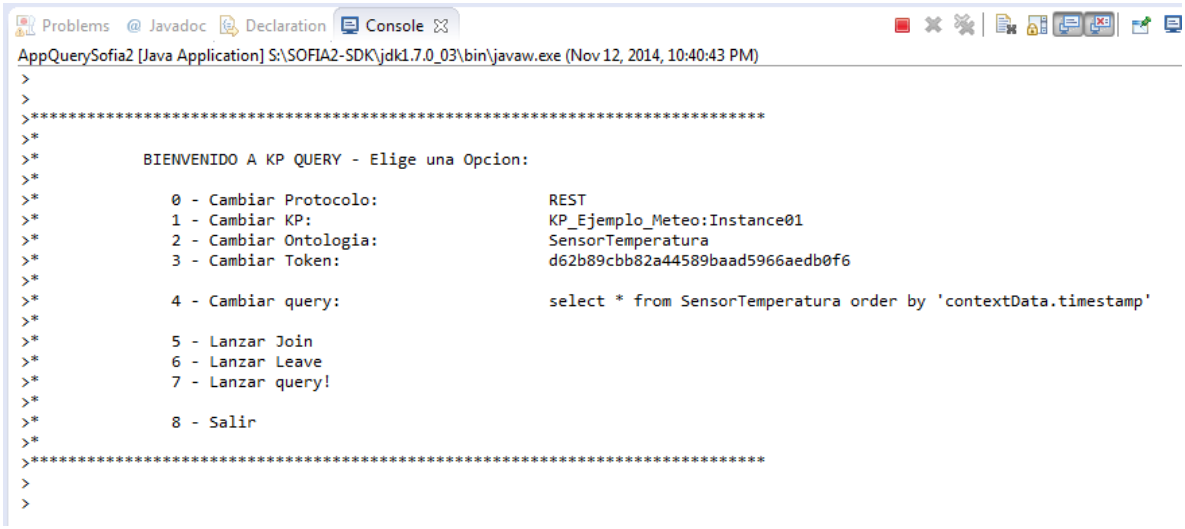
   File>Import>General>Existing Projects into Workspace >"s:\query_java\"



1. Desde el proyecto abierto en el Eclipse puedo lanzar la clase AppQuerySofia2.



Esto lanzará en **Console** esta aplicación:



Desde aquí como hicimos en la APP Javascript puedo ir cambiando Token, KP, query,. . .

Cambiemos el Token por el nuestro: d62b89cbb82a44589baad5966aedb0f6

```
3
>Introduce nuevo token
d62b89cbb82a44589baad5966aedb0f6
>token cambiado con exito
>
```

Luego puedo lanzar el Join para iniciar una sesión con la Plataforma:

```
>Lanzando join...
log4j:WARN No appenders could be found for logger (org.apache.cxf.common.logging.LogUtils).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
>join correcto, session b7a19bdf-16f5-4393-a540-12880d3bc774
```

Y finalmente una query:

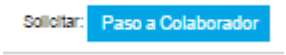
```
>Lanzando query...
>insercion de mensaje correcta
sun.net.www.protocol.http.HttpURLConnection$HttpInputStream@15434d8
[ {
  "_id" : {
    "$oid" : "535f45b7d947103a7029bc92"
  },
  "contextData" : {
    "session_key" : "be09247a-782f-4b5b-a0aa-bc55c35a5ea7",
    "user_id" : "1",
    "kp_id" : "60",
    "kp_identificador" : "KPvisualizacionHT01",
    "timestamp" : {
      "$date" : "2014-04-29T08:24:55.695Z"
    }
  },
  "SensorTemperatura" : {
    "geometry" : {
      "coordinates" : [ 40.512967, -3.67495 ],
      "type" : "Point"
    },
    "identificador" : "S_Temperatura_00001",
    "medida" : 17,
    "timestamp" : {
      "$date" : "2014-04-29T08:24:54.005Z"
    },
    "unidad" : "C"
  }
}, {
  "_id" : {
```

Nota: si no se muestra información puede ser debido a que no se encuentre información en la base de datos.

Desarrollo de una APP Sofia2 que inserte datos.

Hasta ahora hemos visto como desarrollar APPs de consumo de información. Pero estas mismas aplicaciones también pueden insertar información, siempre que tengan permisos para insertar datos en la Plataforma, esto se puede conseguir:

1. Teniendo rol COLABORADOR, lo que nos permite crear Ontologías y luego insertar datos conforme a estas.
    - En la plataforma Sofia2 CloudLab se puede solicitar rol COLABORADOR desde la pantalla principal de la consola: http://sofia2.com/console/ a través del link:



El administrador validará la petición, si se rechaza se le informará de los motivos.

2. Que un usuario con rol COLABORADOR nos cree un Token con permisos para insertar instancias de una Ontología de la que es propietario.

Cómo desarrollar sobre la plataforma Sofia2

Para poder desarrollar aplicaciones SOFIA2 es necesaria la instalación del SDK de SOFIA2 conforme se indica en el punto Descarga e Instalación del SDK de SOFIA2

Una vez instalado el SDK de SOFIA2 los **pasos para desarrollar sobre la plataforma SOFIA** son los siguientes:

Registro en la plataforma SOFIA2

La plataforma SOFIA es un entorno de interoperabilidad controlado.

Diferentes KPs intercambiarán información para interoperar.

Cada KP tiene un usuario propietario, pudiendo un mismo usuario ser propietario de varios KPs.

Alta de usuario en la plataforma

Un usuario puede darse de alta en la plataforma por sí mismo o por un administrador.

- **Alta por sí mismo**: En la URL http://sofia2.com/console/ la pantalla de **login** proporciona el enlace *crear una cuenta nueva.*
- **Alta por administrador**: El administrador dispone en la plataforma de una sección de **Gestión de Usuarios**. Desde esta sección se proporciona al administrador de un formulario para dar de alta nuevos usuarios a la plataforma, con el rol necesario:

Una vez dado de alta el usuario en la plataforma, este tendrá acceso a la misma, en función a su rol. Asimismo, como veremos más adelante, los KPs de los que el usuario sea propietario podrán establecer una sesión lógica para enviar/recibir información.

Ontologización de la información

La plataforma SOFIA propone como clave de la interoperabilidad de aplicaciones, la definición ontológica de la información que intercambian las aplicaciones.

Una ontología es una clasificación de la información, que estandariza las propiedades de los conceptos del dominio con los que interoperarán los distintos KPs, de manera que distintos KPs trabajando con las mismas ontologías pueden intercambiar información a través de la plataforma de una manera totalmente desacoplada, mediante el intercambio de instancias de tales ontologías.

SOFIA2 propone el intercambio de información entre KPs en formato JSON. De modo que una ontología en SOFIA2 es la especificación unívoca de un formato JSON de información, por lo que estas ontologías se definen de acuerdo a un schema JSON http://json-schema.org/.

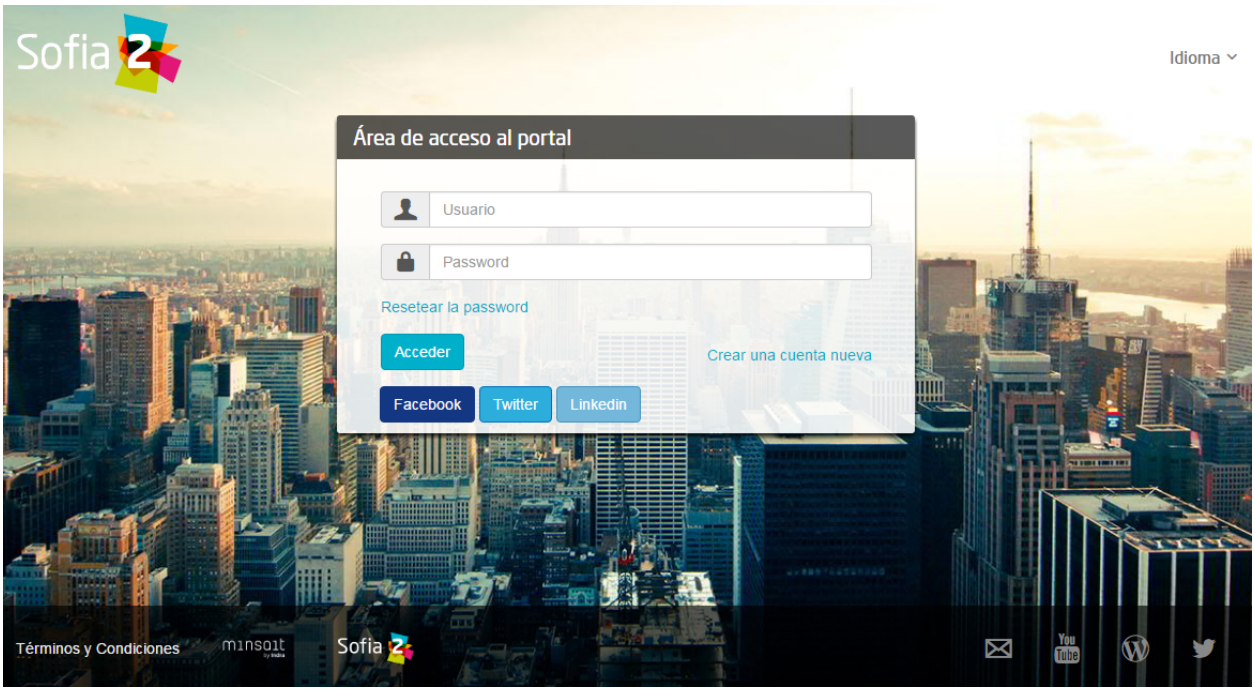
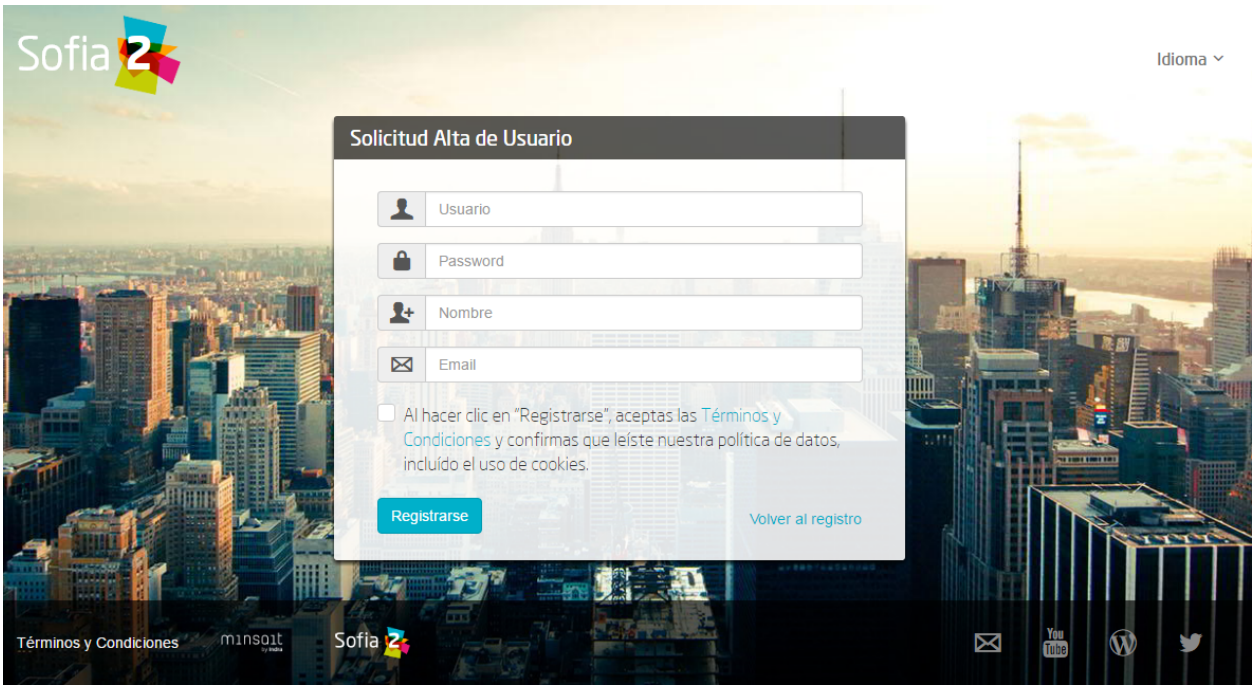
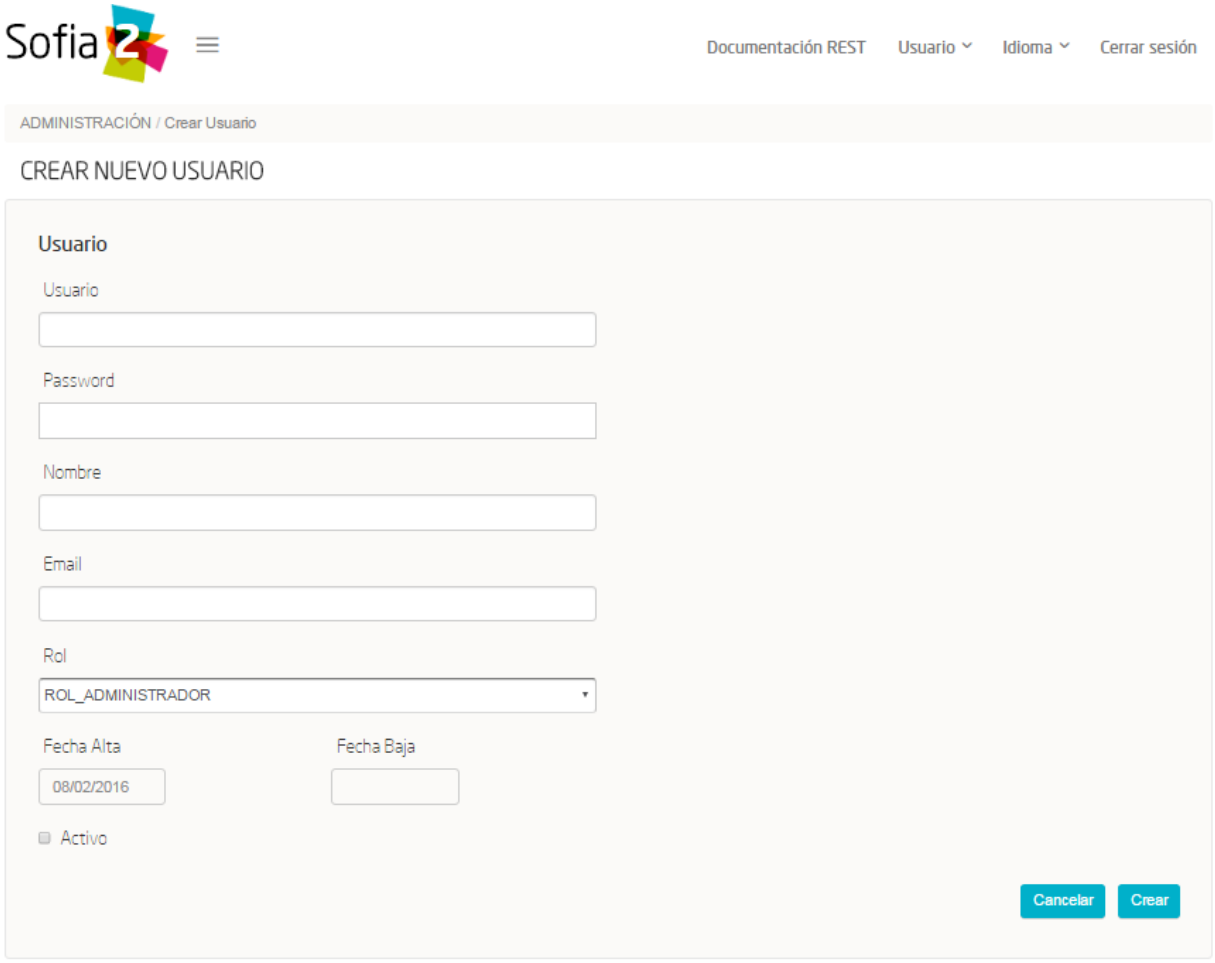


Fig. 24.1: Que redirecciona a un formulario de alta de usuario:




Sofia2
Documentación REST
Usuario
Idioma
Cerrar sesión
ADMINISTRACIÓN / Crear Usuario
CREAR NUEVO USUARIO
Usuario
Usuario
Password
Nombre
Email
Rol
ROL_ADMINISTRADOR
Fecha Alta
08/02/2016
Fecha Baja
Activo
Cancelar
Crear

Identificación de los conceptos de la información

Consiste en identificar las entidades que agruparán los datos que intercambiarán los KPs a través de la plataforma.

Un concepto agrupará uno o varios datos relacionados y de interés para los KP, de manera que un KP productor de información recolectará todos estos datos, los agrupará conforme a la especificación hecha en la ontología que describa tal concepto y los enviará de esta manera, normalizados a la plataforma. Mientras que un KP consumidor de información, recibirá estos datos normalizados y los explotará.

Por ejemplo, en un SmartSpace de sensorización de una SmartCity, conceptos relevantes para ser ontologizados serian:

- SensorTemperatura
- SensorHumedad
- SensorCO2
- SensorMovimiento
- Semaforo
- AspersorRiego

  .....

Identificación de los atributos de los conceptos

Consiste en identificar los datos agrupados por los conceptos de información y que serán relevantes para los KPs

Por ejemplo para el concepto SensorTemperatura, se podrían considerar los siguientes atributos:

- **IdentificadorSensor**: string
- **Timestamp**: integer
- **Medida**: double
- **Unidad**: string
- **Localización GPS**: Object
  - **Altitud**: double
  - **Latitud**: double
  - **Longitud**:double

Modelado en formato JSONSchema

Identificados los datos a intercambiar, el siguiente paso es estandarizarlos para que tengan una definición unívoca para los KPs en la plataforma. En esto consiste la ontologización de la información, donde cada concepto relevante se define de acuerdo a un schema JSON.

Por ejemplo, el concepto SensorTemperatura con los atributos identificados anteriormente se definiría en formato JSONSchema del siguiente modo:

SensorTemperatura.json

```
{
  "$schema": "http://json-schema.org/draft-03/schema#",
  "title": "SensorTemperatura Schema",
  "type": "object",
  "properties": {
```

```
        "_id": {
            "type": "object",
            "$ref": "#/identificador"
        },
        "SensorTemperatura": {
            "type": "string",
            "$ref": "#/datos"
        }
    },
    "identificador": {
        "title": "id",
        "description": "Id insertado del SensorTemperatura",
        "type": "object",
        "properties": {
            "$oid": {
                "type": "string",
                "required": false
            }
        }
    },
    "datos": {
        "title": "datos",
        "description": "Info SensorTemperatura",
        "type": "object",
        "properties": {
            "identificador": {
                "type": "string",
                "required": true
            },
            "timestamp": {
                "type": "integer",
                "minimum": 0,
                "required": true
            },
            "medida": {
                "type": "number",
                "required": true
            },
            "unidad": {
                "type": "string",
                "required": true
            },
            "LocalizacionGps": {
                "required": true,
                "$ref": "#/gps"
            }
        }
    },
    "gps": {
        "title": "gps",
        "description": "Gps SensorTemperatura",
        "type": "object",
        "properties": {
            "altitud": {
                "type": "number",
                "required": false
            },
            "latitud": {
```

```
                "type": "number",
                "required": true
            },
            "longitud": {
                "type": "number",
                "required": true
            }
        }
    },
    "additionalItems": false
}
```

Por lo que la información que los KPs produzcan/consuman de este tipo de sensor quedará normalizada al siguiente formato:

SensorTemperatura-instance.json

```
{
 "SensorTemperatura":
  {
  "identificador":"ST-TA3231-1",
  "timestamp":1357930309163,
  "medida":25.1,
  "unidad":"C",
   "localizacionGps":
    {
      "altitud":0.0,
      "latitud":40.512967,
      "longitud":-3.67495
    }
  }
}
```

Alta de la ontología en la plataforma

Una ontología deberá ser registrada en la plataforma para quedar operativa y poder ser utilizada por los KPs para insertar/consumir la información descrita por la misma.

Para ello se dispone en la plataforma de un apartado de **Gestión de Ontologías**, donde editar y dar de alta nuevas ontologías, así como su administración posterior:

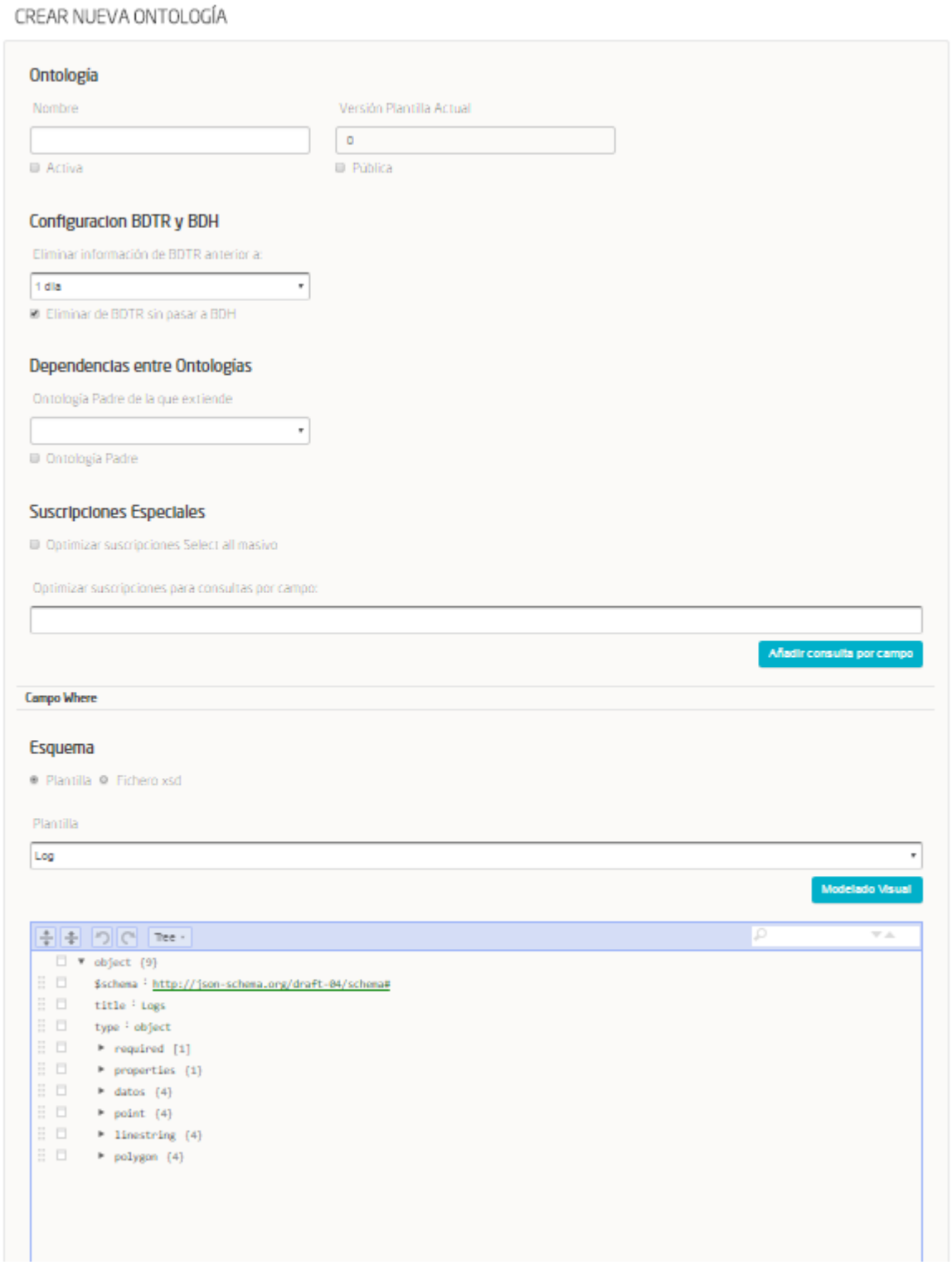
Desarrollo de Clientes SOFIA2 (KPs)

Un KP es cualquier aplicación que produce o consume información para colaborar con otros a través de la plataforma, formando de este modo un **Smart Space** con aquellos otros KPs con los que colabora.

Para desarrollar un KP, aparte de programar su lógica de negocio, hay que realizar los siguientes pasos sobre la plataforma:

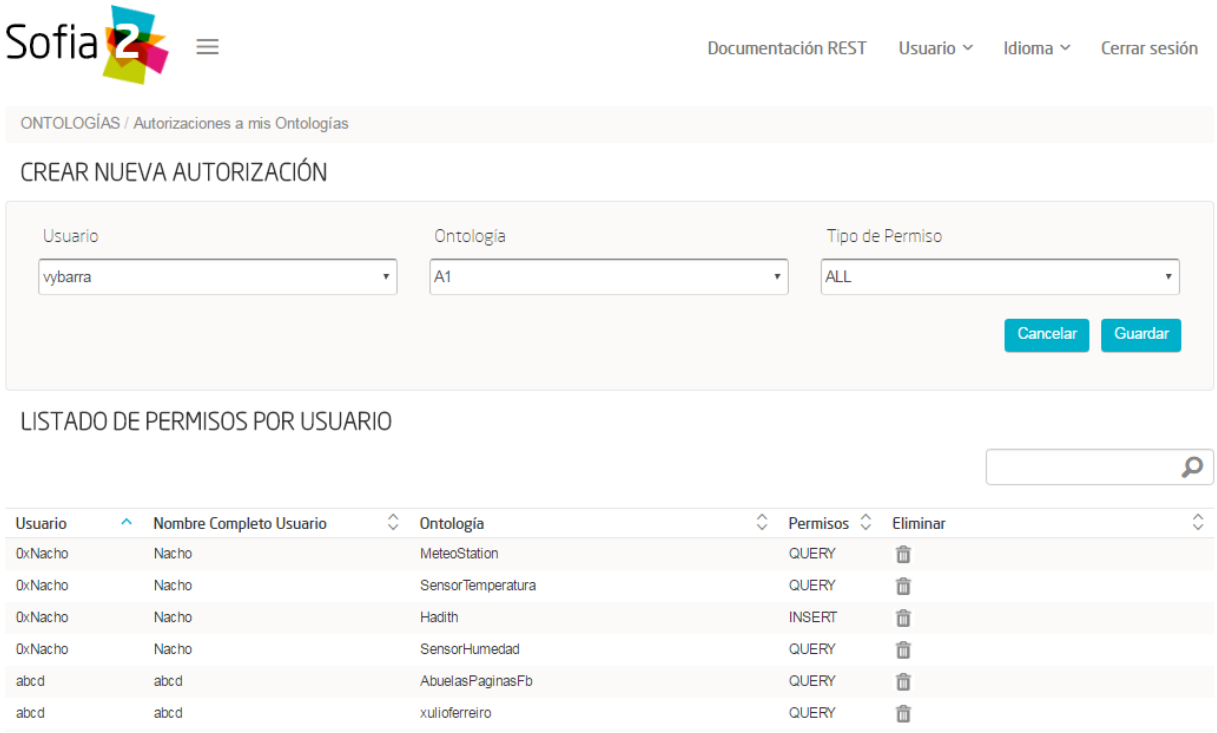
Alta de permisos para usuario propietario en la plataforma

Para que los KPs de un usuario puedan producir o consumir datos de una determinada ontología, el usuario deberá disponer de los permisos adecuados sobre dicha ontología.


CREAR NUEVA ONTOLOGÍA
Ontología
Nombre
Versión Plantilla Actual
0
Activa
Pública
Configuracion BDTR y BDH
Eliminar información de BDTR anterior a:
1 día
Eliminar de BDTR sin pasar a BDH
Dependencias entre Ontologías
Ontología Padre de la que extiende
Ontología Padre
Suscripciones Especiales
Optimizar suscripciones Select all masivo
Optimizar suscripciones para consultas por campo:
Añadir consulta por campo
Campo Where
Esquema
Plantilla
Fichero xsd
Plantilla
Log
Modelado Visual
Tree
object {9}
$schema : http://json-schema.org/draft-04/schema#
title : Logs
type : object
required [1]
properties {1}
datos {4}
point {4}
linestring {4}
polygon {4}

Una ontología registrada en la plataforma puede no ser visible para un usuario, o en caso de serlo, puede estar limitado en sus operaciones a determinados permisos.

La plataforma proporciona a los administradores en la sección de **Ontologías > Autorizaciones a mis Ontologías**, una pantalla para administrar las autorizaciones de un usuario sobre las distintas ontologías registradas.



De manera que en función del tipo de KPs que vaya a desarrollar un usuario, habrá que proporcionarle permiso de **INSERT**, **QUERY** ó **ALL** sobre la ontología que describe los datos que manejará el KP.

Alta de KP en la plataforma

Un usuario deberá registrar en la plataforma sus KPs, de lo contrario, la plataforma rechazará la conexión de los mismos.

Para registrar un KP, la plataforma proporciona la sección **Gestion KPs**, donde un usuario podrá crear un nuevo KP o administrar los que ya tiene dados de alta:

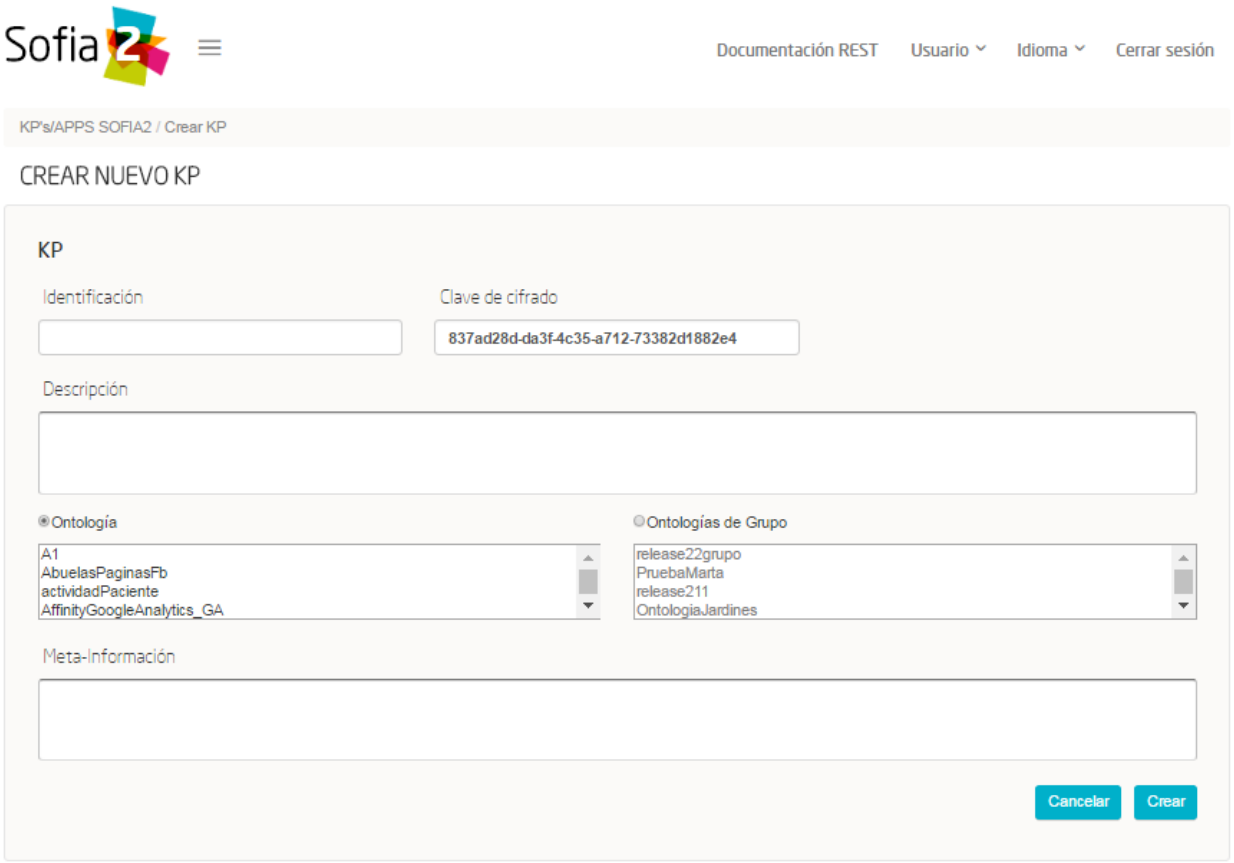
Como vemos, un KP podrá hacer uso de una o varias ontologías, siendo esta la información que producirá o consumirá de la plataforma.

Una vez registrado en la plataforma, el KP ya podrá establecer conexiones con la misma.

Conexión del KP con la plataforma

La conexión de un KP con la plataforma debe ser vista como dos tipos de conexión

- **Conexión Física**: Establecida por el protocolo de trasporte utilizado para la conexión por un KP (TCP/IP, MQTT, JMS, Ajax-Push...). La manera de realizar esta tipo de conexión depende en gran medida del API de KP utilizado (Java, Javascript, Arduino, C++...).


Sofia 2
Documentación REST
Usuario
Idioma
Cerrar sesión
KP's/APPS SOFIA2 / Crear KP
CREAR NUEVO KP
KP
Identificación
Clave de cifrado
837ad28d-da3f-4c35-a712-73382d1882e4
Descripción
Ontología
Ontologías de Grupo
A1
AbuelasPaginasFb
actividadPaciente
AffinityGoogleAnalytics_GA
release22grupo
PruebaMarta
release211
OntologiaJardines
Meta-Información
Cancelar
Crear

- **Conexión Lógica**: Establecida por el protocolo SSAP (Smart Space Access Protocol) de mensajería definido en SOFIA. Es común a todos los APIs de KP.

  Nos vamos a centrar en la conexión Lógica que debe mantener un KP con la plataforma.

  Para que un KP puede conectarse a la plataforma y producir/consumir datos e interoperar con otros KP, es necesario que abra una sesión con un SIB de la plataforma.

  El protocolo SSAP proporciona dos operaciones en este sentido:

- **JOIN**: Donde un KP informa a la plataforma el usuario y password de su propietario así sus datos de instancia, de manera que si todo es correcto, la plataforma autentica al KP y abre una sesión con el mismo.
- **LEAVE**: Donde un KP informa a la plataforma que va a cerrar la sesión.

  Mientras exista una sesión entre el KP y la plataforma, el KP podrá utilizar el resto de operaciones del protocolo SSAP para producir/consumir información.

  Para obtener más información acerca de las Apis distribuidas por la plataforma se recomienda revisar la Guía de Apis SOFIA2, en donde se indica su uso e instalación.

Captación/Explotacion de la información

Constituye parte de la lógica de negocio de un KP y es independiente de la plataforma. Depende exclusivamente de la naturaleza y propósito del KP el modo de captar la información de las distintas fuentes si es productor de información, así como su explotación una vez recibida la información si se trata de un KP consumidor.

Transformación de la información a formato ontológico

Como ya se ha comentado en el presente documento la información que envíe un KP productor a la plataforma debe cumplir con el formato definido en la ontología que la representa. De manera que con tal información se deberá construir mensaje JSON que agrupe tales datos cumpliendo el **JSONSchema** de la ontología correspondiente, convirtiéndose de este modo los datos en una instancia de la ontología.

Envío a la plataforma según protocolo SSAP

Una vez construido el mensaje JSON con los datos a enviar a la plataforma. Se deberá construir el mensaje SSAP de INSERT correspondiente y que integrará tales datos.

La plataforma validará que el usuario propietario del KP tiene el correspondiente permiso sobre la ontología que representan tales datos, así como que los datos cumplen con el Schema JSON de la ontología. Si hay algún problema, se notificará al KP, si todo va bien, tales datos se agregan a la base de datos de tiempo real del SIB, quedando disponible para el resto de KPs.

Al igual que todas las operaciones SSAP, la operación INSERT está contemplada en todos los API de KP proporcionados. Para obtener más información acerca de la mensajería SSAP se recomienda revisar la Guía de Apis SOFIA2.

Consulta/Suscripción de la información según protocolo SSAP

La información enviada a la plataforma por los KPs, puede ser consultada por otros KPs, bien explícitamente mediante la operación QUERY del protocolo SSAP, bien en modo suscripción a futuras entradas de información mediante la operación SUBSCRIBE.

En ambas operaciones se indican a la plataforma los criterios de la consulta.

En la operación **QUERY**, nos serán devueltas las instancias existentes actualmente en la base de datos de tiempo real que cumplen con los criterios de la consulta.

En la operación **SUBSCRIBE**, la plataforma nos enviará en el futuro nuevas instancias cada vez que un KP las inserte y cumplan con los criterios de la consulta.

SOFIA permite que las operaciones de **QUERY** puedan ser:

- **Query de tipo Nativo**: cuando la query es resuelta por el motor de BDTR subyacente, siendo en la implementación de referencia *MongoDB.*

  db.SensorTemperatura.find().limit(3);

- **Query de tipo SQL-Like**, cuando la query es transformada por SOFIA al lenguaje de query subyacente.

  select * from SensorTemperatura limit 3;

  Podemos encontrar más información sobre los tipos de Query en la guía SOFIA2-APIs SOFIA2.

Recepción de la información a formato ontológico

Del mismo modo que un KP envía la información a la plataforma de acuerdo a una ontología, cuando un KP recibe información de la plataforma, esta información también viene en formato JSON según la ontología correspondiente, de modo que una vez extraída del mensaje SSAP correspondiente, el KP puede tratar dicha información según la definición de la ontología en el ***JSONSchema*** que la define.

Colaboración de KPs en tiempo real

Representa la colaboración entre KPs formando un **Smart Space**. Requiere que:

- Las ontologías representando la información a intercambiar están dadas de alta en la plataforma por medio de la **Consola Web**.
- Los usuarios propietarios de los distintos KPs están dados de alta en la plataforma por medio de la **Consola Web** y existen Tokens activos asociados a los mismos.
- Los usuarios tienen permisos adecuados sobre las ontologías que utilizan sus KPs.
- Los KPs **productores** y **consumidores** dados de alta en la plataforma.
- Los KPs **productores** están funcionando:
  - Han conectado con la plataforma con mensaje SSAP JOIN.
  - Están captando información según su lógica de negocio.
  - Transforman la información captada a ontología correspondiente.
  - Insertan la información en la plataforma con mensajes SSAP INSERT.
- KPs **consumidores** funcionando:
  - Han conectado a la plataforma con mensaje SSAP JOIN
  - Están recuperando información de la plataforma:
    - Explícitamente mediante SSAP QUERY.
    - En modo suscripción mediante SSAP SUBSCRIBE.
  - Explotan la información según su lógica de negocio.

Ejemplo Práctico

A continuación vamos a desarrollar un ejemplo práctico siguiendo los pasos descritos.

En nuestro ejemplo vamos a desarrollar dos KPs que intercambiarán información proveniente de un sensor de luz.

Gráficamente:

Donde:

Ontología *LuminositySensor*: Representa de manera normalizada en formato JSON la información de un sensor de luz.

- **KP Productor:** KP ejecutado en una placa Arduino. Utilizará el API Arduino de SOFIA. Este KP tendrá conectado un sensor fotovoltaico a una de las entradas analógicas de la placa.

  El KP realizará una operación SSAP de JOIN para abrir una sesión con el SIB de la plataforma, y una vez conectado, periódicamente consultará el valor de la entrada analógica del sensor de luz y compondrá un mensaje SSAP de INSERT para enviar al SIB una instancia de la ontología *LuminositySensor* con dicho valor.

- **KP Consumidor:** Página HTML que se ejecutada sobre un navegador web. Utilizará el API Javascript de SOFIA.

  Este KP realizará una operación SSAP de JOIN para abrir una sesión con el SIB de la plataforma, y una vez conectado, enviará al SIB una operación SSAP de SUBSCRIBE, para ser notificado cuando el KP Productor envíe al SIB nuevas instancias de la ontología *LuminositySensor*.

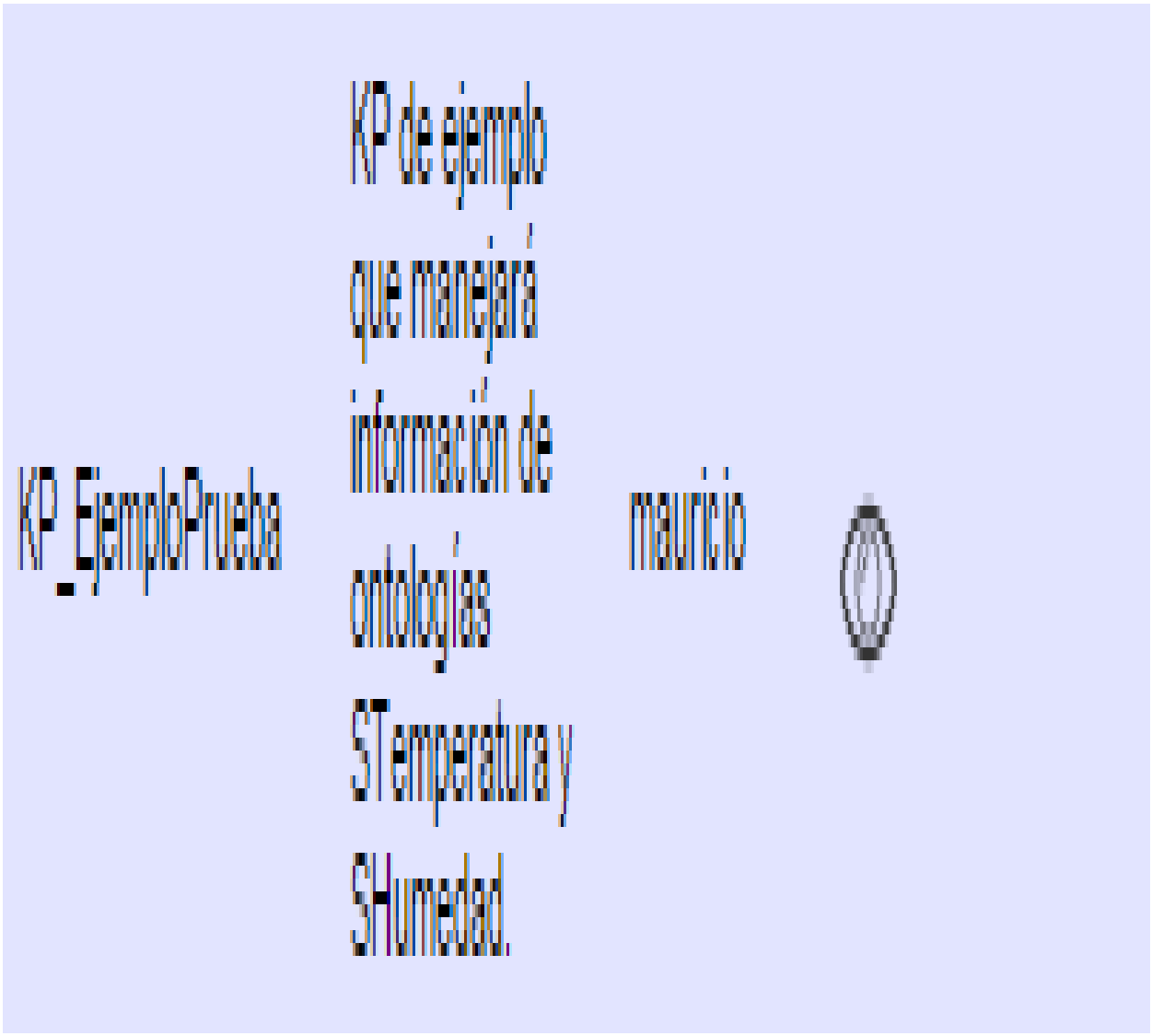
  Una vez suscrito, el KP utilizará las notificaciones del SIB para representar en una gráfica la evolución de la luminosidad.

A continuación vamos a construir paso por paso el escenario descrito:

Alta de usuario en la plataforma

Daremos de alta un único usuario, que será propietario de ambos KPs. Llamaremos a este usuario *sensorizacionLuminosidad***.**

Entramos en la plataforma y pulsamos **solicita un nuevo usuario:**



A continuación introducimos los datos del usuario en el formulario de alta:

Una vez dado de alta por esta vía, el usuario tendrá rol **ROL_USUARIO**, con el cual podrá crear sus propios KPs sobre ontologías sobre las que está suscrito, pero no será lo suficientemente autónomo para poder ejecutar este ejemplo práctico dado que se necesita crear un KP productos de información. Para poder obtener los privilegios oportunos, un usuario con **ROL_ADMINISTRADOR** deberá incrementar su nivel de control sobre la plataforma a **ROL_COLABORADOR** o **ROL_ADMINISTRADOR**.

En nuestro caso le daremos **ROL_COLABORADOR** como se indica en la **Guía de Uso de la Consola Web**:

De este modo, a partir de ahora, nuestro usuario *sensorizacionLuminosidad* será lo suficientemente autónomo sobre la plataforma para poner en marcha sus KPs.

Ontologización de la información

Para el escenario descrito, identificamos un único concepto: **LuminositySensor**, al que dotaremos de los siguientes atributos:

- **Identificador**: Identificación del Sensor String
- **Timestamp**: momento de la medición integer
- **Medida**: Valor en lúmenes de la medición integer
- **Unidad**: Unidad del valor medida (Lumen) String

Una vez identificadas estas propiedades, podemos construir el *JSONSchema* de la ontología que las describe:

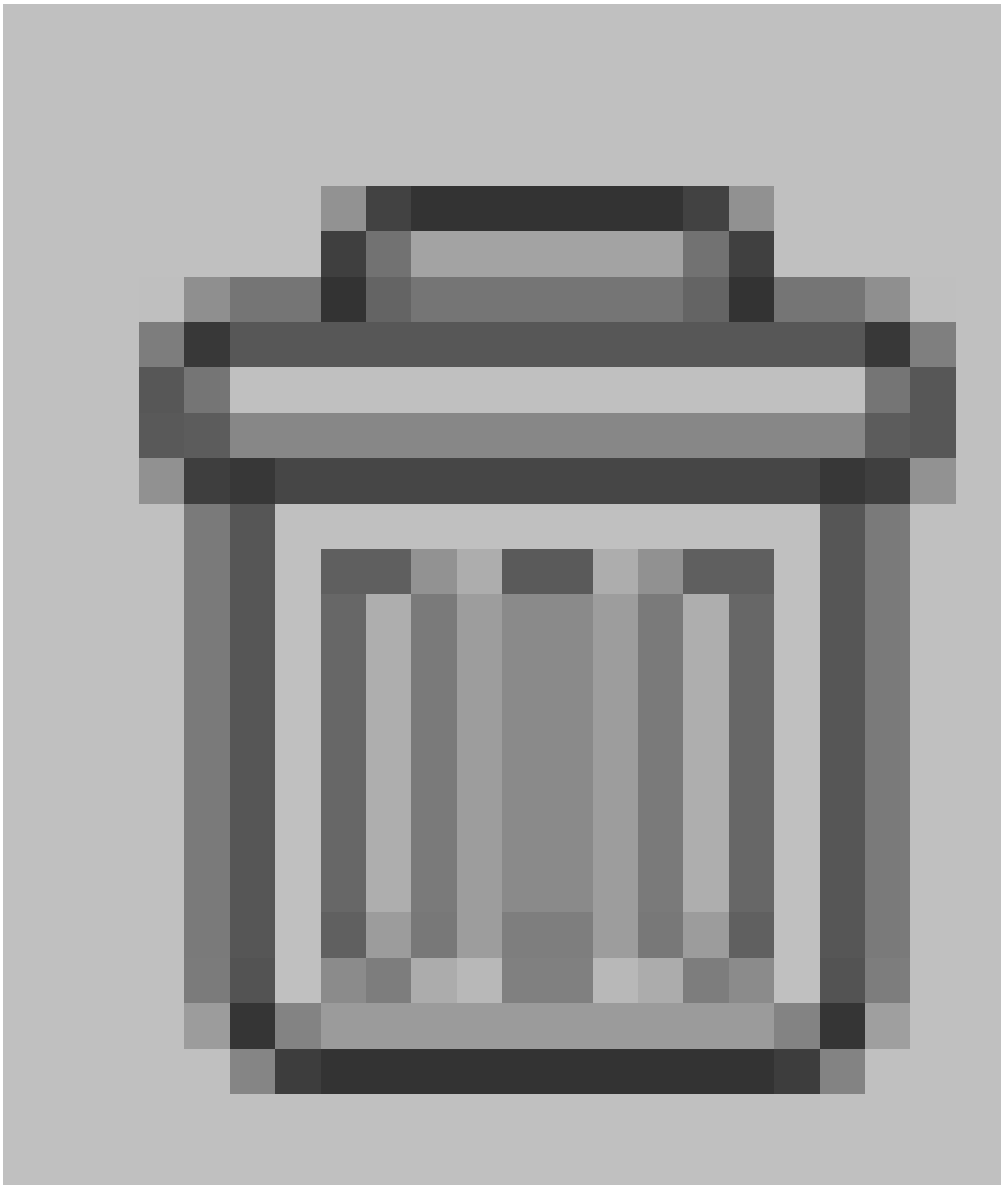
LuminositySensor.json

```
{
“$schema”: “http://json-schema.org/draft-03/schema#”,
“title”: “LuminositySensor Schema”,
“type”: “object”,
“properties”: {
“_id”: {
“type”: “object”,
“$ref”: “#/identificador”
},
” LuminositySensor ”: {
“type”: “string”,
“$ref”: “#/datos”
}
},
“identificador”: {
“title”: “id”,
“description”: “Id insertado del LuminositySensor ”,
“type”: “object”,
“properties”: {
“$oid”: {
“type”: “string”,
“required”: false
}
}
},
“datos”: {
“title”: “datos”,
“description”: “Info LuminositySensor ”,
“type”: “object”,
“properties”: {
“identificador”: {
“type”: “string”,
“required”: true
},
“timestamp”: {
“type”: “integer”,
“minimum”: 0,
“required”: true
},
“medida”: {
“type”: “number”,
“required”: true
},
“unidad”: {
“type”: “string”,
“required”: true
}
}
},
“additionalItems”: false
}
```

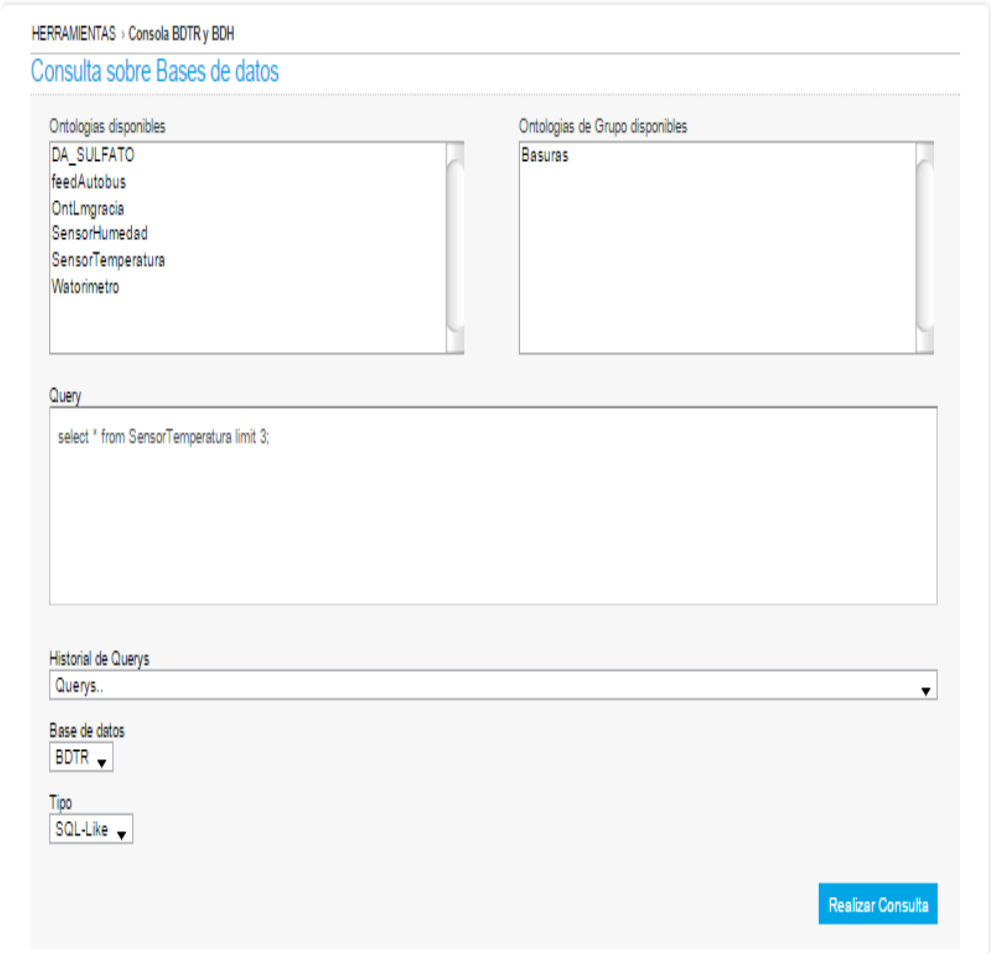
Y damos de alta la ontología en la plataforma en la sección **Gestión Ontologías** como se indica en la

Guía de Uso de la Consola Web.

Para ello nombramos la ontología con el nombre **LuminositySensor**, la marcamos como **Activa**, para que el SIB admita su utilización y dependiendo de si queremos que pueda ser utilizada o no por otros usuarios, la marcaremos Pública o no. Adicionalmente podremos darle una descripción:



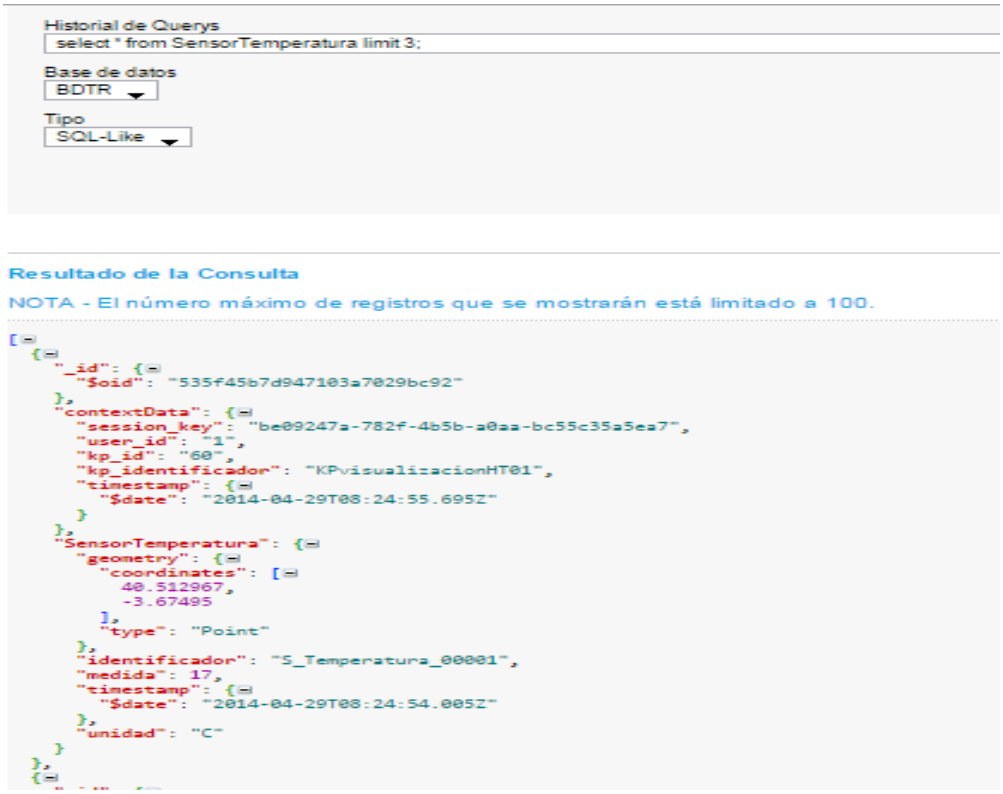
Una vez dada de alta se mostrará la siguiente pantalla y estará disponible para su futura administración en la sección **Gestión Ontologías** de la **Consola Web**:



Desarrollo de KP productor de información

En el desarrollo de nuestro KP deberíamos empezar por dar permisos de **INSERT** a nuestro usuario para la ontología a ***LuminositySensor*** como se indica en la **Guía de uso de la Consola Web**. En nuestro caso, el usuario propietario del KP es también propietario de la ontología que utilizará el KP, por lo que no es necesario dar permiso de INSERT al usuario, ya que tiene todos los permisos.

El siguiente paso es dar de alta el KP en la plataforma. Para ello en la sección **Gestión KPs** creamos un nuevo KP como se indica en la **Guía de uso de la Consola Web** con la siguiente información:



Donde:

Identificacion	Nombre del KP
Clave cifrado	Clave de cifrado XXTEA para comunicaciones cifradas con el SIB
Descripcion	Texto descriptivo del propósito del KP
Ontologias	Ontologías a utilizar por el KP
Metainformacion	Pares clave valor con información adicional del KP

Una vez dado de alta, quedará disponible para ser administrado en el futuro en la sección **Gestión KPs**

```
},
"contextData": {
  "session_key": "be89247a-782f-4b5b-a0aa-bc55c35a5ea7",
  "user_id": "1",
  "kp_id": "60",
  "kp_identificador": "KPvisualizacionHT01",
  "timestamp": {
    "$date": "2014-04-29T08:24:55.695Z"
  }
},
```


Lo siguiente en el desarrollo de nuestro KP productor de información es la programación del mismo. Para obtener más información de cómo desarrollar un KP se recomienda ver la **Guía de Apis SOFIA2**.

Al ser nuestro KP productor una aplicación que se ejecutará sobre una placa **Arduino** estará desarrollada en el lenguaje de Arduino.

manuals/desarrollador/./media/image58.jpeg

Arduino es un **microcontrolador** cuya lógica de programación gira en torno a dos funciones:

- **Setup():** Invocada una única vez al arrancar el microcontrolador. Su propósito es realizar tareas de configuración.
- **Loop():** Invocada iterativamente (cuando finaliza, se vuelve a invocar). Recoge la lógica de negocio del micro-

controlador.

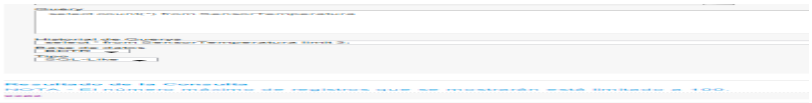
Bajo esta filosofía de programación, abriremos una **sesión al SIB** de la plataforma mediante el mensaje SSAP **JOIN**. Para esto utilizaremos el API Arduino. Este API nos proporciona las siguientes utilidades:

- **Conector KP MQTT** (KPMqtt.h) para establecer una conexión física con el SIB de la plataforma a través de la que enviar los mensajes SSAP.
- **API para la generación y parseo de mensajes SSAP** sin tener que construir los mensajes JSON directamente (SSAPMessageGenerator.h y SSAPBodyReturnMessage.h).

Con estas utilidades, el establecimiento de la conexión física por MQTT y la posterior conexión lógica mediante el mensaje SSAP JOIN se realizaría del siguiente modo:

Una vez conectados a la plataforma, el siguiente paso será la captación de datos y la ontologización de los mismos para insertarlos en el SIB mediante el correspondiente mensaje de INSERT.

La captación de datos de nuestro KP consiste en leer la entrada analógica del sensor fotovoltaico en cada iteración de la función Loop():



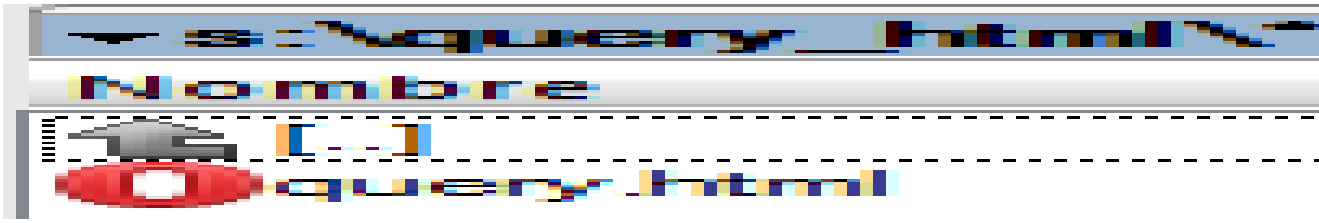
Una vez captados los datos, hay que proceder a convertirlos a la ontología **LuminositySensor** para enviarlos normalizados a la plataforma.

Aquí tendríamos dos opciones:

- Construir una clase *LuminositySensor* con los métodos get y set de sus atributos y una función toJson() que devuelva el JSON de la instancia.
- Construir el JSON directamente como un String concatenándole el valor de la medición.

Dada la sencillez de la ontología, optamos por la segunda opción:

Una vez construida la instancia de la ontología con el valor recogido, procedemos a enviarla a la plataforma mediante el correspondiente mensaje SSAP **INSERT**. Para ello volvemos a utilizar el API Arduino:



El código completo del KP productor puede verse aquí:

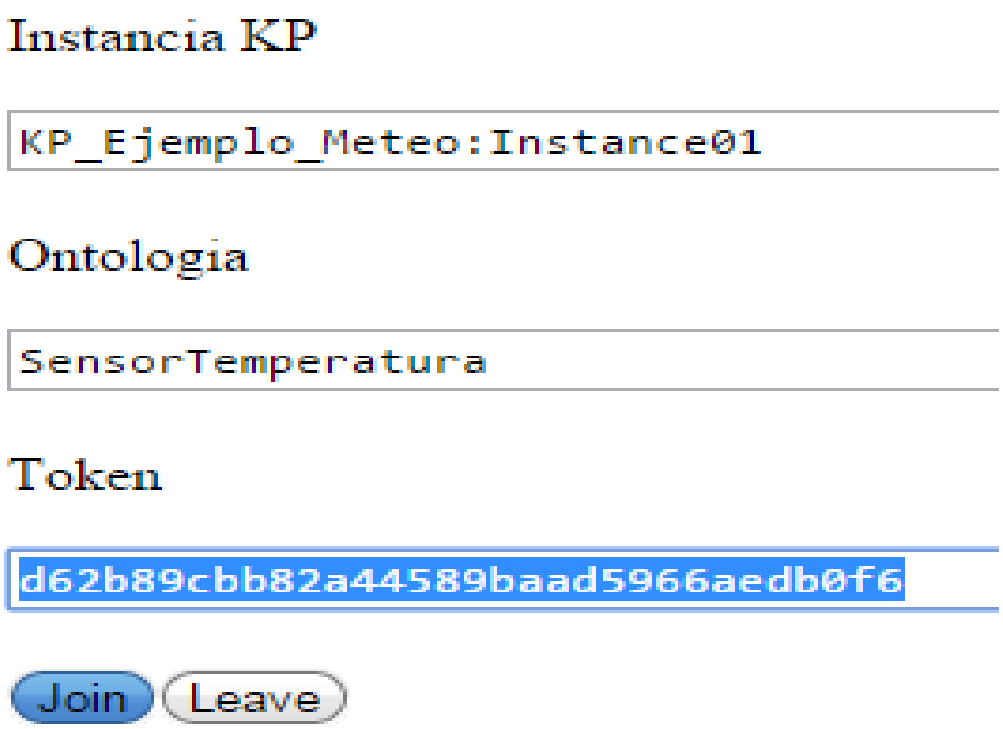
KP_LuminositySensor.ino

```
#include <SSAPMessageGenerator.h>
#include <SSAPBodyReturnMessage.h>
#include <SPI.h>
#include <PubSubClient.h>
#include <Ethernet.h>
#include <KPMqtt.h>
#include <LightSensor.h>
byte mac[] = { 0x00, 0xAA, 0xBB, 0xCC, 0xDE, 0x01 };
IPAddress ip( 192, 168, 10, 129 ); // My Ip
byte serverIp[] = { 192, 168, 10, 128 }; // Server Ip
boolean joined=false;
ConnectionConfig config;
KPMqtt kp;
SSAPMessageGenerator ssapGenerator;
int photoSensorPinIn = 0;//A0 input3
void setup() {
Serial.begin( 9600 );
config.setLocalMac(mac);
config.setLocalIp(&ip);
config.setServerIp(serverIp);
kp.setClientId("Arduino");
kp.setConnectionConfig(&config);
}
SSAPMessage joinMessage;
void loop() {
Serial.println("new Loop");
if(!joined){
join();
}
if(joined){
meterValues();
}
Serial.println("End of Loop");
delay(1000);
}
void join(){
if(!kp.isConnected()){
Serial.println("connect");
kp.connect();
}
Serial.println("Send join");
joinMessage=ssapGenerator.generateJoinMessage("sensorizacionLuminosidad", "slum2013", "ProductorLumi-
nosidad:prod01");
SSAPMessage joinResponse=kp.send(&joinMessage);
if(&joinResponse!=NULL){
char* bodyJoin=joinMessage.getBody();
delete[] bodyJoin;
char* responseBody=joinResponse.getBody();
SSAPBodyReturnMessage bodyMessage=SSAPBodyReturnMessage::fromJSonToSSAPMessage(responseBody);
if(bodyMessage.isOk()){
joined=true;
}else{
joined=false;
}
delete[] bodyMessage.getData();
delete[] bodyMessage.getError();
delete[] responseBody;
delete[] joinResponse.getSessionKey();
delete[] joinResponse.getMessageId();
```

Desarrollo de KP consumidor de información

Al igual que con el KP Productor, en el desarrollo de nuestro KP consumidor deberíamos empezar por dar permiso de **QUERY** a nuestro usuario para la ontología **LuminositySensor**. Pero de nuevo el usuario propietario del KP es también propietario de la ontología que utilizará, por lo que no es necesario dar tal permiso, ya que nuestro usuario tiene todos los permisos. En caso de utilizar otra ontología de la que el usuario no fuera propietario, un administrador debería dotar al usuario de tal permiso tal como se indicó en punto 3.3.1

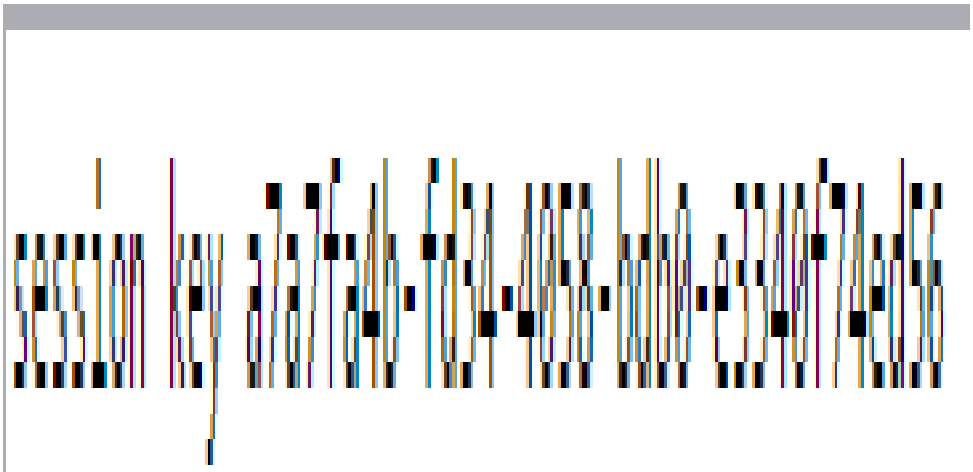
El siguiente paso es dar de alta el KP en la plataforma. Para ello en la sección **Gestión KPs** creamos un nuevo KP con la siguiente información:



Donde:

Identificación	Nombre del KP
Clave cifrado	Clave de cifrado XXTEA para comunicaciones cifradas con el SIB
Descripción	Texto descriptivo del propósito del KP
Ontologías	Ontologías a utilizar por el KP
Metainformación	Pares clave valor con información adicional del KP

Una vez dado de alta, quedará disponible para ser administrado en el futuro en la sección **Gestión KPs** junto al KP Productor que dimos de alta en el paso anterior:



Lo siguiente en el desarrollo de nuestro KP consumidor de información es la programación del mismo. Para obtener más información de cómo desarrollar un KP se recomienda ver la **Guía de Apis SOFIA2**.

Al ser un KP que se ejecutará en un navegador, estará desarrollado en **HTML** y **Javascript**, por lo que utilizaremos el API JavaScript SOFIA. Este API nos proporciona la abstracción necesaria para interactuar con el SIB de SOFIA mediante funciones para todas las operaciones SSAP.

- **join(user, pass, instance, joinResponse)**: Envía al SIB un mensaje SSAP de JOIN con el usuario/password/instancia indicado por parámetros. Recibe la respuesta en la función callback pasada en el argumento joinResponse.
- **leave(leaveResponse)**: Envía al SIB un mensaje SSAP de LEAVE. Recibe la respuesta en la función callback pasada en el argumento leaveResponse.
- **insert(data, ontology, insertResponse)**: Envía al SIB un mensaje SSAP de INSERT con los datos y ontología pasados en los respectivos argumentos. Recibe la respuesta en la función callback pasada en el argumento insertResponse
- **query(query, ontology, queryResponse)**: Envía al SIB un mensaje SSAP de QUERY para la condición y ontología indicados en los respectivos parámetros. Recibe la respuesta en la función callback pasada en el argumento queryResponse.
- **subscribe(suscription, ontology, refresh)**: Envía al SIB un mensaje SSAP SUBSCRIBE para la condición, ontología y tiempo de refresco indicado en los respectivos argumentos.
- **unsubscribe(querySubs, unsubscribeResponse, unsubscribeMessages)**: Envia al SIB un mensaje SSAP UNSUBSCRIBE para la condición indicada en el parámetro querySubs. Recibe la respuesta en la función callback pasada en el argumento unsubscribeResponse. Si se produce algún error, se notificará a través de la función callback pasada en el argumento unsubscribeMessages.

Nuestro KP HTML se compondrá de:

- Un formulario para hacer JOIN/LEAVE al SIB
  - Textfield nombre usuario.
  - Textfield password usuario.
  - Textfield instancia KP.
  - Botón JOIN.
  - Botón LEAVE.
- Un formulario para hacer SUBSCRIBE/UNSUBSCRIBE sobre la ontología *LuminositySensor*.
  - Textfield condición de suscripción.
  - Textfield periodo de refresco.
  - Botón SUBSCRIBE.
  - Botón UNSUBSCRIBE.
- Una gráfica que mostrará la evolución de los datos de luminosidad que se reciban como notificaciones de la suscripción.

La conexión/desconexión al SIB, como hemos comentado, se realiza con los botones mostrados en la imagen anterior. El código html es el siguiente:

Query! Clear


```
{
   "_id": {
      "$oid": "535f45b7d947103a7029bc92"
   },
   "contextData": {
      "session_key": "be09247a-782f-4b5b-a0aa-bc55c35a5ea7",
      "user_id": "1",
      "kp_id": "60",
      "kp_identificador": "KPvisualizacionHT01",
      "timestamp": {
         "$date": "2014-04-29T08:24:55.695Z"
      }
   },
   "SensorTemperatura": {
      "geometry": {
         "coordinates": [
            40.512967,
            -3.67495
         ],
         "type": "Point"
      },
      "identificador": "S_Temperatura_00001",
      "medida": 17,
      "timestamp": {
         "$date": "2014-04-29T08:24:54.005Z"
      },
      "unidad": "C"
   }
}
```

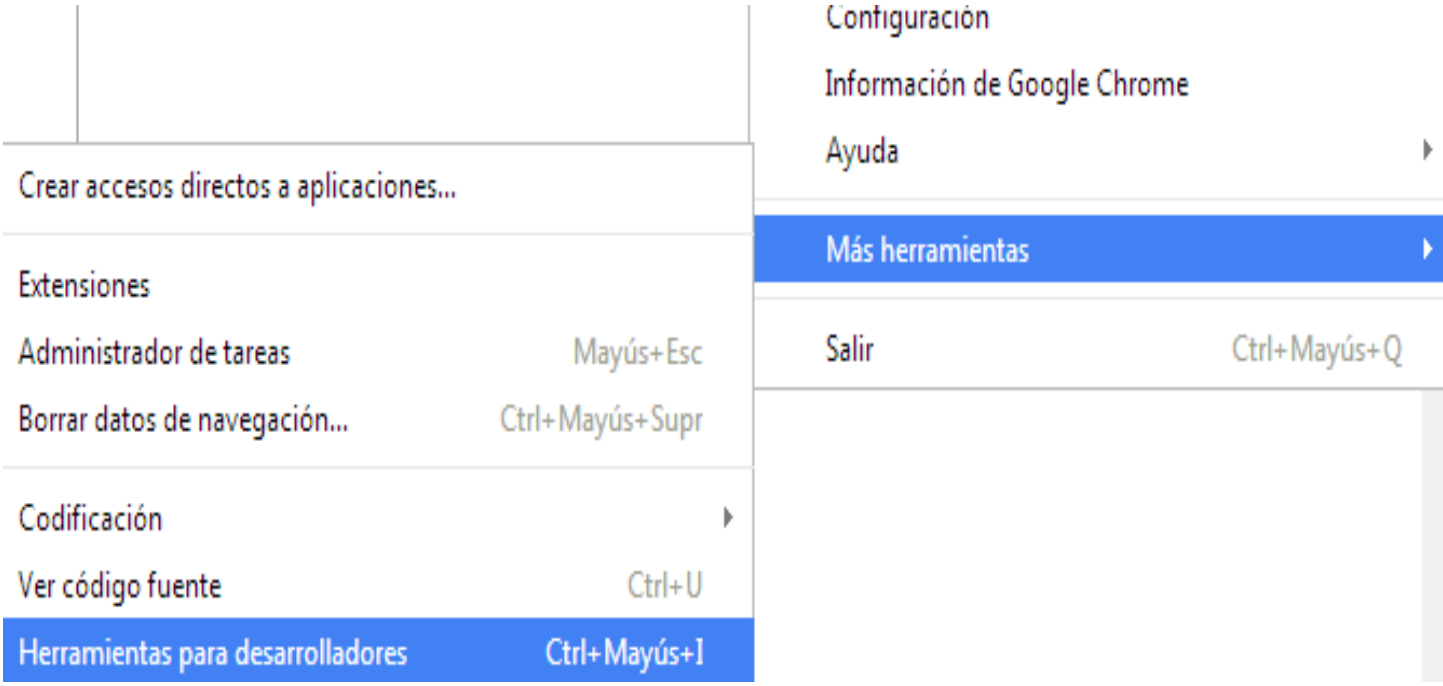

A continuación mostramos las funciones **conectarSIB** y **desconectarSIB** invocadas por tales botones y que hacen uso del API JavaScript SOFIA:

```
select * from SensorTemperatura where SensorTemperatura.medida>17
```

Query! Clear


```
      "user_id": "1",
      "kp_id": "60",
      "kp_identificador": "KPvisualizacionHT01",
      "timestamp": {
         "$date": "2014-04-29T08:25:07.684Z"
      }
   },
   "SensorTemperatura": {
      "geometry": {
         "coordinates": [
            40.512274,
            -3.675679
         ],
         "type": "Point"
      },
      "identificador": "S_Temperatura_00001",
      "medida": 29,
      "timestamp": {
         "$date": "2014-04-29T08:25:06.002Z"
      },
      "unidad": "C"
   }
}
```


La suscripción a la información insertada en el SIB por el KP productor la hacemos de igual modo con un formulario de suscripción. El código html del formulario es el siguiente:



Este formulario tiene una combo para seleccionar si nos suscribiremos a un valor de luminosidad mayor, menor o igual al un valor indicado en otro campo. Además incluye un nuevo campo para indicar el periodo de refresco de la suscripción. Y los botones para lanzar tanto la suscripción como la desuscripción.

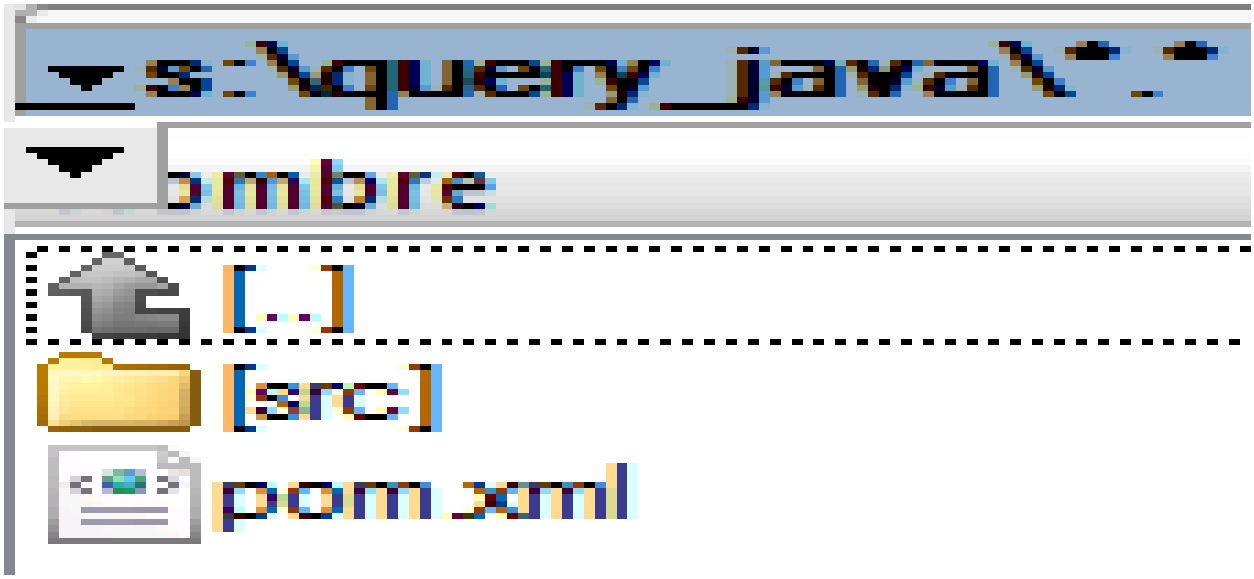
A continuación mostramos las funciones **suscribirSIB()** y **desuscribirSIB()** invocadas por tales botones:

Finalmente, una vez suscritos a la información falta implementar la recepción de las notificaciones desde el SIB para poder explotar tal información.

Para esto, el API JavaScript nos obliga a implementar la siguiente función:

function indicationForSubscription(ssapMessageJson, sourceQuery)

En ella recibimos un mensaje SSAP de tipo INDICATION, en cuyo atributo **body** tendremos la instancia de la ontología a la que nos suscrito y que ha cumplido los criterios de la query. Por lo que lo único que tendremos que hacer es extraer tal información del campo body y procesarla para pintarla en la gráfica:



El código completo del KP consumidor puede verse aquí:

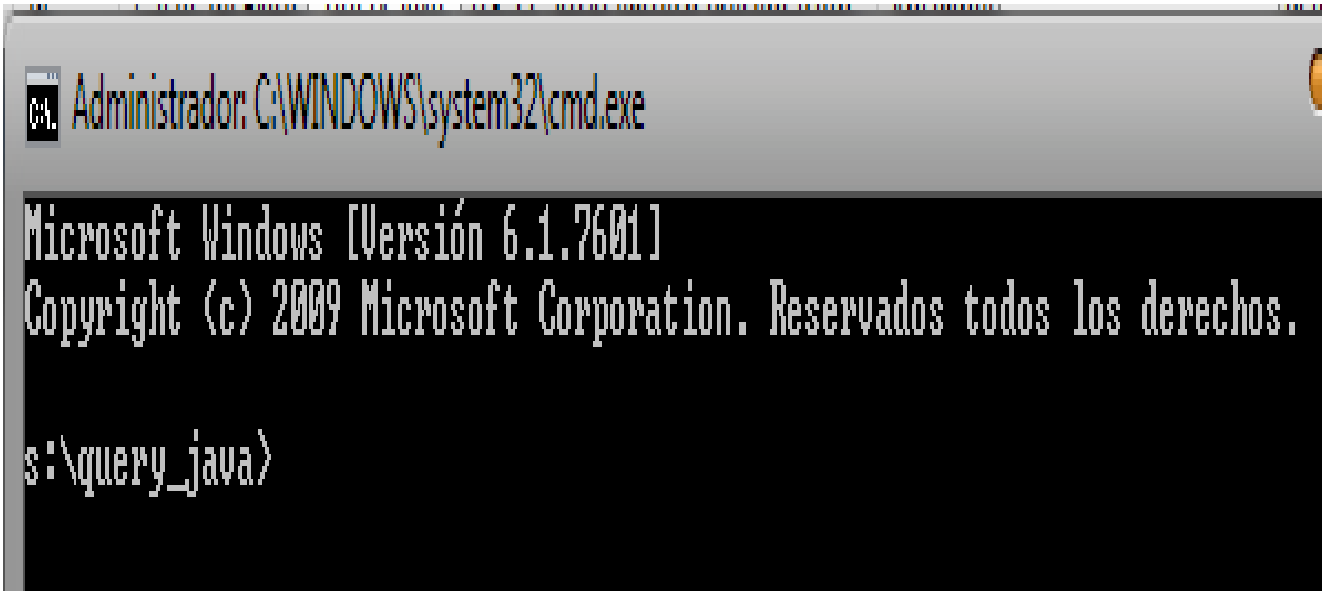
KP_LuminosityJavascript.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.
dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>KP Consumidor</title>
<link rel="stylesheet" type="text/css" href="styles/standard.css" />
<script src="jquery.min.js" type="text/javascript">;</script>
<script type="text/javascript"> var pathToDwrServlet = 'http://localhost:10000/sib-web/dwr';</script>
<script type='text/javascript' src='http://localhost:10000/sib-web/dwr/engine.js'></script>
<script type='text/javascript' src='http://localhost:10000/sib-web/dwr/util.js'></script>
<script type='text/javascript' src='http://localhost:10000/sib-web/dwr/interface/GatewayDWR.js'></script>
<script type='text/javascript' src="kp-core.js">;</script>
<script type='text/javascript' src="dygraph-combined.js">;</script>
<script type="text/javascript">
var datosTH = [];
var grafica = null, map = null;
$(function(){
dwr.engine.setActiveReverseAjax(true);
dwr.engine.setErrorHandler(errorHandler);
dwr.engine.setTimeout(0);
initGrafica();
});
function errorHandler(message, ex){
}
function initGrafica() {
grafica = new Dygraph(
document.getElementById("grafica"),
[[0,0]],
{
title: 'Luminosity / Real Time',
legend: 'always',
labels: ['Time', 'Luminosity'],
ylabel: 'Luminosity (lm)',
yAxisLabelWidth: 50,
digitsAfterDecimal : 0,
drawXGrid: false,
drawYGrid: false,
}
);
}
function conectarSIB(user, pass, inst) {
join(user, pass, inst,function(mensajeSSAP){
if(mensajeSSAP != null && mensajeSSAP.body.data != null && mensajeSSAP.body.ok == true){
$('#info').text("Conectado al sib con sessionkey: "+mensajeSSAP.sessionKey).show();
}else{
$("#info").text("Error conectando del sib").show();
}
});
}
function desconectarSIB() {
leave(function(mensajeSSAP){
if(mensajeSSAP != null && mensajeSSAP.body.data != null && mensajeSSAP.body.ok == true){
$("#info").text("Desconectado del sib").show();
}else{
$("#info").text("Error desconectando del sib").show();
}
});
}
```

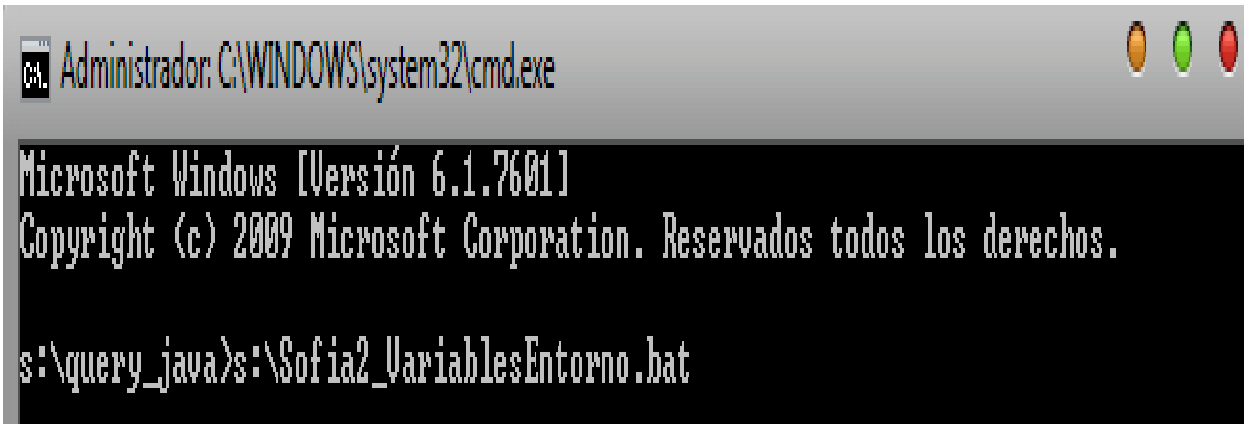
Ejecución

Una vez realizados todos los pasos anteriores, lo siguiente es comprobar que todo funciona correctamente. Para ello:

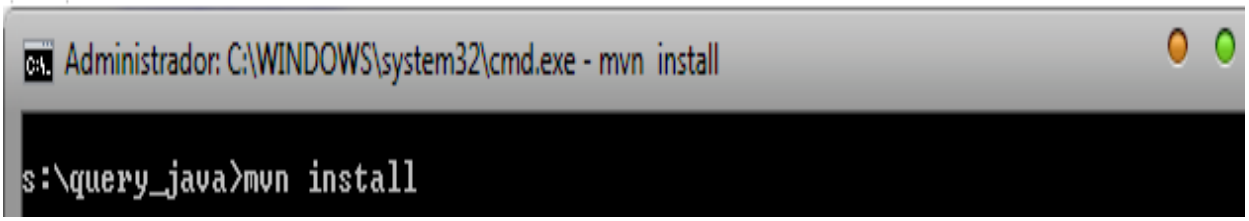
- Arrancar Plataforma SOFIA
- Cargar el programa del KP Productor en Arduino y arrancarlo.
  - Este KP auto-conecta al SIB al arrancar y empieza a enviar datos No es necesario hacer nada más.



- Abrir KP Consumidor con navegador web
  - Conectar al SIB: Basta con pulsar el botón **ConnectSIB**



- Suscribirse: Basta con pulsar el botón **Start**



- Comprobar que en la gráfica se muestran los datos enviados por el KP productor. El efecto será automático en el momento que se lance la suscripción si el KP Productor está enviando datos al SIB.

Taller IoT

Introducción

El objetivo de este taller es la realización de un ejemplo real sobre el que poder evaluar las capacidades de la plataforma Sofia2.

Para ello vamos a simular un edificio que dispone de varias plantas, y tres dispositivos para la lectura de consumo energético, temperatura y humedad en cada una de ellas, para terminar, crearemos un cuadro de mando para visualizar esta información y publicaremos los datos en un API para ser consumido de manera sencilla por cualquier aplicación.

Modelo de datos

El Modelo

El pilar sobre el que se sustenta un proyecto IoT es su modelo de datos, este debe contener toda la información relevante, tanto para un uso inmediato como para posteriores análisis de la información.

Es importante que los dispositivos no tengan que enviar información irrelevante que únicamente genere un coste en las comunicaciones.

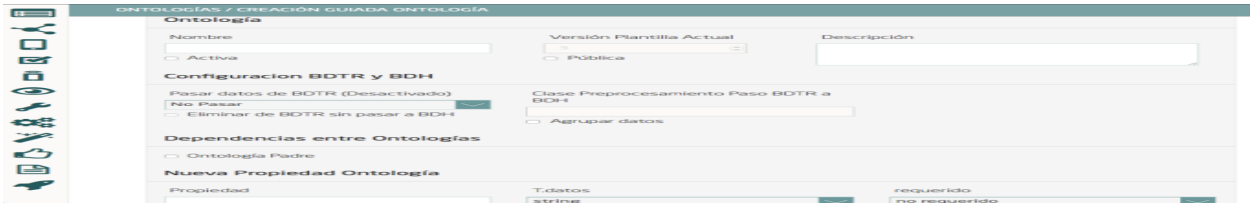
En la generación de un modelo equilibrado radica la complejidad del diseño del modelo de datos.

Ontología

En Sofia2 el modelo de datos recibe el nombre de Ontología, podemos definir ésta de una manera muy superficial como un esquema JSON que definirá de forma explícita los datos que almacenará.

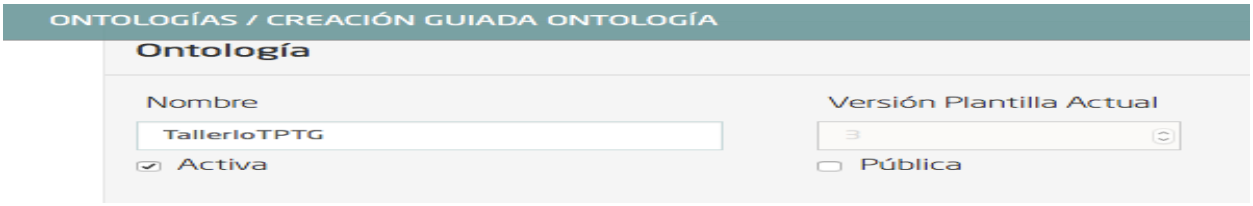
La consola de administración centralizada de sofia2 http://sofia2-analytic.cloudapp.net/console/ dispone de varios métodos para crear una ontología (editor gráfico, modo texto, asistente y desde origen de datos).

Vamos a utilizar la Creación Guiada de Ontología.



Lo primero que tenemos que hacer es darle un nombre a nuestra ontología, la llamaremos TallerIoT<Nuestras iniciales Nombre, 1 Apellido 2 Apellido>.

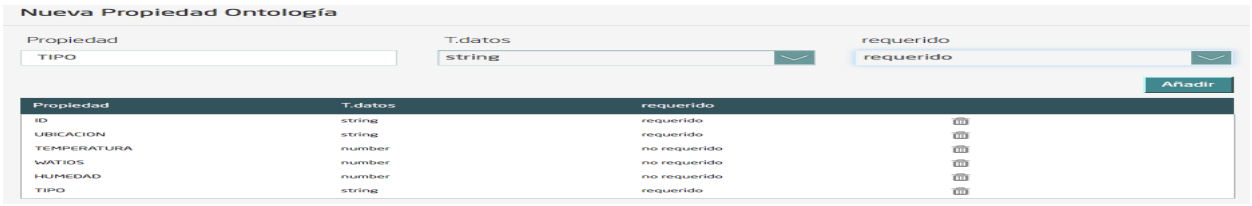
Marcamos la ontología como activa y es una buena práctica indicar la descripción de la finalidad que tiene nuestra ontología.



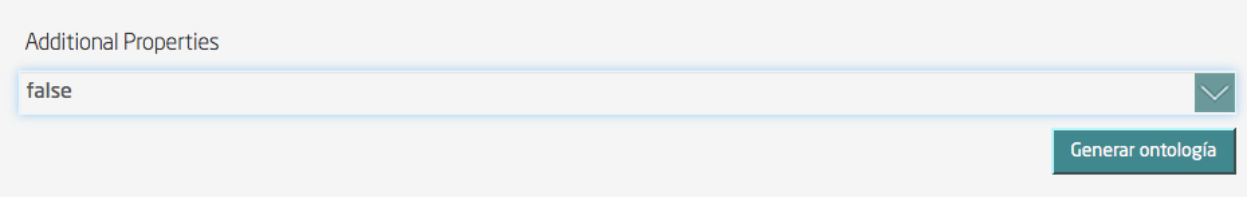
Podemos obviar los detales de Configuración BDTR y BDH y dependencia entre ontologías, pues no tiene relevancia para este taller.

Ahora vamos a añadir los campos de nuestro modelo de datos, vamos a trabajar con un modelo muy sencillo que contendrá la siguiente información:

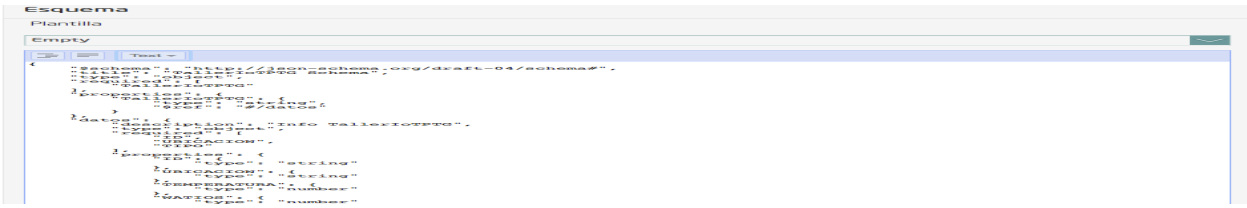
ID	String	requerido
UBICACION	String	requerido
TEMPERATURA	Number	no requerido
WATIOS	Number	no requerido
HUMEDAD	Number	no requerido
TIPO	String	requerido



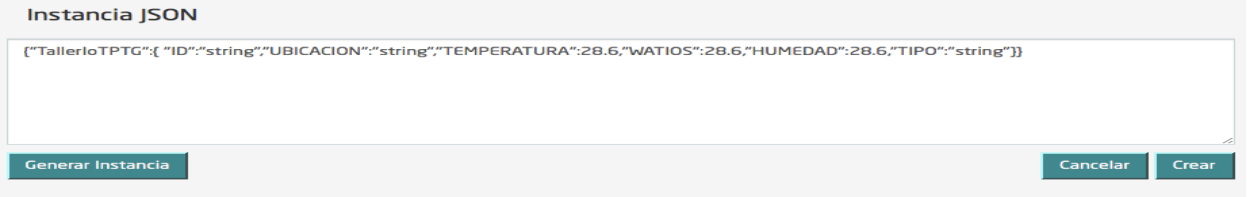
Establecemos la propiedad Additional Properties a false para evitar que nuestra ontología pueda contener otro tipo de información. Y pulsamos el botón generar ontología.



En el apartado Esquema, nos aparecerá la definición del JSON-Schema que define nuestra ontología y que tiene que cumplir todas las instancias de ontología que utilicemos.



Si pulsamos el botón Generar Instancia nos mostrará un ejemplo de una instancia de ontología.

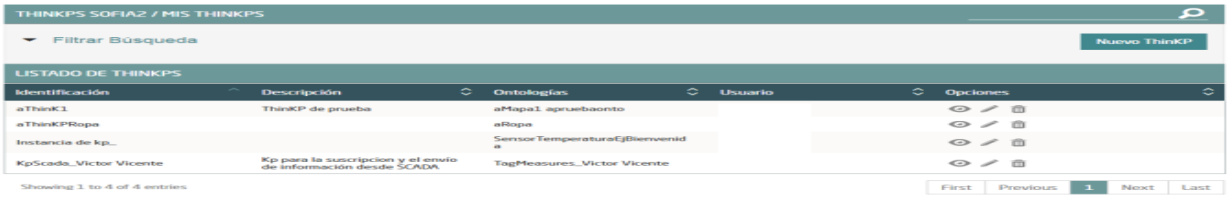


Por último, pulsamos el botón Crear.

ThinKP

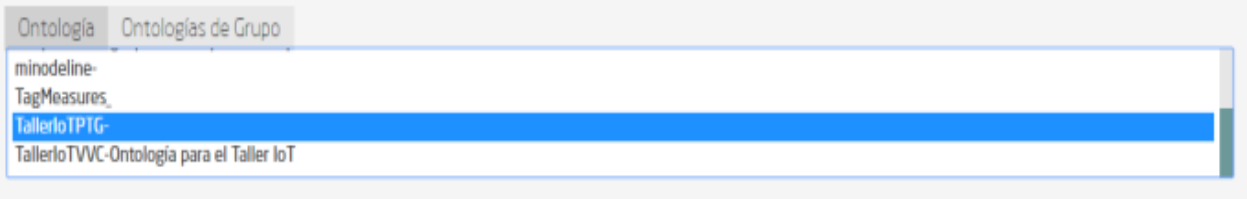
Una vez que hemos definido el modelo de datos y lo hemos plasmado en una ontología, tenemos que crear el ThinKP, la configuración lógica de los dispositivos que van a interactuar con nuestra ontología.

Para ello accedemos al menú Mis ThinKPs y pulsamos sobre el botón Nuevo ThinKP.



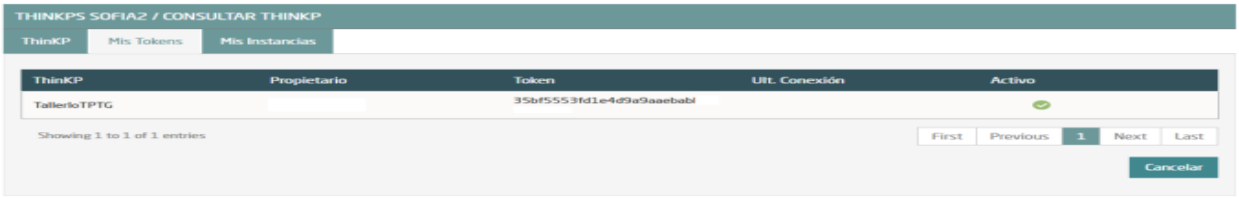
Le damos un nombre a nuestro ThinKP, le llamaremos TallerIoT<Nuestras iniciales Nombre, 1 Apellido 2 Apellido>.

Podemos darle una descripción. Y debemos de seleccionar la ontología que hemos creado en el punto 3. Que debe llamarse TallerIoT<Nuestras iniciales Nombre, 1 Apellido 2 Apellido>

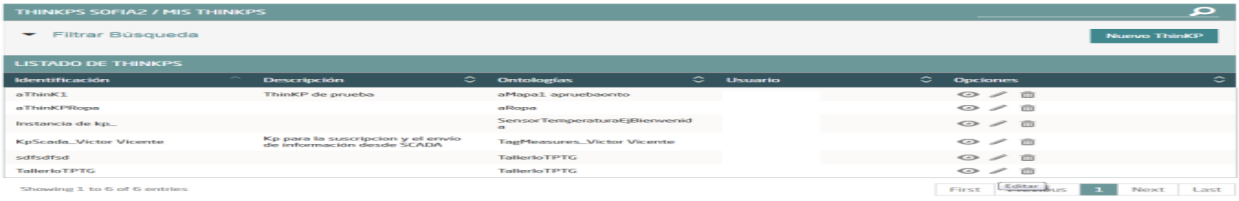


Una vez dada esta información podemos pulsar el botón Crear, nos aparecerá la pantalla de detalle del ThinKP.

Siempre podremos acceder a nuestros Token mediante la pestaña Mis Tokens.



Seleccionamos el icono de edición de nuestro ThinKP (lápiz).

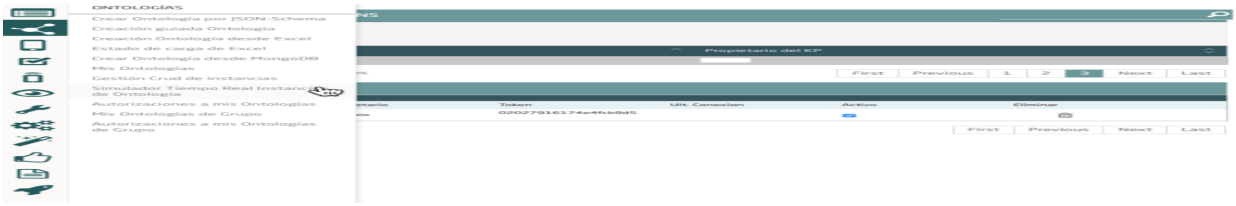


Seleccionamos la pestaña Mis Tokens. Nos aparecerá un listado con los Tokens que tiene asignados y las opciones para activar/desactivar, dar de baja y añadir nuevos Tokens.

Simular datos de entrada

Configuración del simulador

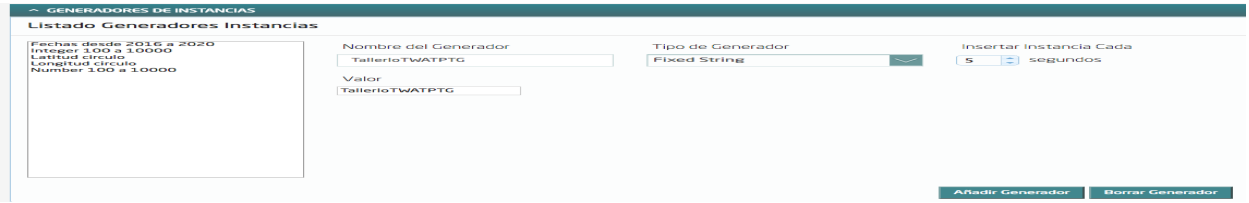
Puesto que no tenemos un dispositivo real que inserte información en nuestra ontología, vamos a hacer uso de las herramientas de Sofia2 para insertar datos simulados, para ello accedemos al menú Simulador Tiempo Real Instancias de Ontología.



Pulsamos el botón Crear Simulador, lo primero que vamos a crear son los Generadores de Instancias que vamos a usar, un Generador de Instancias, es una definición de datos de prueba.

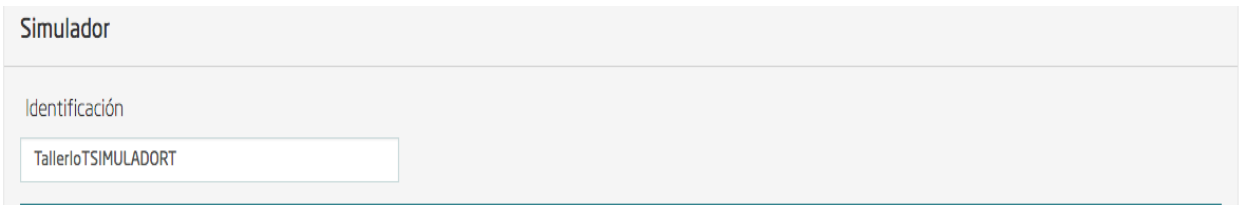
Vamos a crear los siguientes generadores:

- Primero el generador del campo ID para nuestro supuesto Watorimetro, como nombre le ponemos TallerIoTWAT<Nuestras iniciales Nombre, 1 Apellido 2 Apellido>, como tipo de Generador Fixed String y como valor, el mismo que el nombre que le hemos dado.

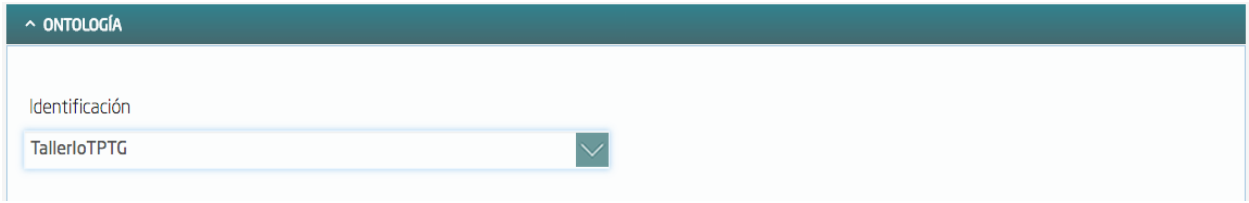


- Ahora el generador del campo ID para nuestro supuesto Termostato, como nombre le ponemos TallerIoTTERMOSTATO<Nuestras iniciales Nombre, 1 Apellido 2 Apellido>, como tipo de Generador Fixed String y como valor el mismo que el nombre que le hemos dado.
- Para terminar con el ID, el generador del campo ID para nuestro supuesto Medidor de Humedad, como nombre le ponemos TallerIoTH<Nuestras iniciales Nombre, 1 Apellido 2 Apellido>, como tipo de Generador Fixed String y como valor el mismo que el nombre que le hemos dado.
- Como generador de valores vamos a crear un único generador que utilizaremos tanto para simular la temperatura, la humedad y los watios consumidos, lo llamaremos TallerIoTVALOR<Nuestras iniciales Nombre, 1 Apellido 2 Apellido> como tipo de Generador Random Number, valores desde 1 hasta 100 y decimales de precisión 2.
- Para la ubicación vamos a crear un generador de tipo Random String, con el listado de palabras HALL, PB, P1, P2, P3, S1 y S2, simulando los pisos de un edificio. Y le vamos a llamar TallerIoTUBICACION<Nuestras iniciales Nombre, 1 Apellido 2 Apellido>
- Por último, vamos a crear los tipos de generador para el campo TIPO, que serán de tipo Fixed String y se llamarán TallerIoTTIPOH<Nuestras iniciales Nombre, 1 Apellido 2 Apellido> y valor HUMEDAD, TallerIoTTIPOT<Nuestras iniciales Nombre, 1 Apellido 2 Apellido> y valor TEMPERATURA y TallerIoTTIPOW<Nuestras iniciales Nombre, 1 Apellido 2 Apellido> y valor WATIOS.

Una vez definidos los Generadores vamos a crear tres simuladores, el medidor de termperatura, el de humedad y el de watios, para ello en el campo identificación ponemos el nombre TallerIoTSIMULADORT<Nuestras iniciales Nombre, 1 Apellido 2 Apellido>, TallerIoTSIMULADORH<Nuestras iniciales Nombre, 1 Apellido 2 Apellido> y TallerIoTSIMULADORW<Nuestras iniciales Nombre, 1 Apellido 2 Apellido>



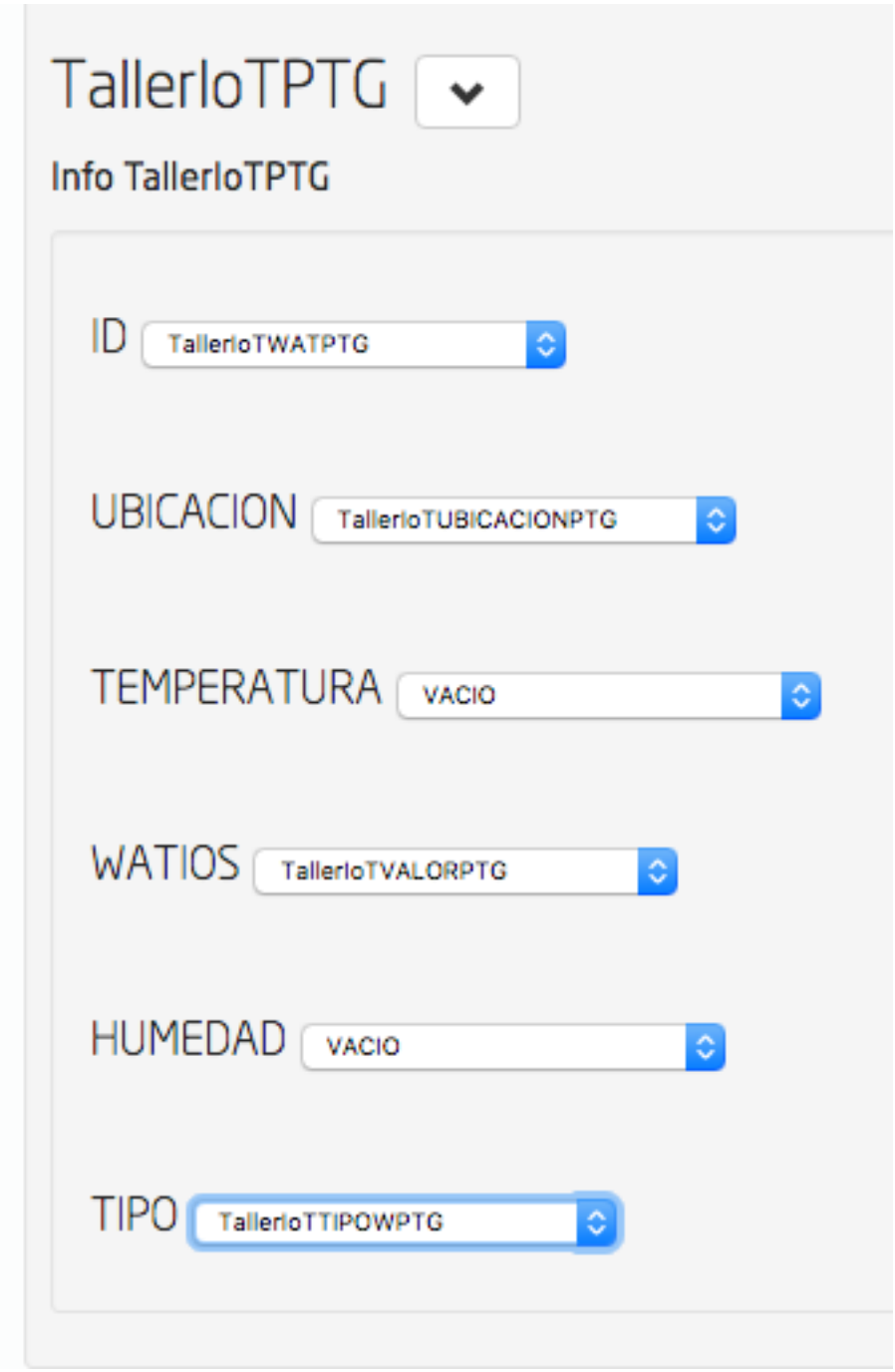
En la pestaña de ontología, seleccionamos nuestra ontología TallerIoT<Nuestras iniciales Nombre, 1 Apellido 2 Apellido>



Creamos la configuración para el Simulador de Temperatura.

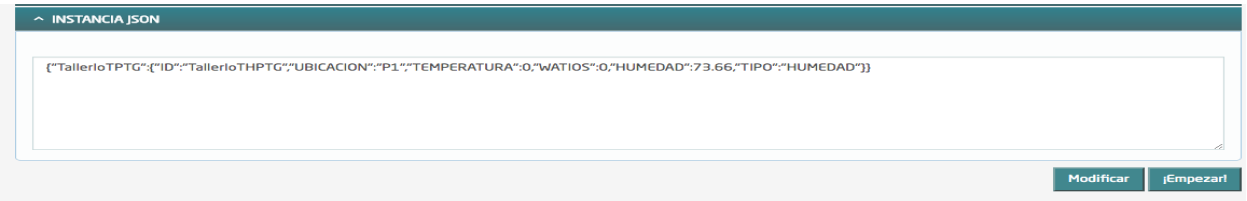
Creamos la configuración para el Simulador de humedad

Creamos la configuración para el simulador del Watorimetro.

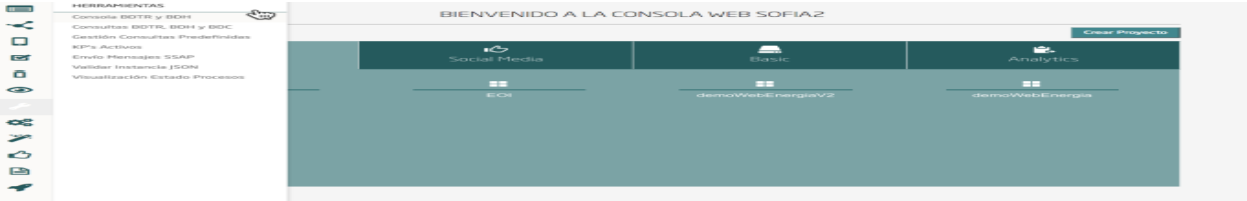


Ejecución de Simulador

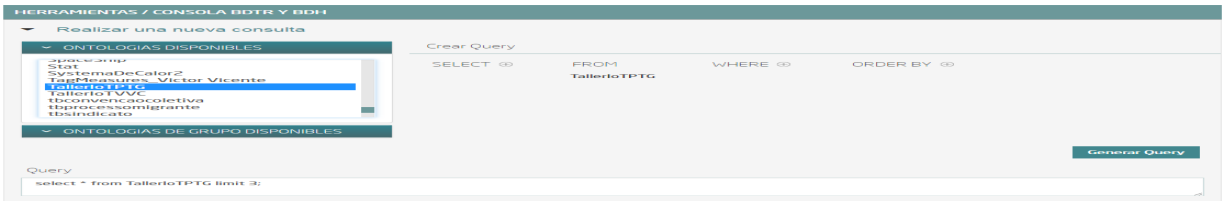
Abrimos tres nuevas pestañas en el explorador, accedemos en cada una de ellas a uno de los tres simuladores y pulsamos el botón ¡Empezar! en cada uno de ellos.



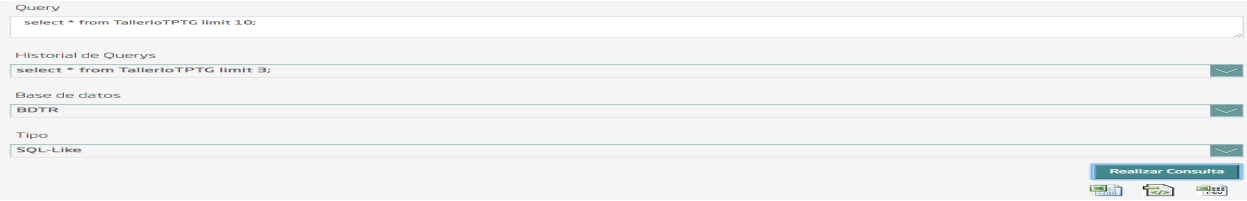
Abrimos una cuarta pestaña y accedemos a la opción del menú Consola BDTR y BDH.



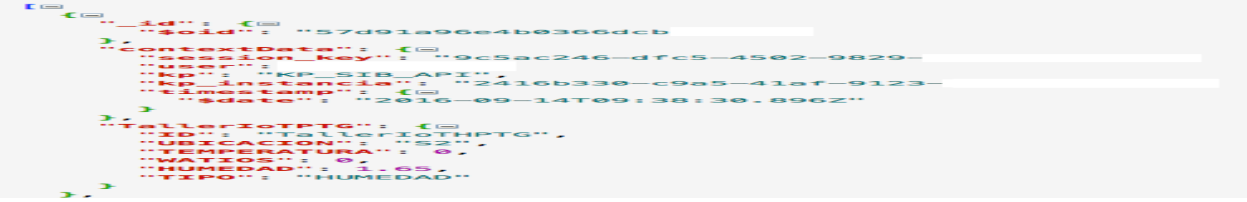
Seleccionamos nuestra ontología.



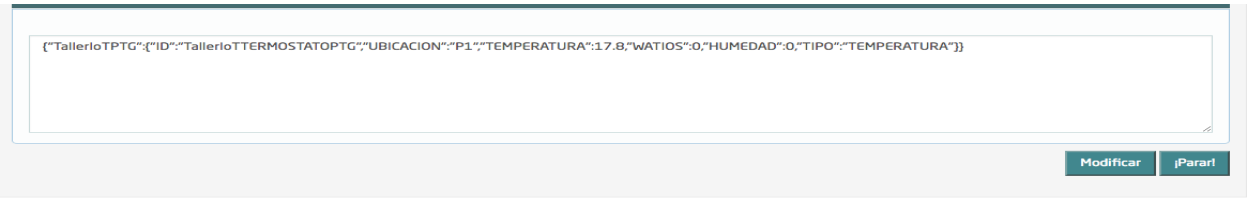
Nos aparecerá una consulta por defecto, para realizar las consultas sobre BDTR usando el lenguaje SQL-Like.



Si pulsamos el botón realizar consulta, nos devolverá los datos contenidos en la Base de Datos. Debemos de ser conscientes que la plataforma por configuración nos devolverá como mucho 100 Registros.



Detenemos nuestros tres simuladores pulsando sobre el botón ¡Parar!.



Cuadro de mando

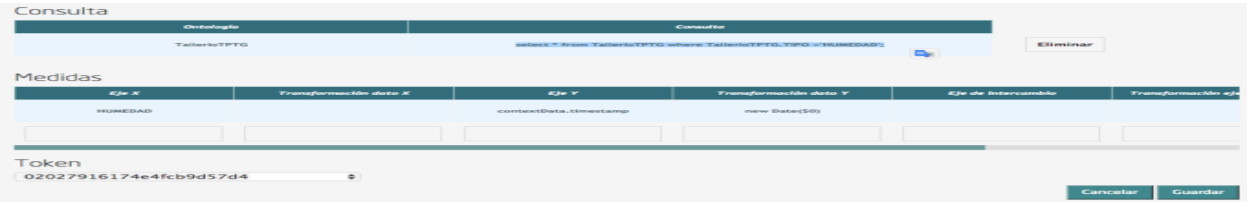
El siguiente paso es crear un cuadro de mando usando las capacidades de presentación gráfica de la plataforma.

Crear Gadget

Accedemos a la opción de menú Mis Gadgets y pulsamos sobre Crear Gadget, seleccionamos crear Gadget y el tipo Columna.

Como nombre le asignamos TallerIoTGADGETH<Nuestras iniciales Nombre, 1 Apellido 2 Apellido> y seleccionamos el ThinKP que creamos.

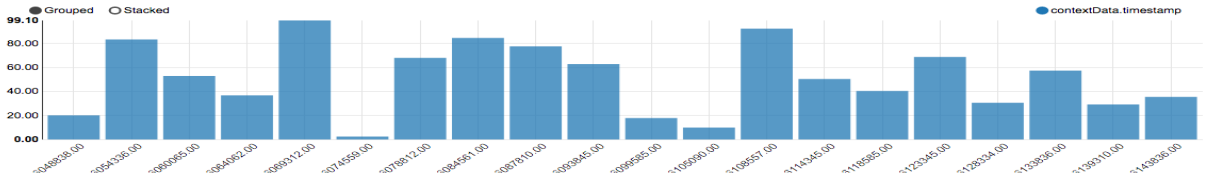
Seleccionamos la pestaña Obtener datos por query, usando la sentencia select * from TallerIoTPTG where TallerIoTPTG.TIPO =’HUMEDAD’; y como medidas Para el Eje X contextData.timestamp, este último debemos transformarlo usando new Date($0) y para el Eje Y HUMEDAD.



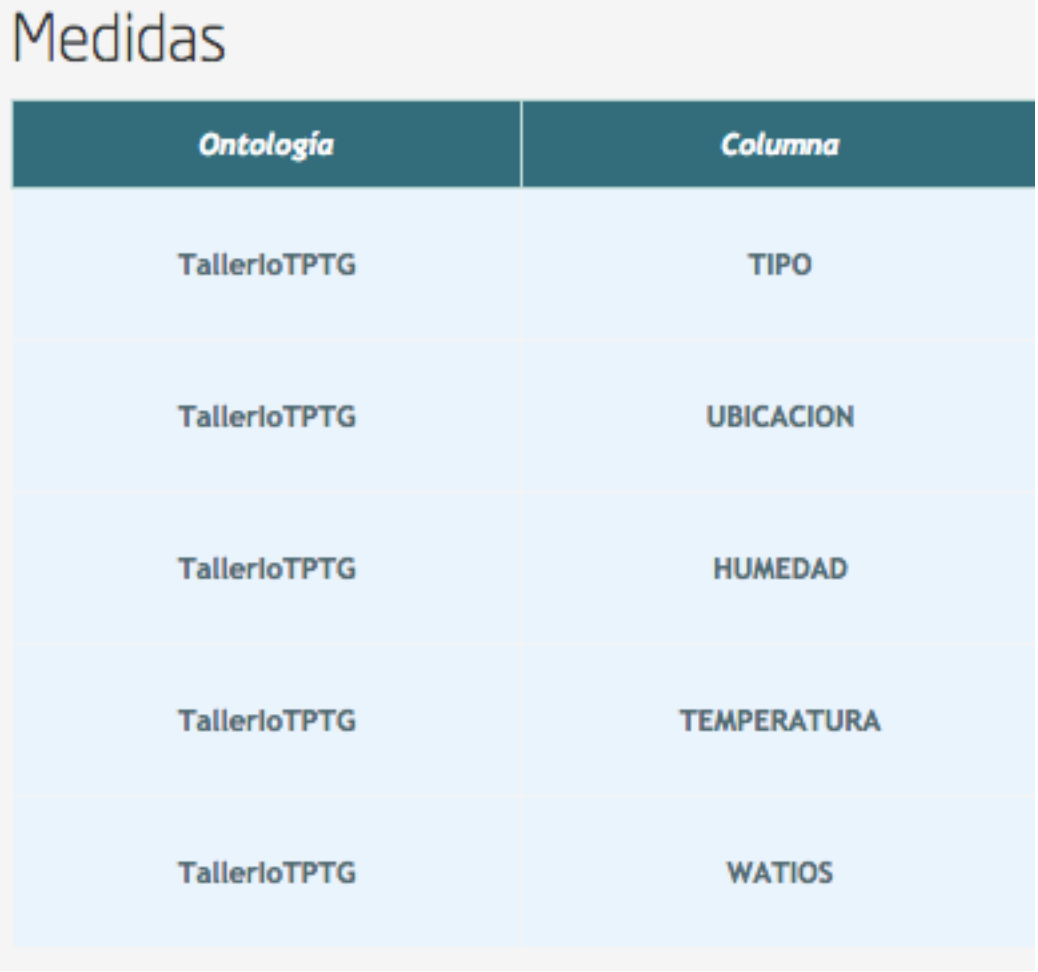
Haremos lo mismo para crear el gráfico de consumo de Watios, al que llamaremos TallerIoTGADGETW<Nuestras iniciales Nombre, 1 Apellido 2 Apellido> en este caso con la consulta select * from TallerIoTPTG where TallerIoTPTG.TIPO =’WATIOS’ y en el Eje Y WATIOS.

Y para crear el gráfico de temperatura, al que llamaremos TallerIoTGADGETT<Nuestras iniciales Nombre, 1 Apellido 2 Apellido> en este caso con la consulta select * from TallerIoTPTG where TallerIoTPTG.TIPO =’TEMPERATURA’ y en el Eje Y TEMPERATURA.

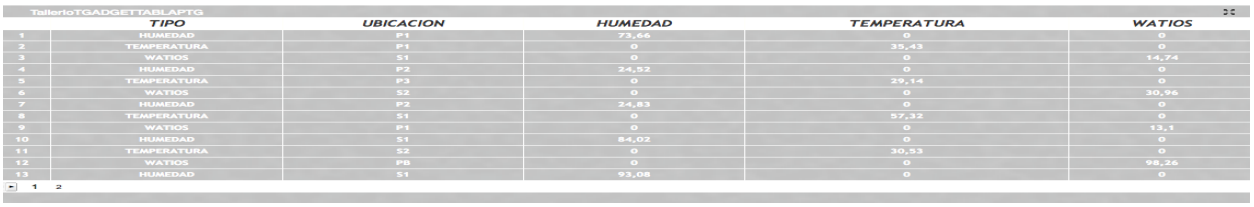
Por cada uno de los Gadget anteriores, creará un gráfico como el siguiente:



Por último, crearemos un Gadget de tipo tabla, le llamaremos TallerIoTGADGETTABLA<Nuestras iniciales Nombre, 1 Apellido 2 Apellido>, seleccionaremos la pestaña Obtener datos en directo y añadiremos las siguientes columnas:



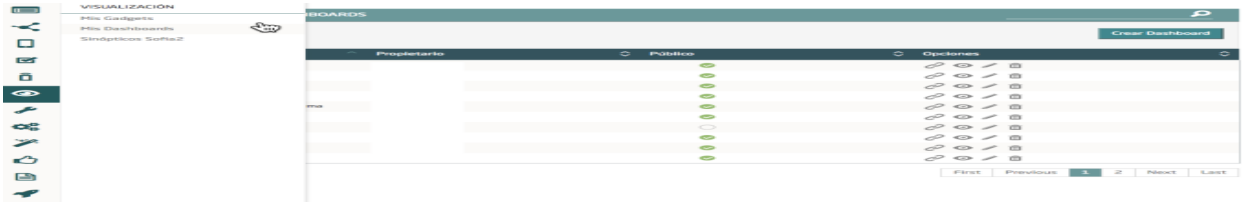
El resultado de este Gadget es una tabla como la siguiente:



	TIPO	UBICACION	HUMEDAD	TEMPERATURA	WATIOS
1	HUMEDAD	P1	77,25	0	0
2	TEMPERATURA	P1	0	35,43	0
3	WATIOS	S1	0	0	14,74
4	HUMEDAD	P2	24,52	0	0
5	TEMPERATURA	P3	0	29,14	0
6	WATIOS	S2	0	0	30,96
7	HUMEDAD	P2	24,83	0	0
8	TEMPERATURA	S1	0	57,32	0
9	WATIOS	P1	0	0	13,1
10	HUMEDAD	S1	84,02	0	0
11	TEMPERATURA	S2	0	30,53	0
12	WATIOS	P8	0	0	98,26
13	HUMEDAD	S1	93,08	0	0

Crear Dashboard

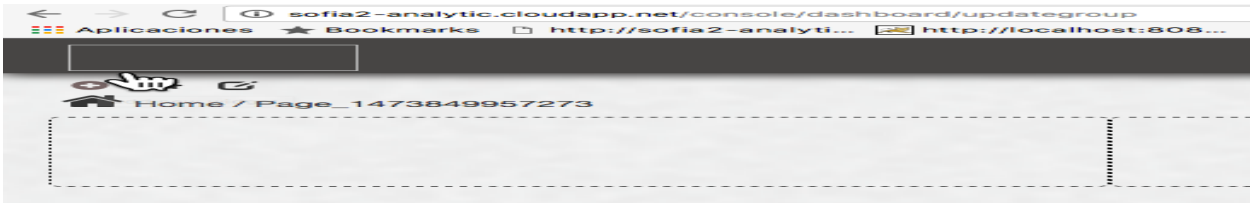
Una vez que hemos creado los Gadget, ahora vamos a crear un Dashboard que los use, para ello accedemos a la opción de menú Mis Dashboards y pulsamos sobre Crear Dashboard.



Llamaremos al dashboard TallerIoTDASHBOARD<Nuestras iniciales Nombre, 1 Apellido 2 Apellido> y lo marcaremos como público. Pulsamos el botón de Nueva Página.

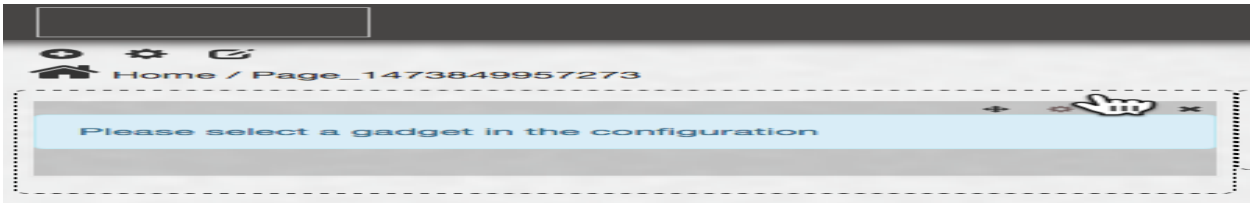
Habilitamos el modo de edición.

Pulsamos sobre el símbolo + que nos permitirá añadir un nuevo Gadget.



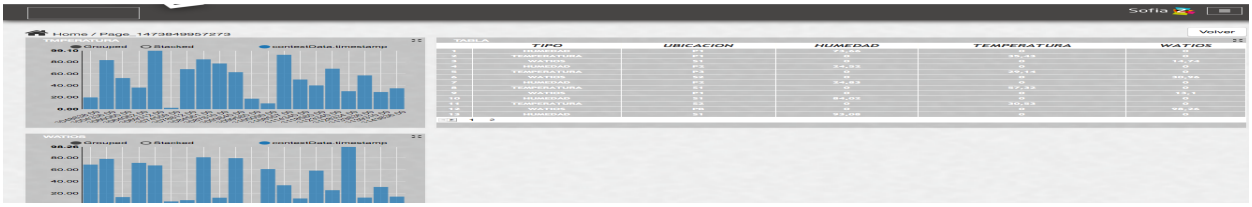
Seleccionamos el tipo de Gadget que queremos añadir, en nuestro caso son tres de tipo Bar y uno de tipo Table.

Una vez añadido el tipo de Gadget, pulsamos sobre el botón configuración.



Seleccionamos el Gadget que queremos añadir a nuestro Dashboard.

El resultado final será el Dashboard con todos los Gadget que hemos añadido.

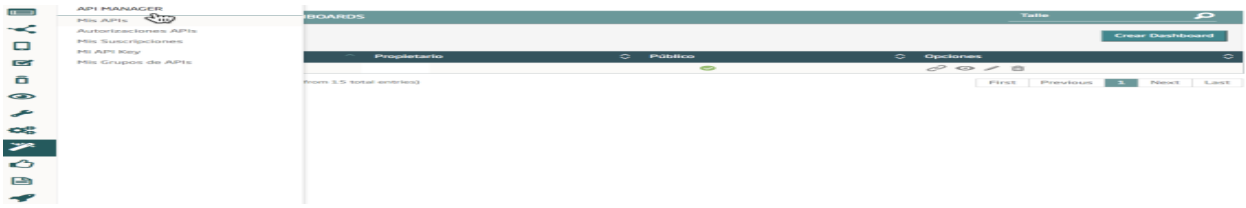


En el listado de Gadget si pulsamos sobre el símbolo del enlace,

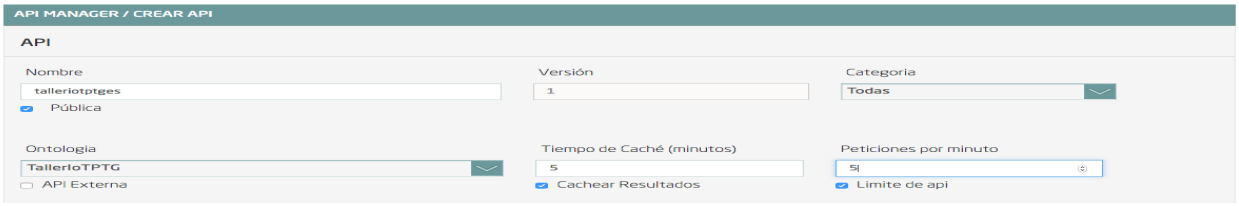
nos aparece un cuadro de diálogo con una URL en la que accedemos directamente al Dashboard y que podemos publicar.

Publicar Ontología como API

Sofia2 permite publicar nuestras ontologías como Api RST, para ello accedemos a la opción de menú Mis Apis.



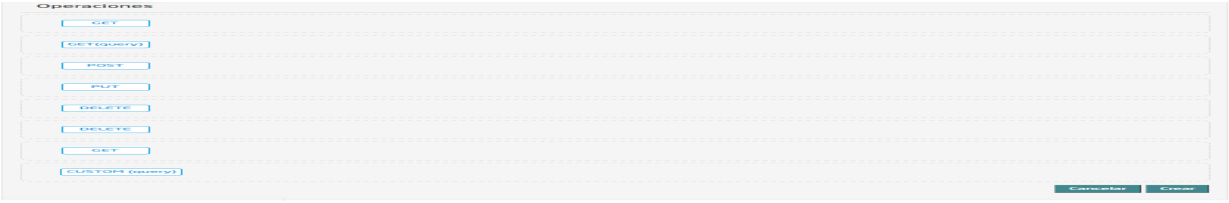
Pulsamos sobre el botón Crear Api. Le asignamos el nombre TallerIoTAPI<Nuestras iniciales Nombre, 1 Apellido 2 Apellido>, lo marcamos como público. Desmarcamos la opción API Externa y seleccionamos nuestra ontología.



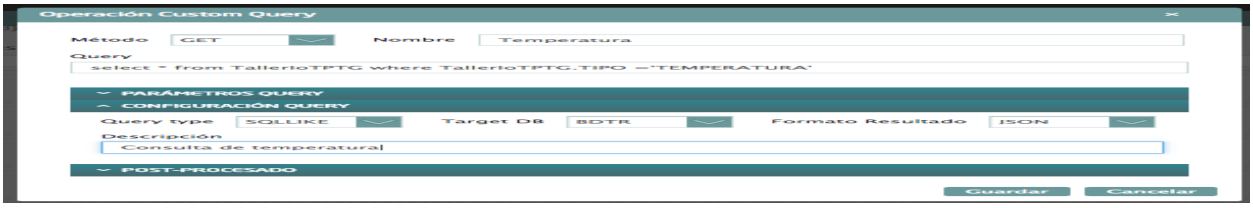
Vamos a establecer una cache de 5 minutos para los resultados de las consultas. Y un límite de 5 consultas al API por minuto.

Se nos muestra el EndPoint de acceso al API.

Debemos indicar una descripción y un valor para el campo Meta Inf. Por último nos aparecen las operaciones que podemos crear.

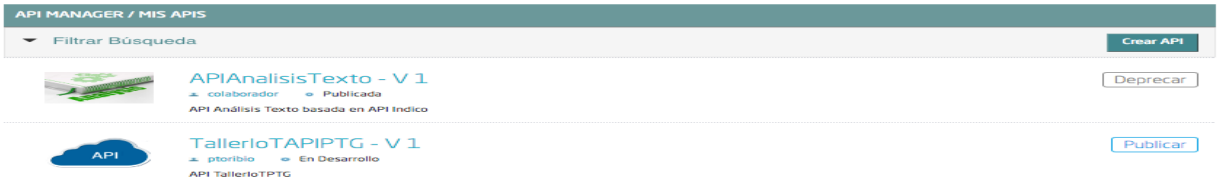


Vamos a crear tres CUSTOM, uno por cada tipo de dato que almacenamos, Humedad, Temperatura y Watios.



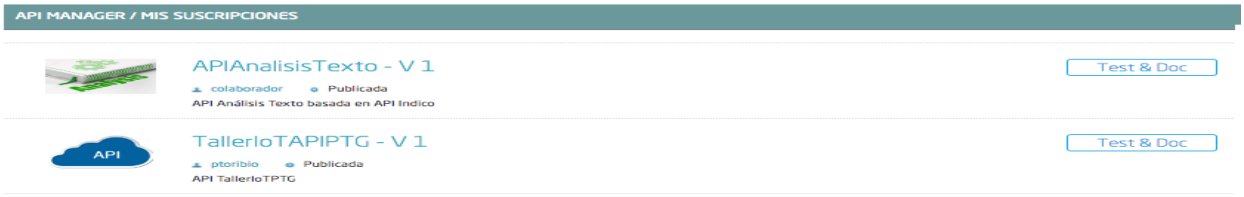
El resultado final debe de ser las siguientes tres API.

Marcamos al API como Publicada pulsando el botón Publicar del listado de API.



Accedemos al menú Mis API Key, donde debemos copiar el Token de Usuaro, el cual necesitamos para invocar las API.

Accedemos al menú Mis Suscripciones, donde aparecerán las API que tenemos publicadas.

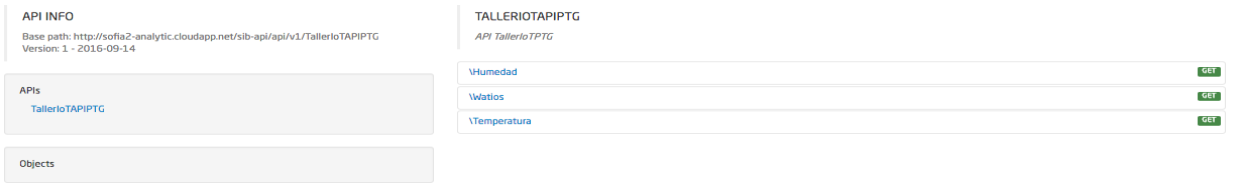


Al pulsar en Test & Doc accedemos a una página de pruebas de las API, donde en la parte derecha aparecen las operaciones que hemos expuesto.

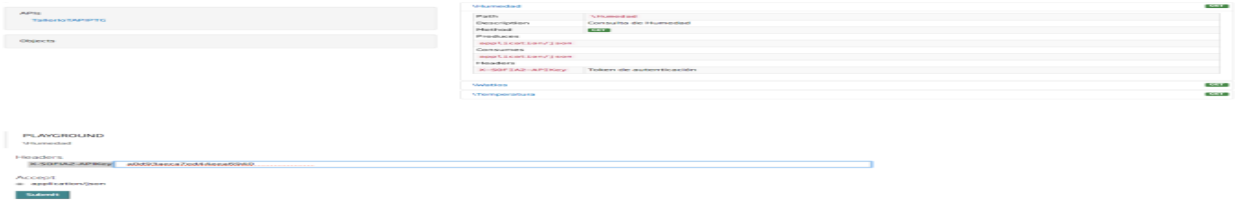
\Humedad

\Watios

\Temperatura

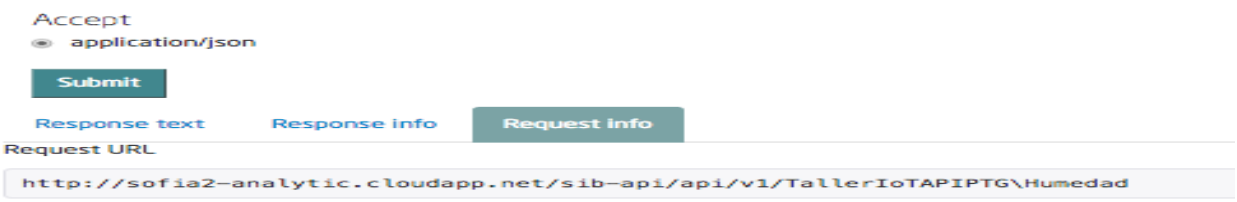


Al púlsar sobre cada opción nos aparece la meta información del servicio y la opción en la parte inferior de ejecutar con el botón Submit, debemos en la cabecera X-SOFIA2-APIKey pegar el Token de Usuario que copiamos en el punto anterior.



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

En la pestaña Request Info podemos ver el URL de invocación de la operación, que será el End Point que se creó cuando generamos el API más la operación.



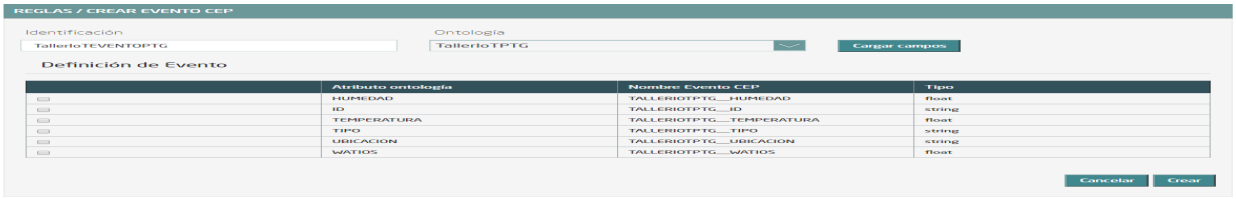
ANEXO

Los siguientes pasos del taller, nos permiten trabajar sobre dos capacidades avanzadas de Sofia2 las cuales dotan a la plataforma de la capacidad de reaccionar a eventos pudiendo analizar los datos de entrada y actuar ante ellos.

Crear Regla CEP

Accedemos a la opción de menú Mis Eventos CEP y pulsamos sobre Crear Evento.

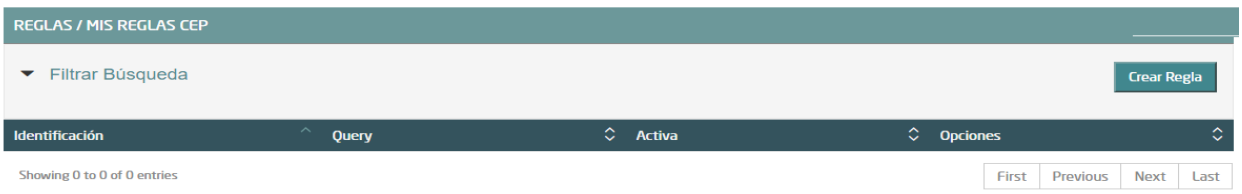
Como Identificación le asignaremos TallerIoTEVENTO<Nuestras iniciales Nombre, 1 Apellido 2 Apellido>, seleccionaremos nuestra ontología y pulsaremos el botón Cargar campos.



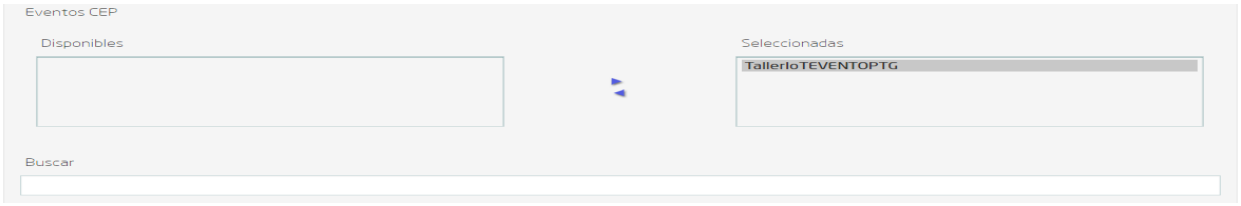
Seleccionamos los campos TEMPERATURA, TIPO y UBICACIÓN y pulsamos el boton Crear.

Fijémonos en la columna Nombre Evento CEP, ese será el nombre que deberemos usar en el siguiente punto.

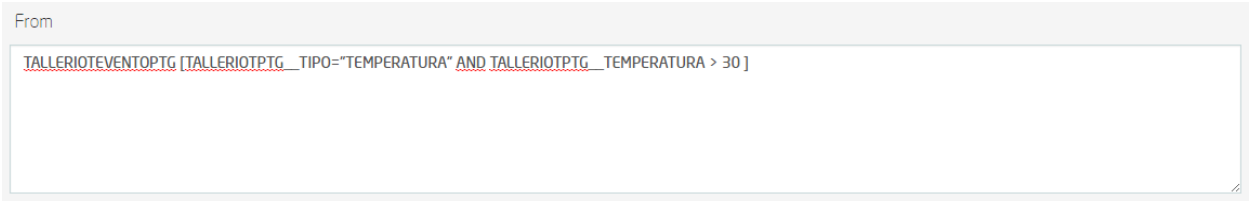
Ahora accedemos al menu Mis Reglas CEP y pulsamos sobre el botón Crear Regla.



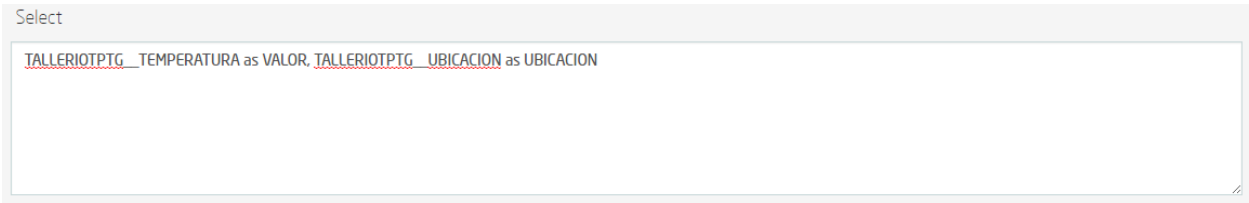
Seleccionamos el Evento que hemos creado.



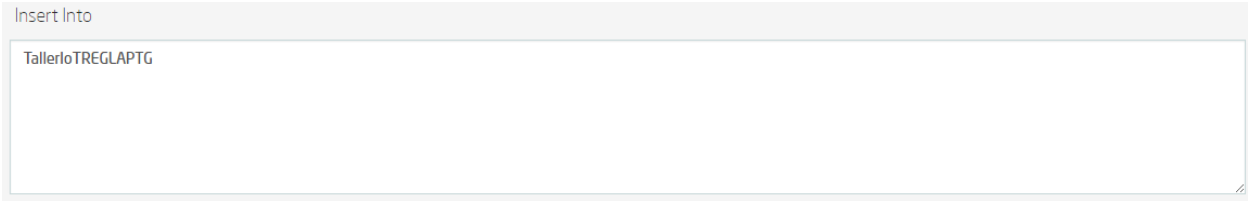
En el from establecemos los parámetros de cumplimiento de la regla.



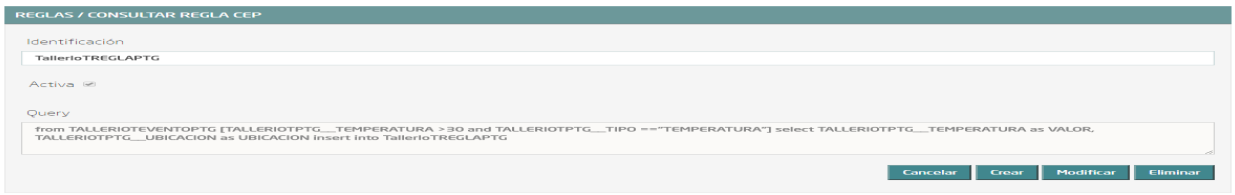
En el select los campos que queremos recuperar cuando se lance la regla CEP.



En el Insert Into la regla que queremos generar, en nuestro caso TallerIoTREGLA<Nuestras iniciales Nombre, 1 Apellido 2 Apellido>. Una vez introducidas las tres casillas, pulsamos el botón Crear.

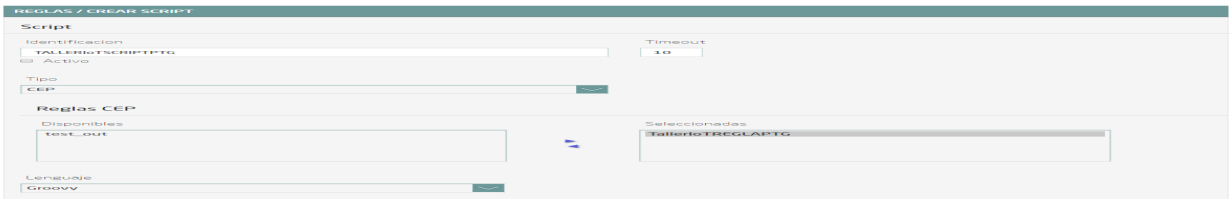


Ya hemos creado una Regla que generará un evento cada vez que llegue una instancia de ontología con el valor TEMPERATURA mayor a 30 y que sea de tipo TEMPERATURA.

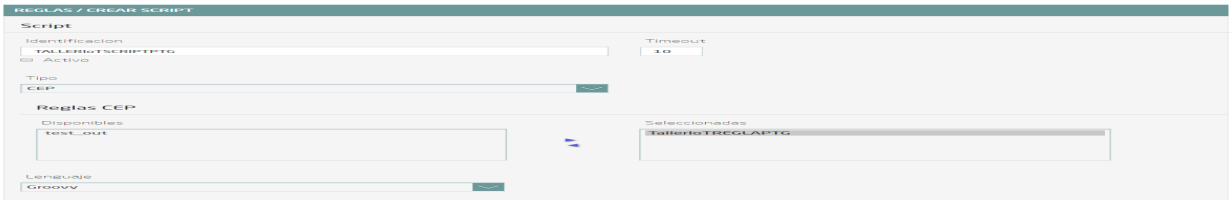


Crear Regla SCRIPT

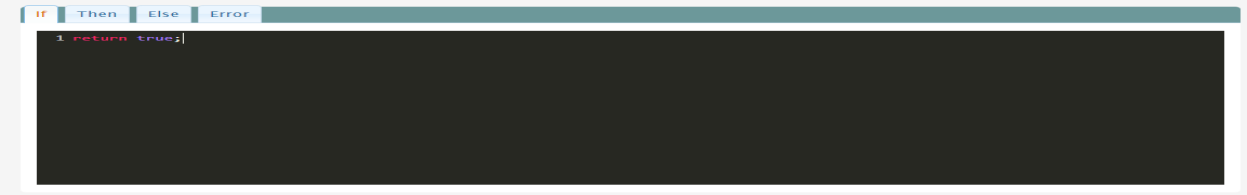
Accedemos a la opción de menú Mis Reglas Script y pulsamos sobre Crear Script.



Asignamos al Script el nombre TallerIoTSCRIPT<Nuestras iniciales Nombre, 1 Apellido 2 Apellido>. Le asignamos un timeout de 5 segundos, elegimos el tipo de Script CEP y seleccionamos la regla que hemos creado antes. Ahora cuando se lance el evento asociado a nuestra regla, se ejecutará este Script. Por último, elegimos el lenguaje del Script Groovy.



Forzamos la ejecución del bloque Then añadiendo un return true; en el bloque de evaluación if.

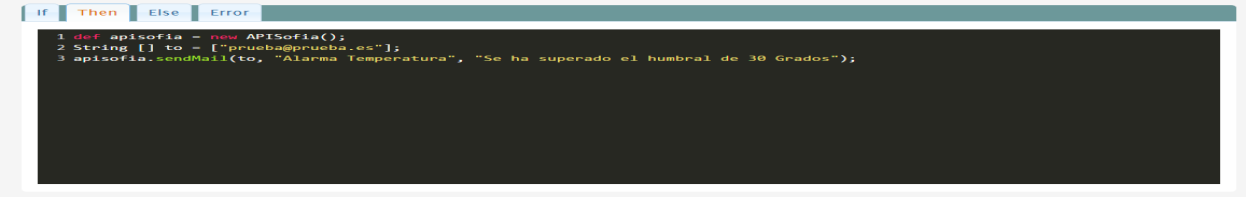


En la guia http://sofia2.com/docs/SOFIA2-Guia%20de%20Uso%20Motor%20Scripting.pdf encontraremos más información sobre el uso de Script y las API que disponibiliza.

Ejercicio Final

En el bloque then añadiremos la lógica que queramos que se ejecute cuando se produzcan los eventos del CEP.

El siguiente código envía un email avisando de que hemos excedido los 30 grados.



Si queremos recuperar los datos del Evento, la proyección que hicimos a traves de la clausula select de la Regla CEP, disponemos del Objeto inEvents.

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

Y a través del atributo getValuesJson podemos recuperar cada uno de los atributos del Evento, que eran VALOR y UBICACIÓN.

Para terminar, os propongo crear una nueva ontología, la llamaremos TallerIoTAlarma<Nuestras iniciales Nombre, 1 Apellido 2 Apellido>, esta deberá contener los campos UBICACIÓN String y VALOR Number, ambos requeridos.

Podemos usar el mismo ThinKP que creamos en el punto 4 y asignarle también esta ontología, y por último usar las API Script para realizar una inserción en la ontología Alarma cuando se produzca un evento.

A continuación, un ejemplo de como insertar una ontología desde las Reglas Script:

```
def apissap = new APISSAP();
def data = [
              VALOR: 35,
              UBICACION: "S1
           ];
def json = new  groovy.json.JsonBuilder(data);
apissap.sendInsertMessage("TOKEN", "IDENTIFICADOR KP:LITERAL LIBRE","NOMBRE ONTOLOGIA", json.toPrettyString());
```

En http://sofia2.com/desarrollador.html#documentacion disponéis de toda la documentación de la plataforma.

La guia http://sofia2.com/docs/SOFIA2-APIs%20Script.pdf describe las API disponibilizadas.

Taller Analytics

Introducción

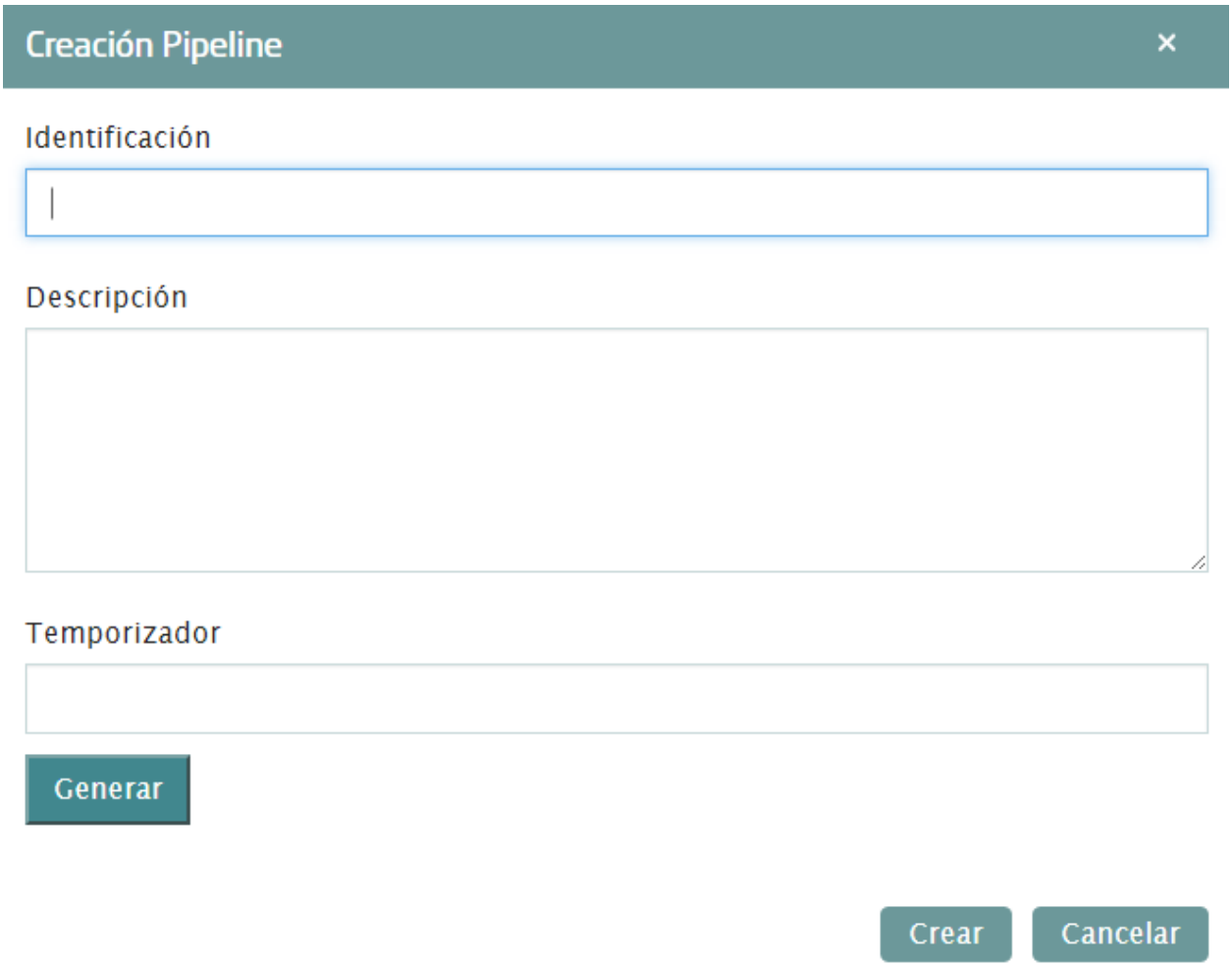
El objetivo de este taller es crear un sistema de recomendación en base a los ratings de los usuarios. Utilizaremos uno de los Dataset de Movielens que ya reside en la plataforma. Lo haremos en dos pasos:

- Ingesta y preparación de los datos mediante Pipelines.
- Creación del modelo mediante Notebook.

Ingesta de los datos

Creación Pipeline

Vamos a realizar la ingesta de los datos de películas con el Dataflow. Lo primero que hay que hacer es crear un Pipeline desde cero. Dentro de las opciones de Menú de Analytics, "Mis Pipelines", y dentro de esta pantalla, hay que pulsar el botón de Crear. Aparecerá una ventana en la que introducir el nombre del Pipeline, una descripción y un temporizador, que para esta práctica no aplica:



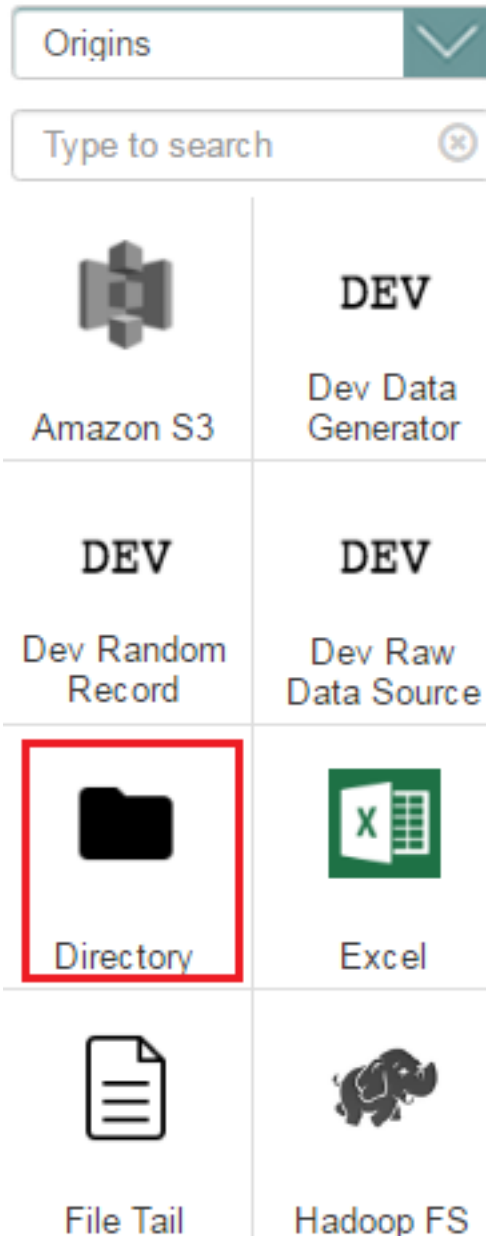
Al crear el pipeline accede directamente al espacio de trabajo en el que crearemos el flujo de información.

Definir Componente Origen

Los datos ya están descargados en la máquina de Sofia2. Dependiendo del entorno está en una ruta u otra. Para sofia2.com la ruta es "/datadrive/movielens" mientras que para sofia2-analytic la ruta "/datadrive/ftp/movielens". En este directorio deberían existir dos ficheros: movies.dat y ratings.dat. Para este pipeline nos interesan los datos de las películas.

Si no estuvieran en la máquina, hay que descargarlos para este taller.

Primero, es necesario crear un Origen de los datos. Como los ficheros ya residen en la máquina de Sofia2, el componente que se necesita es Directory. Pulsa sobre el componente y aparecerá en el espacio de trabajo.



Verás que salen alertas de errores. No te preocupes, al crear el componente vacío, los parámetros de configuración obligatorios están vacíos. Eso es justamente lo que hay que hacer en el siguiente paso.

Pulsa sobre el componente y accederás a su configuración. Para el origen de directorio local, los parámetros de configuración obligatorios son:

Files → Data Format: Representa el formato de los datos de entrada. Hay diferentes opciones, pero la que se necesita en este ejemplo es Text.

Files → Files Directory: Es el directorio de entrada, donde residen los ficheros a leer. En nuestro caso, esta ruta es /datadrive/ftp/movielens. (Si trabajas desde Sofia2.com/console la ruta es /datadrive/movielens).

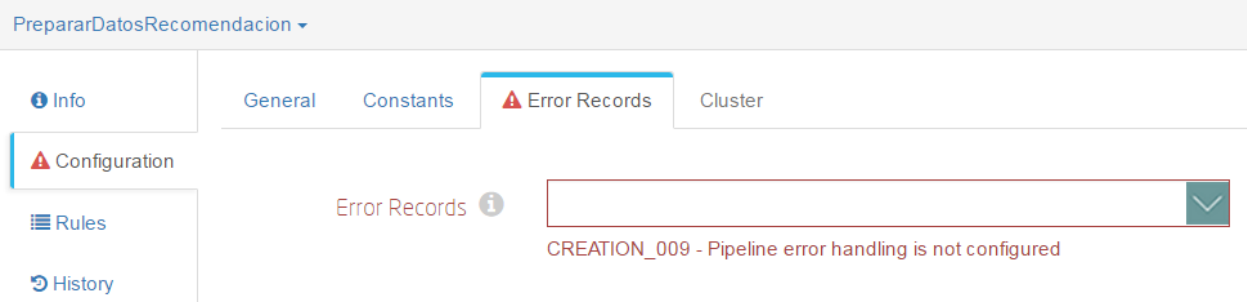
Files → Name Pattern: Es la expresión regular con la que buscará los ficheros a cargar dentro del directorio configurado en el parámetro anterior.

Nos interesa leer un solo fichero, por lo que hay que asignar a este campo en movies.dat.

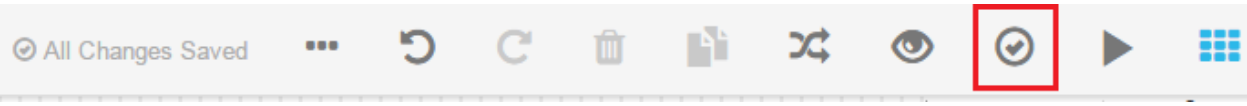
Dependiendo del formato de entrada elegido, se activa la pestaña correspondiente en la ventana de configuración. Verás que en este caso, la pestaña activa es Text. Solo tiene un parámetro que es Max Line *Length* que tiene un valor por defecto que no vamos a modificar.

Ya está configurado el origen. Para empezar, es muy recomendable echar un vistazo a los datos que se van a leer. Para ello, podemos configurar un destino "Dummy" y previsualizar la información. Para esto, accede a los componentes destino y elige "Trash". Como antes, al pulsar sobre el icono, aparece el componente en el espacio de trabajo. Une origen y destino, y ya casi está preparado este flujo.

Como observarás, todavía hay errores de configuración. Esto es porque en la configuración general hay que definir la gestión de registros erróneos. Pulsa en cualquier sitio que no sea un componente dentro del espacio de trabajo. La ventana inferior mostrará la configuración general, y verás que la alerta aparece en la pestaña "Error Records".

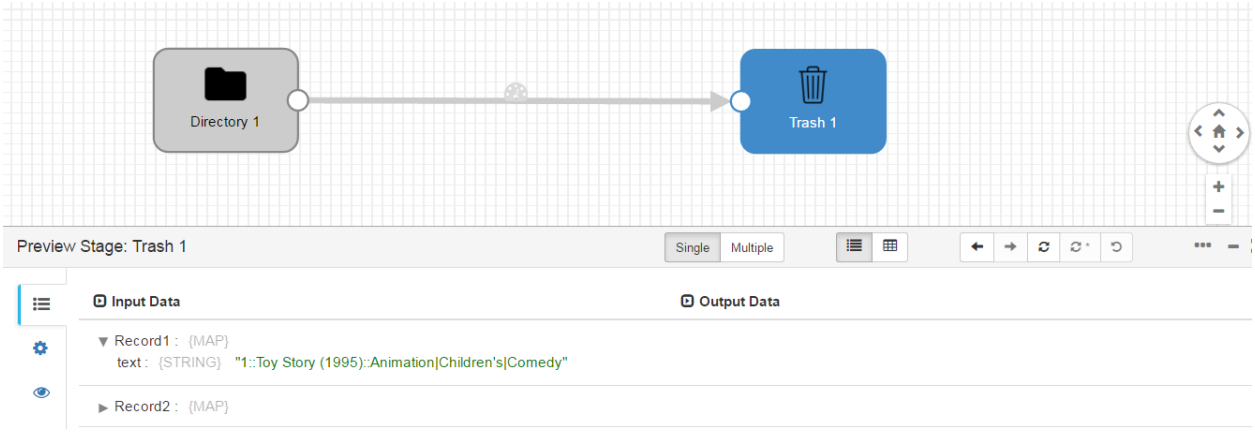


Dentro de esas opciones elige "Discard". Con esto, ya no debería haber errores, pero aún así vamos a validar el flujo. En las opciones del menú de la barra superior, pulsa sobre el botón "Validate":



Si todo es correcto, mostrará un mensaje de OK.

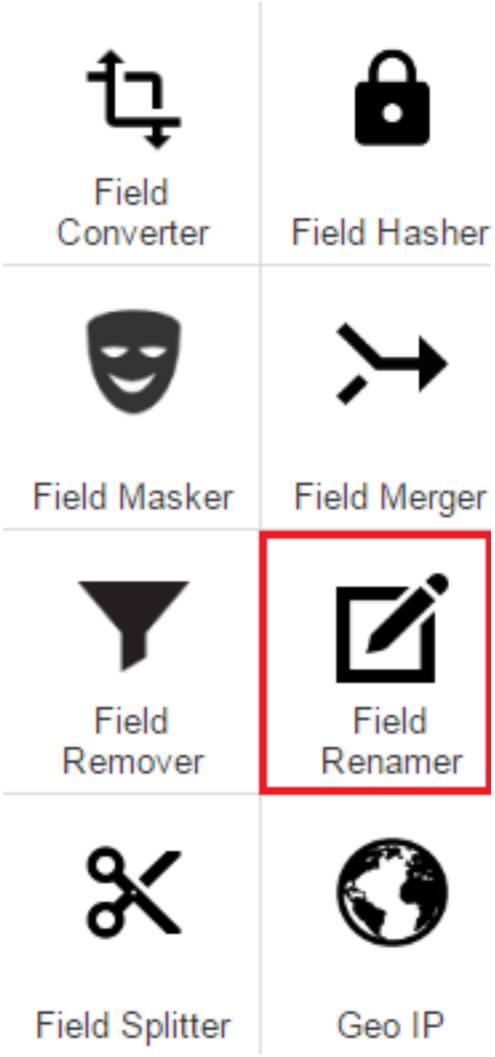
Ya podemos hacer la previsualización. Dentro del menú anterior, el botón justo a la izquierda de Validate es "Preview". Pulsa sobre él y aparecerá una ventana con unos datos de configuración. Lo único que realmente hay que tener en cuenta para este caso es el check de "Write to destinations". Si está marcado, además de previsualizar los datos los escribirá en destino. Desmárcalo si es que está marcado y pulsa sobre "Run Preview":



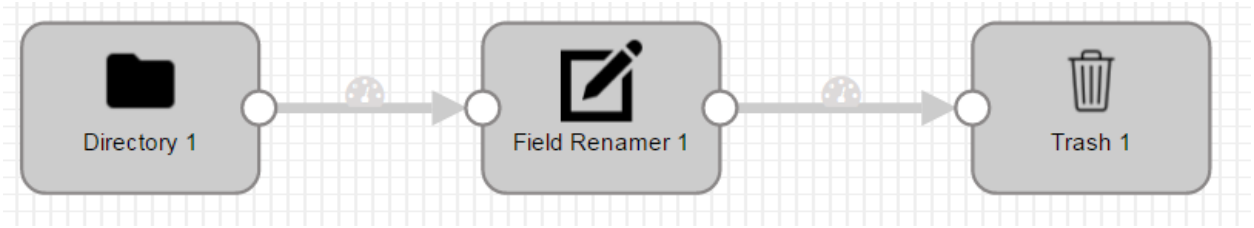
En input data puedes ver lo que lee en cada registro y en cada uno de los componentes. Si pulsas sobre el componente directory, verás lo que genera y si pulsas sobre Trash lo que recibe. En este caso es lo mismo.

Procesado de los datos

Ahora vamos a hacer la preparación de los datos. Como has podido observar en el preview del paso anterior, los campos están separados por "::". El Dataflow, interpreta los separadores como un solo carácter, por lo que no se puede definir como delimitador "::". Esto es lo siguiente que haremos.



Por comodidad, vamos a incluir, antes del cambio de delimitador, un renombrado de campos. En el preview, al desplegar cada registro aparecen los campos definidos. Al leer como formato Text, para cada línea se genera un campo que por defecto se llama "text". Este es el que vamos a renombrar. Para ello, dentro de "Processors", pulsa sobre "Field Renamer". Crea un flujo como el siguiente:

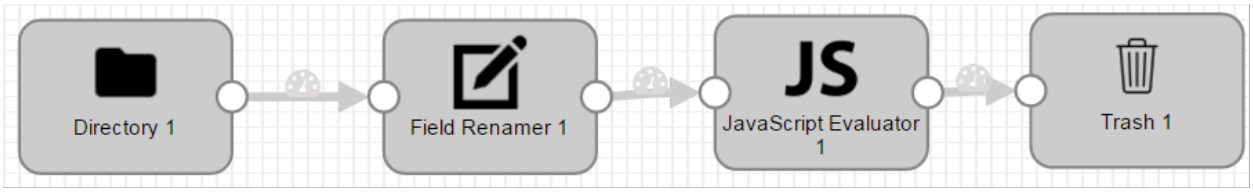


Ahora hay que configurarlo. Este componente es muy simple. Pulsa sobre él, y en su configuración accede a la pestaña "Rename". En "Fields to Rename" hay que introducir el campo origen y el nombre al que cambiarlo. Escribe como

"From Field" /text y como "To Field" /datos.

Puedes probar a previsualizar para comprobar que efectivamente está renombrando el campo.

Ahora ya podemos crear el componente que sustituye el delimitador. Para llevar a cabo esta tarea se pueden usar diferentes processors, concretamente todos los que son "Evaluators". Nosotros lo haremos con el de JavaScript. Como siempre, pulsa sobre el componente y crea un flujo como el siguiente:



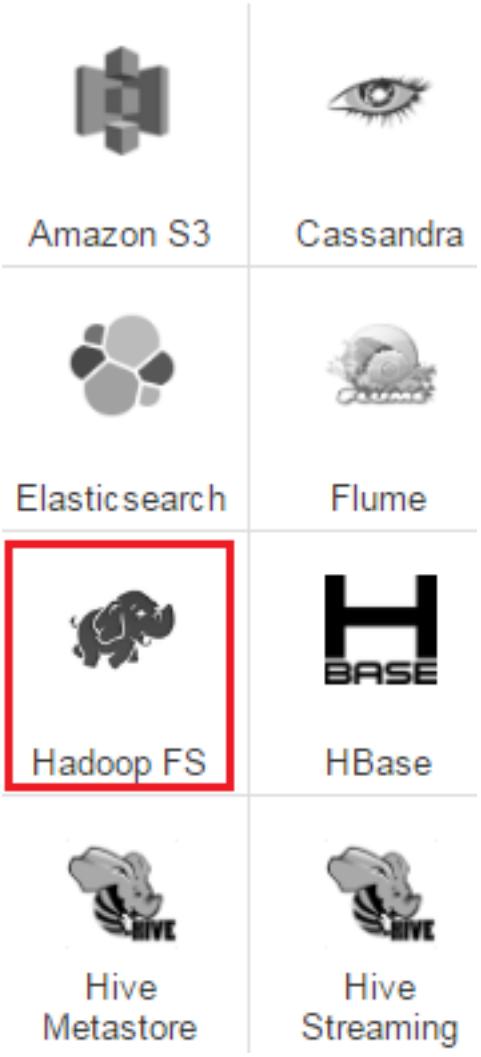
Accede a la configuración del componente, y entra en la pestaña Javascript. Verás un editor de texto que se llama

"script", que ya tiene código predefinido dentro. Es la plantilla sobre la que definiremos nuestros cambios. Dentro del bucle for, añade la siguiente línea de código:

```
records[i].value['datos'] = records[i].value['datos'].replace(/::/g, "%");
```

Esta línea lo que hace es reemplazar "::" por "%". Hemos elegido ese delimitador porque los típicos que suelen ser ";", "," y "|" aparecen en el dataset como parte de los campos. Lanza de nuevo el preview y comprueba que se ha realizado el cambio correctamente.

Definir componente destino



De nuevo, pulsa sobre el componente y crea un flujo como el siguiente:

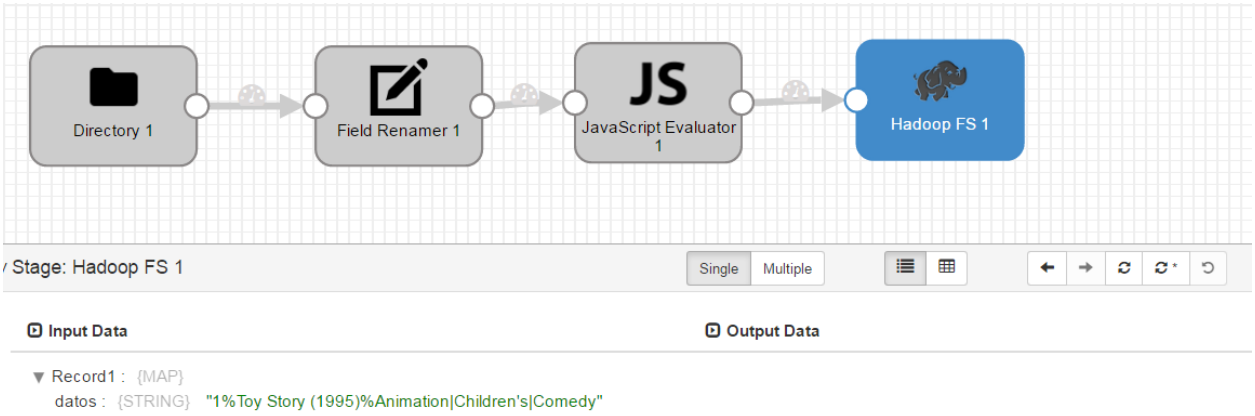
Generar DataSet

```
val recommendations = bestModel.get.predict(candidates.map((userid, _)))

val all_recommendations = bestModel.get.recommendProductsForUsers(10).flatMap{ case(u, r) =>
    r.map{ rat =>
        val m = movies(rat.product).split("%")
        (u, m(0), m(1))
    }
}.toDF("User", "Movie", "Genre")

all_recommendations.registerTempTable("recomendaciones")
```

Accede a la configuración del destino. Hay que modificar 3 pestañas:

- **Hadoop FS:** Corresponde a las conexiones y rutas del HDFS
- **Hadoop FS URI**: hdfs://localhost:8020. Nota: Si realizamos el taller desde Sofia2.com/console cambiaremos 'localhost:8020' por 'sofia2-hadoop.cloudapp.net:8020'
- **HDFS User**: cloudera-scm
- **Output Files:** Es la definición de los ficheros de salida, rutas, formato, etc.
- **File Type**: Text Files
- **Data Format**: Text
- **Files Prefix**: movie
- **Directory Template**: /user/cloudera-scm/movielens/alias_alumno/
- **Text**: Es la configuración del formato elegido en la pestaña anterior.
- **Text Field Path**: /datos

Lanza el preview de nuevo y comprueba que los datos llegan correctamente al destino:



Si todo parece correcto, pulsa sobre el botón de "Start", a la derecha del botón de validación que has usado anteriormente. Verás que se abre otra ventana con las estadísticas de los datos que se van leyendo, tiempos de proceso de cada componente, etc. Cuando veas que ya no está leyendo datos, significa que ya ha recorrido todo los ficheros de entrada. Como nosotros no necesitamos más datos que esos, podemos parar el pipeline.

¿Sabrías hacer lo mismo para el fichero de Ratings?

¿Sabrías generar el fichero en el HDFS como delimitado, definiendo los nombres de los campos separados por ";"?

Nota: Si generas el fichero con punto y coma como delimitador, ten en cuenta que en los siguientes pasos tendrás que usar ese mismo delimitador y no el "%" como aparece en el documento.

NOTEBOOK

Con ayuda de los notebooks de Sofia2 vamos a generar el modelo de recomendación de películas usando los datos que hemos cargado en la plataforma en el ejercicio anterior. Proponemos llevarlo a cabo con Spark usando Scala, y más concretamente implementaremos el ALS.

Definición de las rutas de los datos de entrada

El primer paso es leer los datos de películas y ratings, y para eso primero hay que que definir la ruta de los datos. Define las variables *ratings path* y *movies_path* con las correspondientes rutas donde hayas hecho la carga a la plataforma.

1. Definición De Las Rutas De Los Datos Descargados

```
val ratings_path = "hdfs://sofia2-analytic:8020/user/cloudera-scm/movielens/ratings*"
val movies_path = "hdfs://sofia2-analytic:8020/user/cloudera-scm/movielens/movies*"
```

Nota: Si realizamos el taller desde Sofia2.com/console cambiaremos 'sofia2-analytic:8020' por 'localhost:8020'

Estructurar los datos

Lo siguiente es guardar la información de películas y puntuaciones. Vamos a leer dicha información mediante RDDs de Spark.

Hay que definir un formato concreto tanto para las películas: (movieId, movieName) como para los rating: (timestamp % 10, Rating(userId, movieId, rating)).

También aprovechamos a importar las librerías de Mlib que se van a usar en el ejemplo. En concreto se necesitan ALS, Rating y MatrixFactorizationModel.

2. Guardar Los Datos En Rdd

```
import org.apache.spark.mllib.recommendation.{ALS, Rating, MatrixFactorizationModel}

val movies = sc.textFile(movies_path).map { line =>
  val fields = line.split("%")
  // format: (movieId, movieName)
  (fields(0).toInt, fields(1))
}.collect.toMap


val ratings = sc.textFile(ratings_path).map { line =>
  val fields = line.split("%")
  // format: (timestamp % 10, Rating(userId, movieId, rating))
  (fields(3).toLong % 10, Rating(fields(0).toInt, fields(1).toInt, fields(2).toDouble))
}
```

Comprobaciones de los datos

Ahora, comprueba que efectivamente se han leído los datos. ¿Cuántas puntuaciones has descargado? ¿Cuántas películas hay en el catálogo? ¿Cuántas películas se han puntuado? ¿Y cuántos usuarios lo han hecho?

3. Comprobaciones

```
val numRatings = ratings.count
val numUsers = ratings.map(_._2.user).distinct.count
val numMovies = ratings.map(_._2.product).distinct.count

println("Got " + numRatings + " ratings from "
  + numUsers + " users on " + numMovies + " movies.")
```

Dividir el dataset

Antes de construir el modelo hay que dividir el dataset en partes más pequeñas, una para entrenamiento(60%), otra para validación(20%) y otra más para testing(20%).

4. Dividir El Dataset

```
 val training = ratings.filter(x => x._1 < 6)
  .values
  .cache()
val validation = ratings.filter(x => x._1 >= 6 && x._1 < 8)
  .values
  .cache()
val test = ratings.filter(x => x._1 >= 8).values.cache()

val numTraining = training.count()
val numValidation = validation.count()
val numTest = test.count()

println("Training: " + numTraining + ", validation: " + numValidation + ", test: " + numTest)
```

Función para evaluar el modelo

Una vez divididos los datos, definamos la función que evaluará el rendimiento del modelo. En concreto usaremos Root Mean Squared Error (RMSE) y esta es la versión en Scala:

5. Función Para Evaluar El Modelo

```
import org.apache.spark.rdd.RDD

/** Compute RMSE (Root Mean Squared Error). */
def computeRmse(model: MatrixFactorizationModel, data: RDD[Rating], n: Long): Double = {
    val predictions: RDD[Rating] = model.predict(data.map(x => (x.user, x.product)))
    val predictionsAndRatings = predictions.map(x => ((x.user, x.product), x.rating))
    .join(data.map(x => ((x.user, x.product), x.rating))).values
    math.sqrt(predictionsAndRatings.map(x => (x._1 - x._2) * (x._1 - x._2)).reduce(_ + _) / n)
}
```

Elección del modelo

Ahora puedes usar esta función para definir los parámetros para el algoritmo de entrenamiento. El algortimo ALS requiere 3 parámetros: el rango de la matriz de factores, el número de iteraciones y una lambda. Vamos a definir diferentes valores para estos parámetros y probar diferentes combinaciones de ellos para determinar cuál de ellas es la mejor:

6. Elegir Los Parámetros Para El Modelo

```
val ranks = List(8, 12)
val lambdas = List(0.1, 10.0)
val numIters = List(10, 20)
var bestModel: Option[MatrixFactorizationModel] = None
var bestValidationRmse = Double.MaxValue
var bestRank = 0
var bestLambda = -1.0
var bestNumIter = -1
for (rank <- ranks; lambda <- lambdas; numIter <- numIters) {
  val model = ALS.train(training, rank, numIter, lambda)
  val validationRmse = computeRmse(model, validation, numValidation)
  println("RMSE (validation) = " + validationRmse + " for the model trained with rank = "
    + rank + ", lambda = " + lambda + ", and numIter = " + numIter + ".")
  if (validationRmse < bestValidationRmse) {
    bestModel = Some(model)
    bestValidationRmse = validationRmse
    bestRank = rank
    bestLambda = lambda
    bestNumIter = numIter
  }
}
```

¿Cuál crees que es el mejor modelo?

Ahora vamos a lanzar nuestra función sobre los datos de Test.

7. Rmse Sobre Test

```
// evaluate the best model on the test set
val testRmse = computeRmse(bestModel.get, test, numTest)

println("The best model was trained with rank = " + bestRank + " and lambda = " + bestLambda
  + ", and numIter = " + bestNumIter + ", and its RMSE on the test set is "+
  + testRmse + ".")
```

Ejecutar las recomendaciones para un usuario

Una vez elegido el mejor modelo ya solo quedan las recomendaciones de películas por usuario. La idea es preguntar por el usuario, que para el Dataset usado es un numérico. Vamos a hacerlo tipo formulario, de tal forma que primero pregunte por el usuario, se inserte en un campo de texto y por último lance la recomendación. Para preguntar por el usuario:

Elige Usuario

```
println("¿A qué usuario hay que recomendar peliculas?  " )
val uid : AnyRef = z.input("User Id")
val userid = z.input("User Id").asInstanceOf[String].toInt
```

Para este ejemplo, definimos que se muestren las 10 mejores recomendaciones para el usuario insertado en el campo de texto.

Recomendaciones

```
val candidates = sc.parallelize(movies.keys.toSeq)

val recommendations = bestModel.get
      .predict(candidates.map((userid, _)))
      .collect()
      .sortBy(- _.rating)
      .take(10)

var i = 1
println("Movies recommended for you:")
recommendations.foreach { r =>
  println("%2d".format(i) + ": " + movies(r.product))
  i += 1
}
```

Persistir las recomendaciones

Ahora solo nos queda guardar las mejores recomendaciones para cada usuario en ontología. La idea es guardar registros de la forma: UserId, MovieName, MovieGenre.

Generar DataSet

```
val recommendations = bestModel.get.predict(candidates.map((userid, _)))

val all_recommendations = bestModel.get.recommendProductsForUsers(10).flatMap{ case(u, r) =>
    r.map{ rat =>
        val m = movies(rat.product).split("%")
        (u, m(0), m(1))
    }
}.toDF("User", "Movie", "Genre")

all_recommendations.registerTempTable("recomendaciones")
```

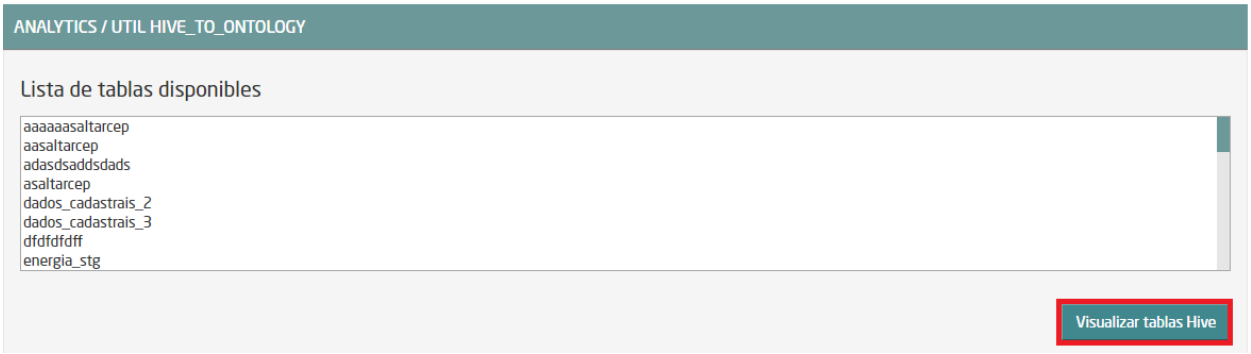
Creamos la tabla HIVE con los datos almacenados en el DataFrame. Modifica el nombre de la tabla de la imagen "recomendaciones_arturo" por un identificador único, por ejemplo, recomendaciones_tunombre.

Crear Tabla HIVE

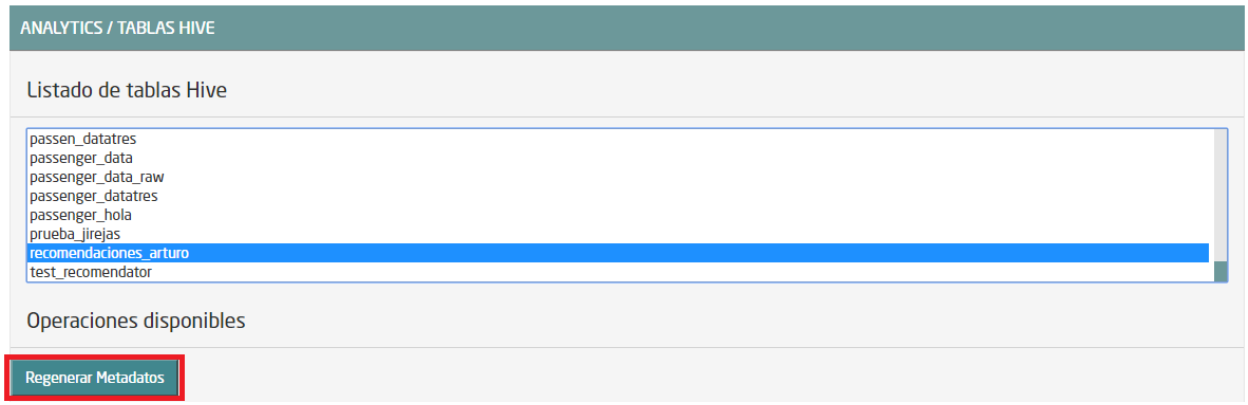
```
%sql
create table recomendaciones_arturo as select *  from recomendaciones
```

Generar Ontología

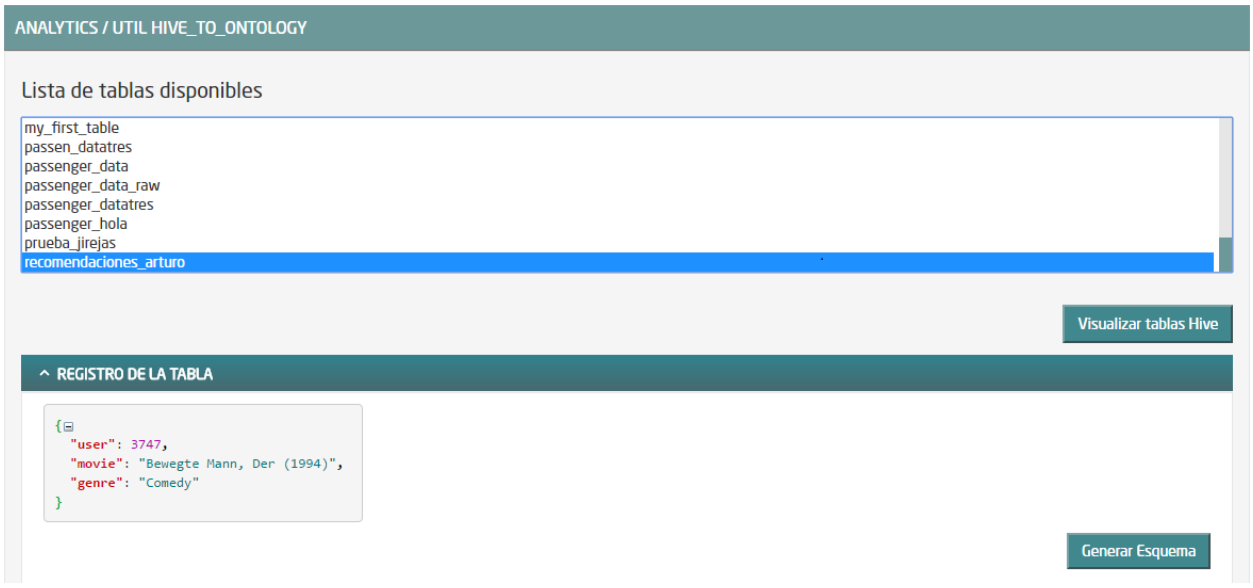
Vamos a generar una ontología a partir de la tabla HIVE que hemos creado en el punto anterior. Para ello, entra en la opción de menú de Analytics y selecciona "UTIL HIVE_To_Ontology". Se abre una ventana en la que aparece una lista de las tablas disponibles. La tabla que acabas de crear no debería aparecer. Esto sucede, porque la tabla es HIVE y esa lista muestra las entidades de IMPALA. Por lo tanto, hay que dar visibilidad a la tabla. Para ello, pulsa sobre el botón "Visualizar tablas HIVE":



Se abrirá otra ventana, en la que debería aparecer nuestra tabla. Selecciónala y pulsa sobre "Regenerar Metadatos":



Una vez ejecutado, vuelve a la ventana anterior con el botón "Cancelar". Ahora aparece nuestra tabla en la lista:

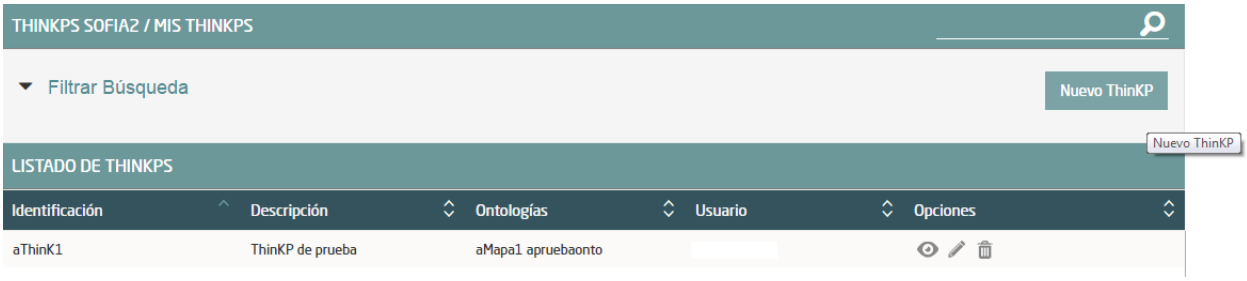


Una vez elegida la tabla, pulsa sobre "Generar Esquema" y finalmente pulsa en "Crear".

Una vez hecho esto, aparece una ventana con los datos de la ontología recién creada. Solo falta un paso más, que consiste en activar la ontología. Desde esta misma ventana, pulsa el botón "Modificar", que está al final de la página. Se abrirá otra ventana en la que hay que marcar el CheckBox "Activa" (marcado en rojo en la siguiente imagen):

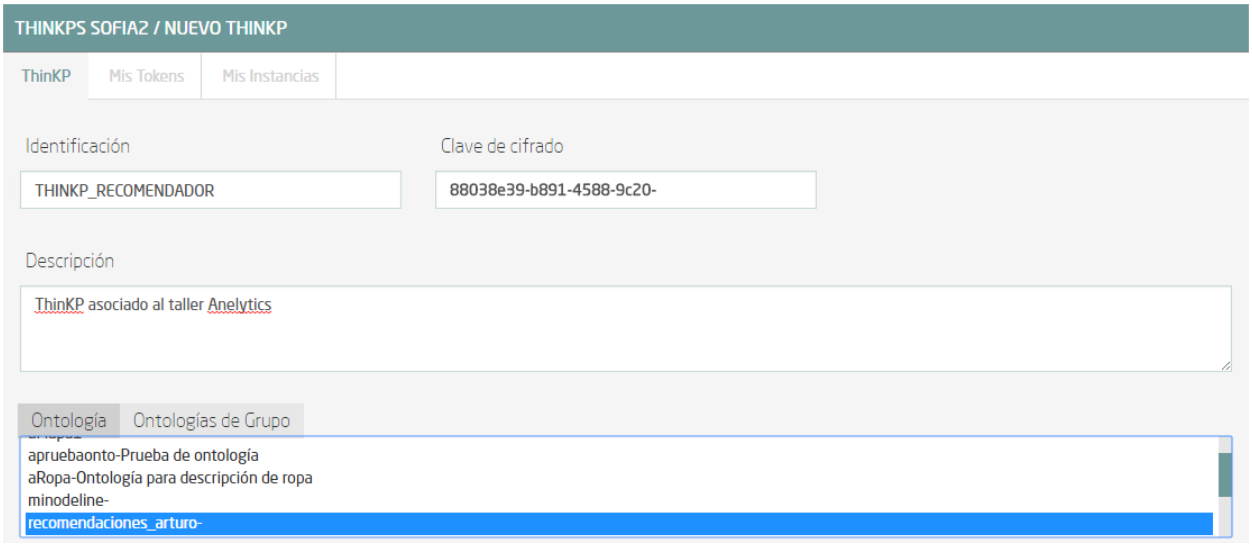
Finalmente, se genera la instancia y se pulsa sobre "Guardar". Pero para poder trabajar con ella, tenemos que asociarle un ThinKP válido. Si ya tienes uno creado puedes asociarlo a esta ontología en "Mis ThinKPs" -> Editar (tienes que elegir el ThinKP), añadiendo la ontología en cuestión a la lista asociada al ThinKP. Para este taller, vamos a crear uno nuevo.

Accede al menú "ThinKPs Sofia2" -> "Mis ThinKPs" y pulsa sobre "Nuevo ThinKP":

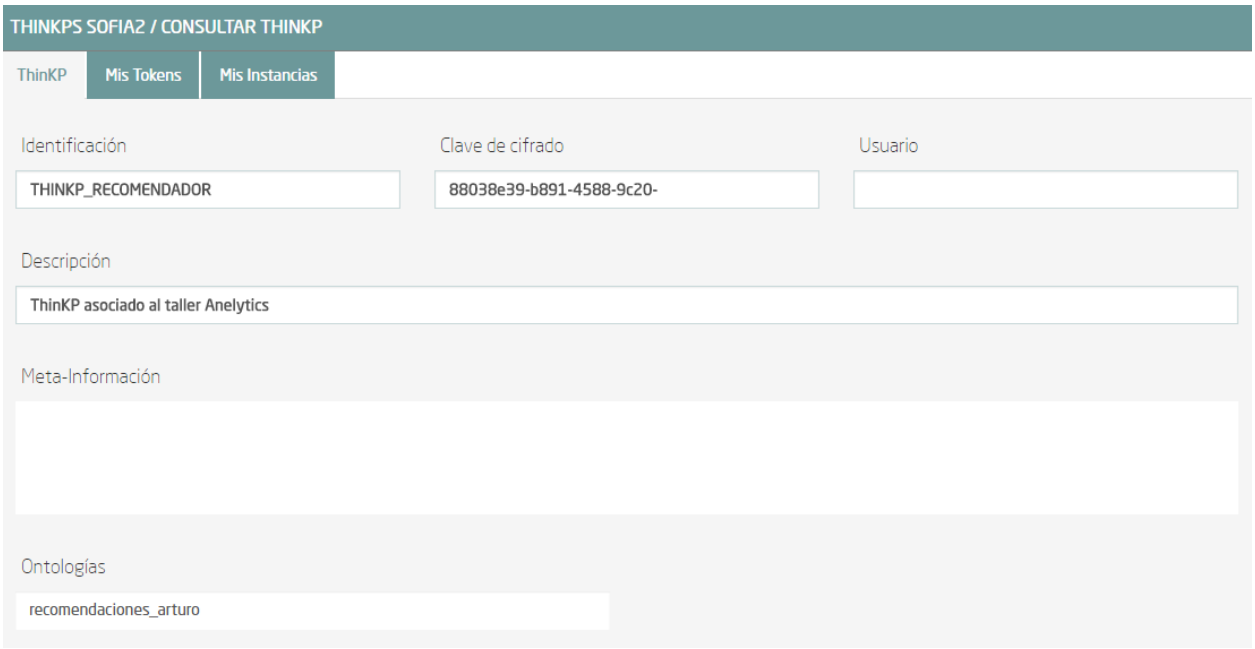


Se abre una ventana en la que hay que rellenar "Identificación" con el nombre del nuevo ThinKP, y elegir las ontologías a las que tendrá acceso al ThinKP.

Nota: Al elegir las ontologías para el ThinKP, para marcar más de una, usa el Ctrl y Shift.



Una vez rellenados los datos, pulsa sobre "Crear" y aparecerá una ventana resumen del ThinKP:



Ahora ya está preparada la ontología para trabajar con ella. Entra en la consola dentro del menú "Herramientas" y lanza alguna consulta sobre la ontología recién creada.

Nota: Es recomendable restringir los resultados de las consultas en la consola de Sofia2 con "limit numero_registros" (p.e. select * from ontología limit 5)

VISUALIZACIÓN

Para terminar vamos a crear un dashboard sobre la ontología creada.

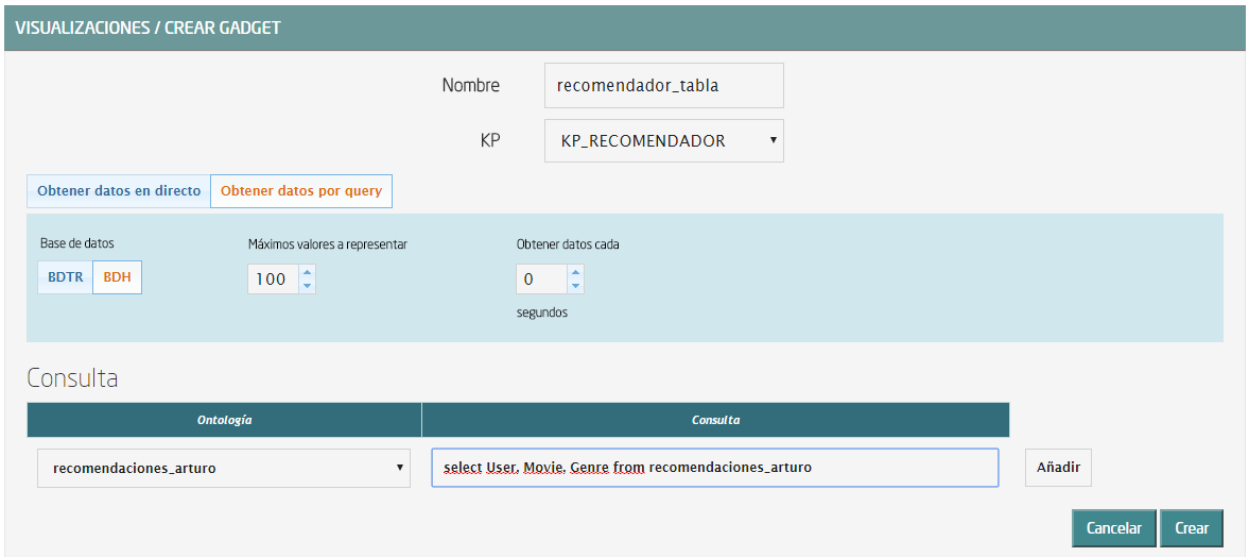
Crear Gadgets

Primero crearemos los gadgets que se mostrarán en el dashboard. Accede a la opción de Menú -> Mis Gadgets. Dentro de la nueva ventana, pulsa el botón "Crear Gadget".

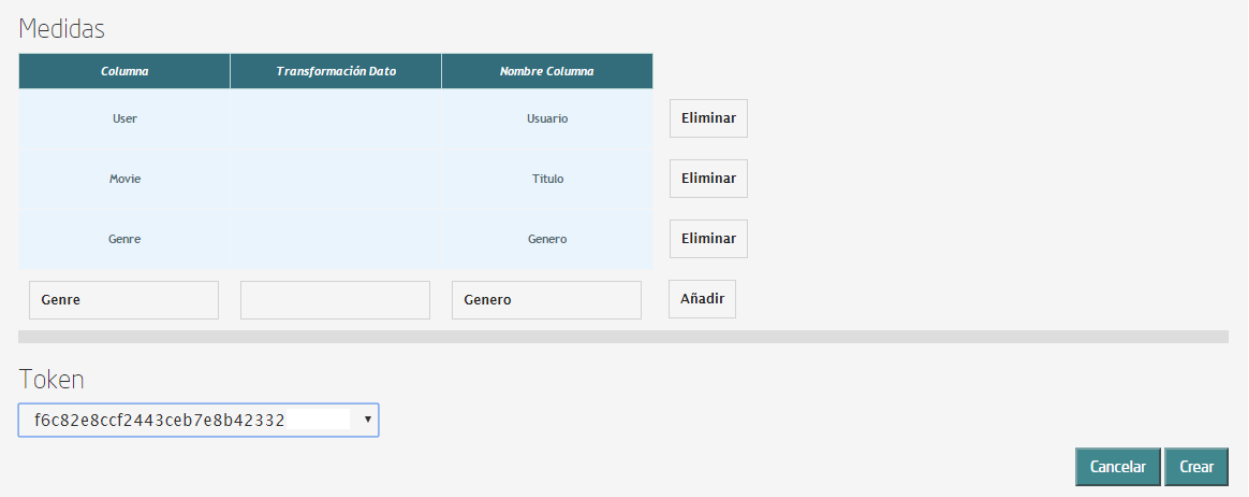
El primer Gadget que crearemos es uno de tipo tabla. Elige esa opción del catálogo. Tendrás que rellenar los datos necesarios para su creación:

- **Nombre:** p.e. recomendador_tabla_tunombre
- **KP:** el ThinKP que hayas creado en los pasos anteriores
- **Obtener Datos por query**
- **Base de Datos:** BDH
- **Máximos valores a representar:** 100
- **Obtener datos cada (segundos):** 0
- **Consulta**
- **Ontología:** elige la ontología que hayas creado para este taller
- **Consulta:** select User, Movie, Genre from nombre_ontologia

Debería quedar algo así:



Pulsa sobre el botón de añadir, junto a la query. Se desplegarán más opciones que tienes que rellenar con el nombre del campo a mostrar junto con su transformación (no es obligatoria) y el alias que aparecerá en el Gagdet. Añade los campos User, Movie y Genre.



Si todo es correcto, debajo de este bloque debería aparecer una previsualización de la tabla. Para terminar pulsa sobre "Crear". Vamos a por el segundo Gagdet. Ve al menú de creación de Gadgets y elige en el catálogo el tipo "Pie". De nuevo tenemos que rellenar una serie de atributos:

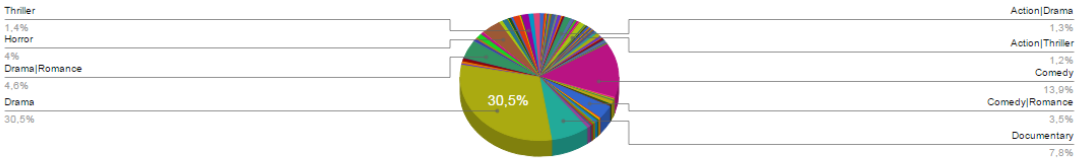
- **Nombre:** recomendador_pie_tunombre
- **KP:** el mismo ThinKP que para la tabla
- **Obtener datos por query:**
- **Base de datos:** BDH
- **Máximos valores a representar:** 100
- **Obtener datos cada (segundos):** 0
- **Consulta**

- **Ontología:** elige la ontología que hayas creado para este taller
- **Consulta:** select Genre, count(distinct Movie) as num from nombre_ontologia group by Genre

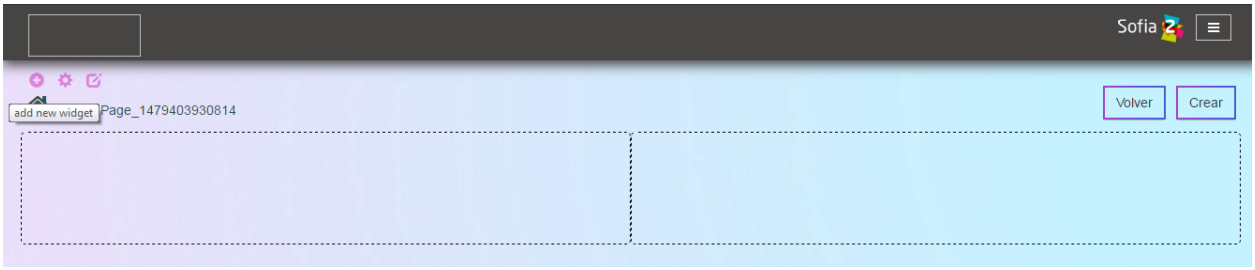
Como en la tabla, tenemos que rellenar las medidas:

- **Categoría:** Genre
- **Valor:** num

Una vez añadidas las medidas, aparecerá una previsualización del gráfico:



Ya sólo queda crear el dashboard. Para ello, accede al menú Visualización -> Mis Dashboards. En la nueva ventana que rellena el nombre, elige el theme que más te guste y pulsa en "Nueva Página". Se abrirá una ventana como esta:



Pulsa sobre el botón + para añadir un gadget creado y elige el tipo "Table Gadget". Una vez hecho, pulsa sobre el botón de configuración del Gadget, elige el que has creado en los pasos anteriores, ponle un nombre y pulsa sobre el botón "Close".

Debería mostrarte la tabla que has creado con el Gagdet.

De manera análoga, añade el Gagdet de Pie que has creado en los pasos anteriores. Una vez hecho esto, puedes mover los gadgets de posición mediante el botón de mover de cada uno de ellos. Finalmente, el dashboard podría ser algo así:

Gestión de dispositivos en Sofia2

Introducción

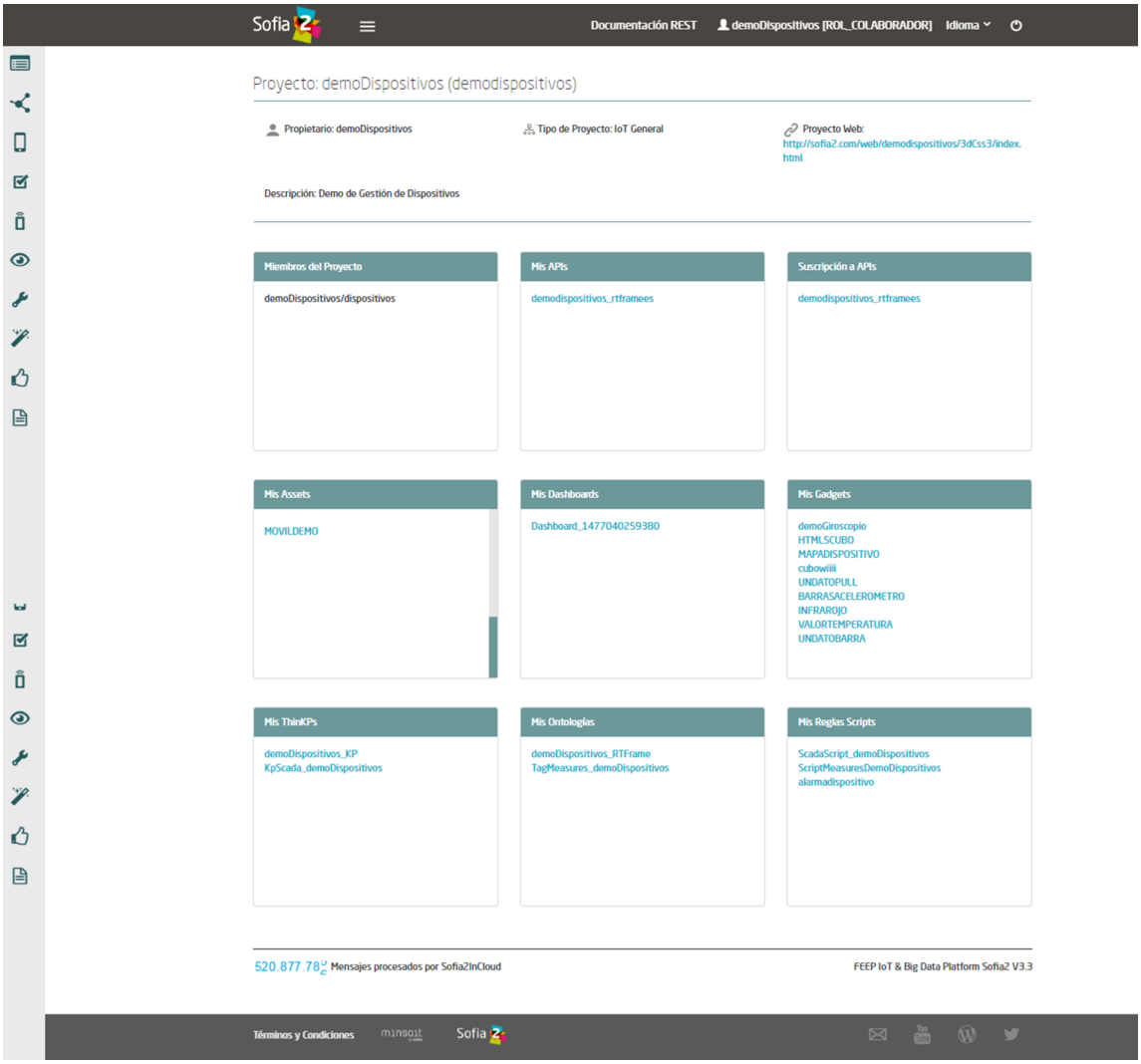
Este tutorial pretende servir como una referencia paso a paso para la evaluación de la gestión de dispositivos IoT por la plataforma Sofia2 Smart IoT Platform.

En concreto esta guía construirá un entorno de demostración con la siguiente estructura:

En este caso en concreto, se utiliza el dispositivo SensorTag de Texas Instruments como dispositivo de adquisición de datos. Este dispositivo dispone de múltiples sensores y una interfaz de comunicación a través de Bluetooth Low Energy. Usando un Smartphone, se establecerá una conexión Bluetooth con el SensorTag, y comenzará un mapeo de los valores medidos por los sensores. El Smartphone será el encargado de encapsular la información y transmitirla a Sofia2 usando redes de telefonía móvil o una red WiFi.

Gracias a las capacidades de Sofia2, la información se puede almacenar, tratar y representar de manera sencilla. En este tutorial iremos explicando paso a paso cómo crear un proyecto en la plataforma FEEP IoT Enablement Platform Sofia2, en el que configuraremos un Dashboard mostrando diversos valores recibidos desde los dispositivos, además de un Sinóptico para mostrar también esta capacidad. Además usaremos el motor de reglas, utilizando Groovy, que evaluará si el valor de una de las magnitudes críticas medidas ha excedido un valor máximo, lanzando en consecuencia un SMS de alerta.

Todas estas capacidades quedarán englobadas en un proyecto sobre Sofia2. A modo de vista general del proyecto, cuando todos los elementos estén conformados se podrá visualizar como en la siguiente figura, donde podremos distinguir los distintos componentes que conforman la solución:

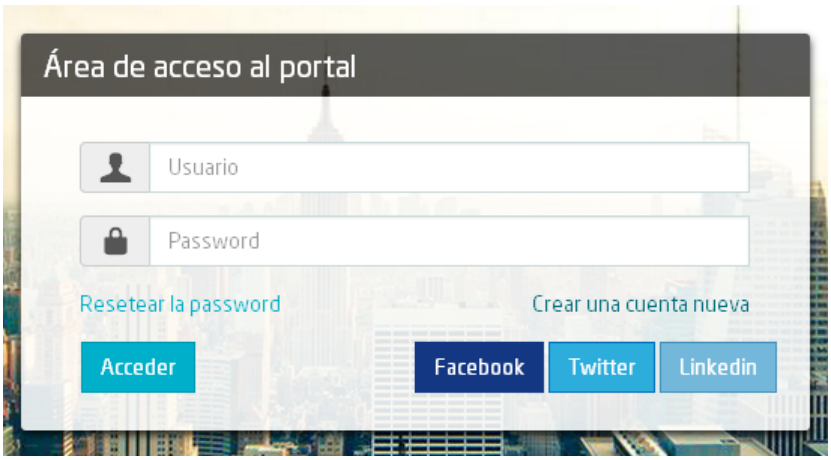


Por último, para finalizar el tutorial, explicaremos las capacidades de la plataforma que nos permitirá gestionar los dispositivos conectados y su estado.

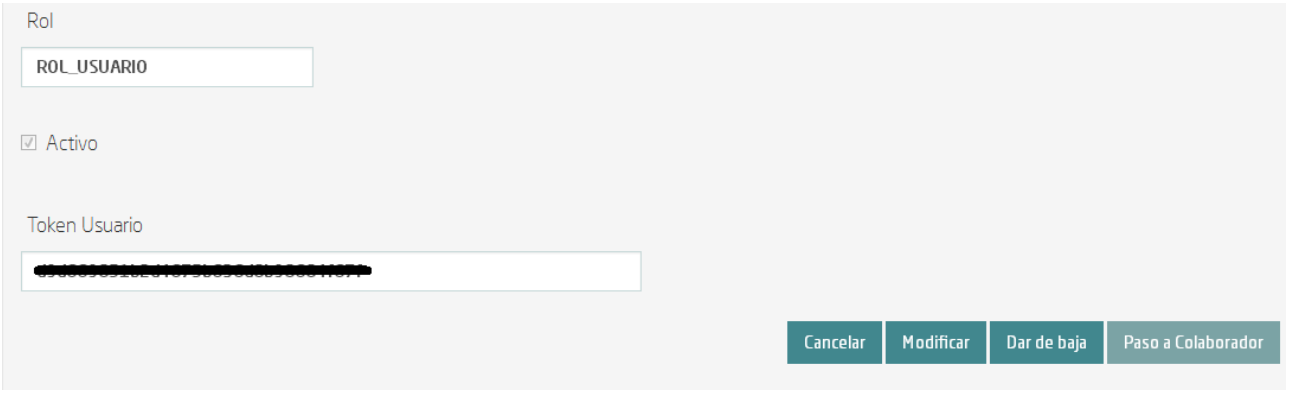
Primeros pasos: Creando un usuario y definiendo la ontología

Creando un usuario

Este tutorial se realizará sobre la instancia de Sofia2 . El primer paso será generar una cuenta de usuario exclusiva para este proyecto. Tras la creación, el usuario tendrá privilegios de ROL USUARIO



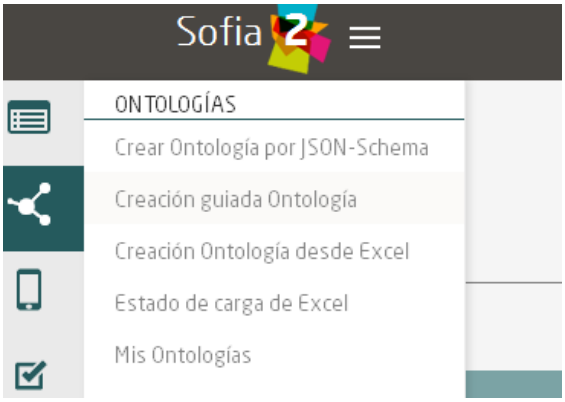
y necesitará solicitar los privilegios de ROL COLABORADOR para poder acceder a todas las capacidades que se pretenden desarrollar en este escenario.



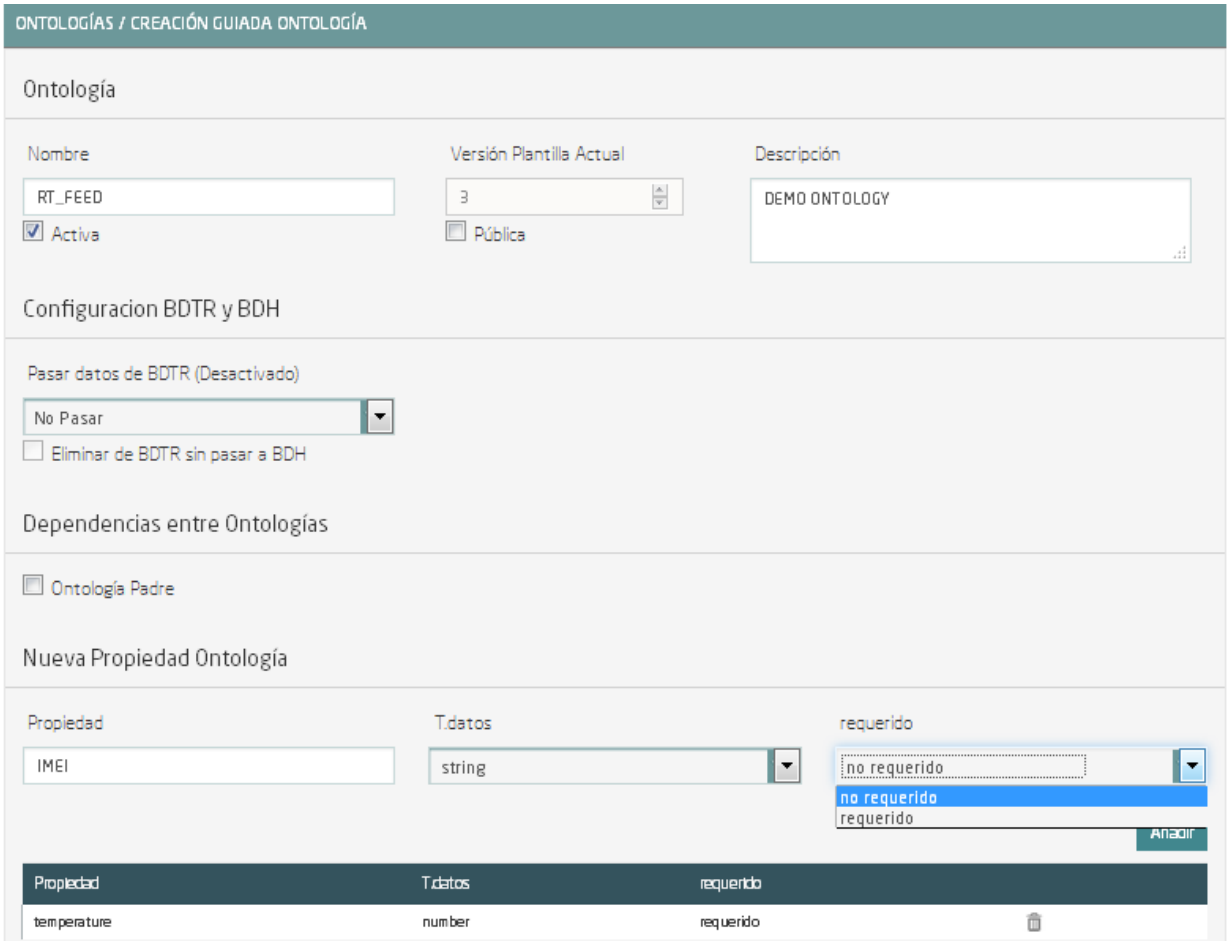
Definición de la ontología a utilizar

Para el escenario de esta demostración se creará una sola ontología con el objetivo de recoger las distintas magnitudes obtenidas tanto del dispositivo SensorTag como del Smartphone.

Sofia2 soporta múltiples métodos de creación de ontologías (interfaz gráfico de definición de JSON Schemas, generación automática desde Excel, Wizard de creación de ontologías, etc). En este caso se mostrará la **creación guiada de la ontología**. Todas las operaciones posibles que involucran a las ontologías se sitúan bajo el segundo menú de comandos:



En la creación guiada de Ontología se pueden configurar múltiples parámetros:



De inicio hay que definir un nombre que identificará a la ontología de aquí en adelante, y existe un campo de descripción para anotar las particularidades y usos de la misma. Justo debajo del campo de nombre existe un campo para activar la ontología.

A continuación aparece la configuración de las bases de datos, en cuanto al trasvase de información desde la base de datos en tiempo real (BDTR), a la base de datos histórica (BDH). Para este escenario de demo, se mantendrán los datos en la BDTR.

El apartado de dependencia entre ontologías no aplica a este escenario. A continuación aparece el apartado de añadir nueva propiedad a una ontología y que será el que se use en esta demo para añadir los datos que se desean manejar. Para este caso, se crearán los siguientes campos:

Campo	Descripción	Fuente	Tipo
deviceID	Código IMEI del dispositivo gateway	Smartphone	String
date	Fecha de generación de la trama	Smartphone	Date
accelX	Aceleración en G's sobre el eje X	SensorTag	Number
accelY	Aceleración en G's sobre el eje X	SensorTag	Number
accelZ	Aceleración en G's sobre el eje X	SensorTag	Number
gyroX	Velocidad de giro en rad/s en el eje X	SensorTag	Number
gyroY	Velocidad de giro en rad/s en el eje X	SensorTag	Number
gyroZ	Velocidad de giro en rad/s en el eje X	SensorTag	Number
temperature	Temperatura ambiente en ºC	SensorTag	Number
humidity	Valor de porcentaje de humedad relativa	SensorTag	Number
geometry	Coordenadas de geoposición	Smartphone	Geometry

El resultado final es la ontología completamente definida y lista para recibir información. En el escenario de demo,

hemos denominado a la ontología como **demoDispositivos_RTFrame**. En la siguiente figura se muestra un ejemplo de instancia de esta ontología:


```
{
    "demoDispositivos_RTFrame": {
        "deviceID": "string",
        "date": {
            "$date": "2014-01-30T17:14:00Z"
        },
        "accelX": 28.6,
        "accelY": 28.6,
        "accelZ": 28.6,
        "giroX": 28.6,
        "giroY": 28.6,
        "giroZ": 28.6,
        "temperature": 28.6,
        "humidity": 28.6,
        "geometry": {
            "type": "Point",
            "coordinates": [9, 19.3]
        }
    }
}
```


Con esta definición y manteniendo la ontología activa, en el lado de la plataforma Sofia2 sólo faltaría definir el ThinKP que se usará para interactuar con los datos, y tras este paso, ya se podrá enviar y obtener datos de la plataforma.

Conectando el dispositivo

Creación del ThinKP asociado

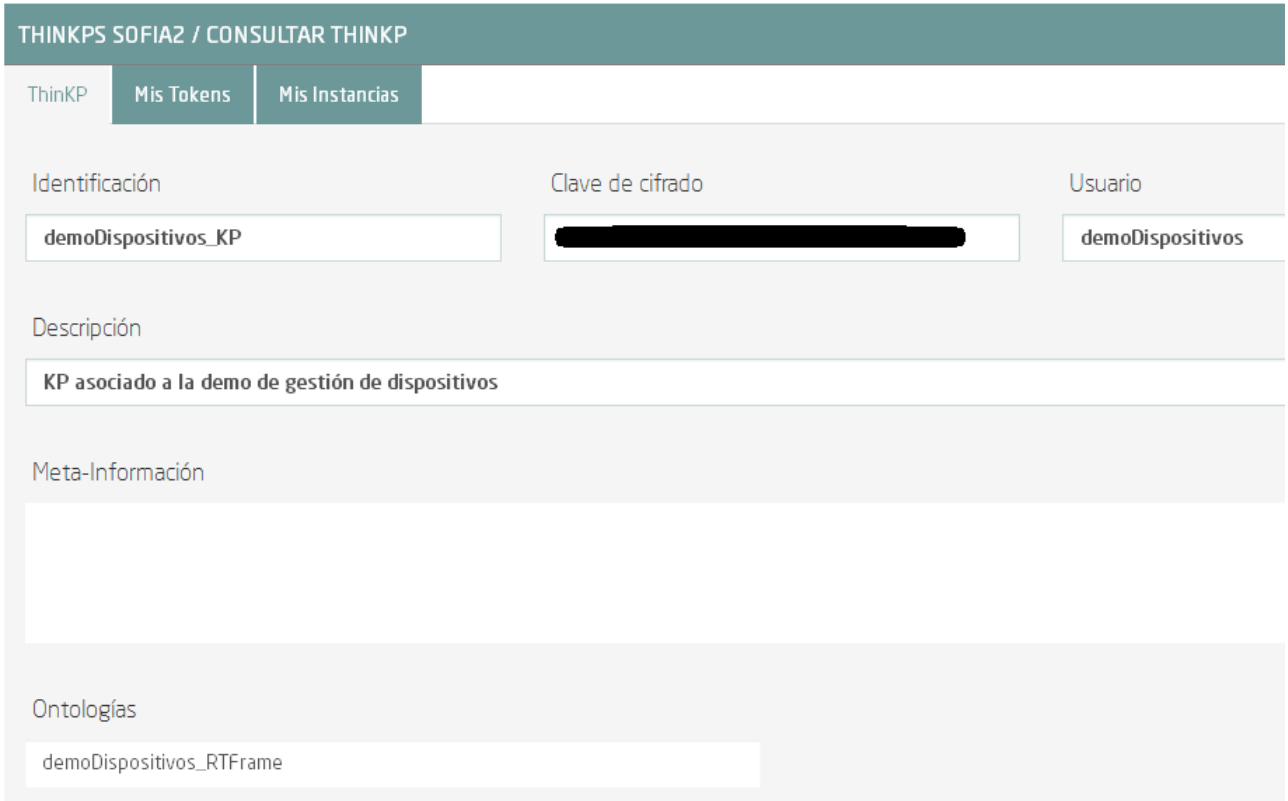
En este apartado se creará un ThinKP para este usuario de demo. Para ello hay que pulsar sobre el tercer icono del menú de comandos de la izquierda de la pantalla, y seleccionar Mis ThinKPs:

En la parte derecha de la siguiente pantalla aparecerá el botón de creación de un nuevo ThinKP:

Tras pulsar el botón se desplegará el cuadro de creación del nuevo ThinKP. La creación es muy sencilla y tan solo requiere la introducción de un identificador y una breve descripción.

Además será necesario asociar al menos una ontología asociada al ThinKP. En este caso tan solo se accederá a la ontología que creamos en los apartados anteriores, demoDispositivos_RTFrame, por lo que habría que seleccionarla y pulsar el botón de creación.

Con esto quedaría tal y como se muestra:



Uso de la instancia del ThinKP en dispositivos IoT

Tras finalizar la definición del ThinKP, quedaría listo para que distintas instancias del mismo pudieran interactuar con la plataforma. En esta demostración, se utilizará una instancia de ThinKP en el dispositivo que posee el rol de Gateway entre la placa de sensores y la plataforma Sofia2, el smartphone. Utilizando una instancia de ThinKP, se habilitará la inserción, lectura y en definitiva uso de las capacidades de Sofia2 desde el dispositivo Android.

En la siguiente figura se muestra como simplemente bastaría con introducir los valores de los parámetros asociados al ThinKP recién creado en Sofia2.

```
private final static String TOKEN = "609[redacted]a95684";
private final static String KP_INSTANCE = "demoDispositivos_KP:demoDispositivos_KP01";
private final static String ONTOLOGY_NAME = "demoDispositivos_RTFrame";
```

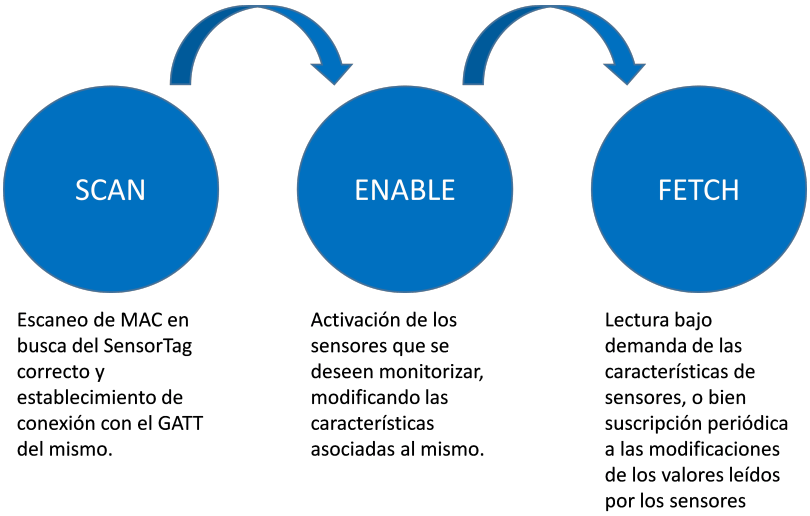
En esta demostración se enviarán los datos de sensores hacia la plataforma, utilizando el protocolo REST que otorga una gran simplicidad a la inserción de datos utilizando operaciones POST. En la siguiente figura se muestra un extracto del método de envío de tramas a Sofia2, en donde se produce el mensaje de JOIN para abrir una sesión en Sofia2, realizando un POST que utiliza los parámetros de la instancia de ThinKP asociada.

```
public synchronized  boolean send(Context context, String TOKEN, String KP_INSTANCE, String ONTOLOGY_NAME, LinkedList<String> frames ) {
    URL url = null;
    try {
        url = new URL(SERVICE_URL); //"http://sofia2.com/sib/services/api_ssap/v01/SSAPResource/"

        HttpURLConnection connection = (HttpURLConnection)url.openConnection();
        connection.setDoInput(true);
        connection.setDoOutput(true);
        connection.setRequestMethod("POST");
        connection.setConnectTimeout(10000);
        connection.setReadTimeout(10000);
        connection.setAllowUserInteraction(false);
        connection.setUseCaches(false);
        connection.setRequestProperty("Content-Type","application/json");

        connection.connect();

        DataOutputStream dStream = new DataOutputStream(connection.getOutputStream());
        dStream.writeBytes("{\"join\": true,\"instanceKP\": \"" + KP_INSTANCE + "\",\"token\": \"" + TOKEN + "\"}");

        dStream.flush();
        dStream.close();
        int responseCode = connection.getResponseCode();

        Log.d(TAG,"Sending 'POST' JOIN request to URL : " + url);

        if(responseCode!=200){
            Log.i(TAG,"Error Join " + responseCode);
            return false;
        }

        Log.d(TAG,"Join OK");
```

Con esto se obtendría un conector con Sofia2, a través del cual se pueden introducir datos en la ontología asociada siendo en este caso demoDispositivos_RTFrame.

En cuanto a la toma de datos, en esta demostración se conecta el smartphone con el dispositivo SensorTag a través de BLE (Bluetooth Low Energy). Las características de los servicios disponibles para esta placa en concreto se pueden encontrar en la web asociada de Texas Instruments.

La captura de datos del dispositivo SensorTag se puede estructurar en 3 bloques principales:



En la fase de **SCAN**, basta con utilizar el API de BLE de Android. En este ejemplo en concreto se ha desarrollado la aplicación para que sea soportada desde la versión KitKat de Android hasta las actuales. Para el escaneo se utiliza la llamada del sistema onLeScan, que se ejecuta cada vez que una nueva MAC de un dispositivo BLE ha sido detectada por el smartphone. En esta aplicación en concreto, simplemente se filtra la dirección del SensorTag y se lanza un Runnable para conectar con el dispositivo:

```
private BluetoothAdapter.LeScanCallback mLeScanCallback =
        new BluetoothAdapter.LeScanCallback() {

            @Override
            public void onLeScan(final BluetoothDevice device, int rssi, byte[] scanRecord) {
                        if(device.getAddress().equals(pref_sensor_id)&& !FOUND_FLAG){
                            Log.i(TAG,"Found!");
                            FOUND_FLAG = true;
                            mDeviceAddress = device.getAddress();
                            bleConnHandle.post(connect);
                        }
            }
        };
```

Para iniciar/pausar el escáner basta con llamar a las funciones startLeScan/stopLeScan, mostradas en la figura, pasándoles la referencia del callback de escaneo definido anteriormente.

```
private Runnable runnableCode = new Runnable() {
    @Override
    public void run() {
        if(SCAN_FLAG){
            FOUND_FLAG = false;
            loadPreferences();
            mBluetoothAdapter.startLeScan(mLeScanCallback);
            bleScanHandle.postDelayed(runnableCode, 20000);
            SCAN_FLAG = false;
            //bleConnHandle.post(connect);

        }
        else{
            mBluetoothAdapter.stopLeScan(mLeScanCallback);
            bleScanHandle.postDelayed(runnableCode, 40000);
            SCAN_FLAG = true;
        }
    }
};
```

Una vez se establece la conexión con el equipo, se pasa a la fase de **ENABLE**, donde hay que activar los sensores que se deseen monitorizar, siguiendo las directrices de la wiki de SensorTag.

El servidor GATT del SensorTag presenta un servicio para cada sensor de los que monta, y que a su vez constan de 3 características principales:

- Configuración: Sirve para encender/apagar el sensor.
- Datos: Característica donde se almacena el valor capturado por el sensor.
- Periodo: Característica que almacena el valor de la resolución de lectura del sensor.

Si se desea recibir notificaciones cuando varíen los datos de la característica de datos, habrá que activarlas siguiendo las indicaciones, y la aplicación recibirá un callback con el nuevo valor.

En esta demostración se utilizan los sensores de temperatura a través de IR (con capacidad de leer temperatura ambiente, y temperatura de un objeto a corta distancia) y el de movimiento (con capacidad de leer datos de acelerómetro, giróscopo y magnetómetro). En la siguiente figura se presenta un extracto de la información necesaria para interactuar con el sensor. En la fase de **ENABLE**, habría que escribir '0x01' en la característica de configuración del equipo, mientras que en la fase **FETCH**, se puede o bien leer directamente la característica de datos, o activar las notificaciones periódicas (usado en el proyecto).

IR Temperature Sensor

Type	UUID	Access	Size (bytes)	Description
Data	AA01*	R/N	4	Object[0:7], Object[8:15], Ambience[0:7], Ambience[8:15]
Notification	2902	R/W	2	Write 0x0001 to enable notifications, 0x0000 to disable.
Configuration	AA02*	R/W	1	Write 0x01 to enable data collection, 0x00 to disable.
Period	AA03*	R/W	1	Resolution 10 ms. Range 300 ms (0x1E) to 2.55 sec (0xFF). Default 1 second (0x64)

Con los datos de sensores obtenidos, bastará con encapsularlos en base a la ontología creada, por ejemplo conformando un String como el de la siguiente figura. En este ejemplo en concreto, se reporta también el código IMEI del dispositivo móvil a modo de indicador, y se añade la localización por GPS del smartphone para geo-localizar las medidas.

```
private String composeFrame(Location location){
    if(location.getAccuracy()>maxAccuracy){
        return "";
    }
    else{
        return "{ \"demoDispositivos_RTFrame\": { \"geometry\": { \"coordinates\": [ "+
                location.getLongitude()+", "+location.getLatitude()+" ], \"type\": \"Point\" }, \"" +
                "deviceID\": \""+IMEI+"\", \"" +
                "temperature\": "+temperatureValue+", \"" +
                "humidity\": "+humidityValue+", \"" +
                "accelX\": "+accXValue+", \"" +
                "accelY\": "+accYValue+", \"" +
                "accelZ\": "+accZValue+", \"" +
                "giroX\": "+gyrXValue+", \"" +
                "giroY\": "+gyrYValue+", \"" +
                "giroZ\": "+gyrZValue+", \"" +
                "date\": { \"$date\": \""+sdf.format(new Date())+"\"}}}";
    }
}
```

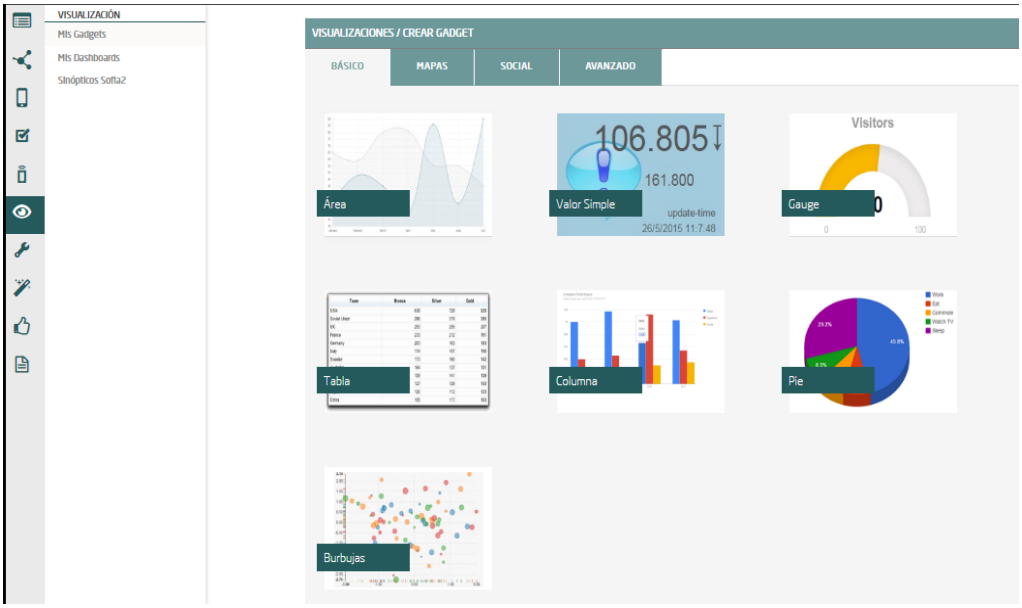
Visualizando los datos

Una vez realizados el diseño y la configuración de la ontología, en conjunto con la integración de los dispositivos IoT con Sofia2, dispondremos en la plataforma de todos estos datos, que se podrán utilizar de diversas maneras. Por ejemplo, representándola en tiempo real en un dashboard o un sinóptico, o procesándola mediante el motor de reglas.

El uso de estas dos capacidades de Sofia2 será lo que describamos en este apartado.

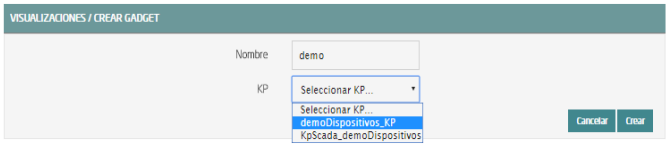
Composición de un Dashboard

Sofia2 tiene la capacidad de configurar gadgets y dashboards sobre la información disponible. Para ello accederemos al menú de Visualización, submenú de Gadgets:



Para nuestro ejemplo, crearemos un par de gadgets de valor simple, para visualizar los datos de temperatura del sensor, y un par de gadgets de columna, para visualizar los ejes x, y y z del giroscopio y el acelerómetro de los sensores de nuestro SensorTag.

Para cualquiera de los dos casos, lo primero que tenemos que hacer es dar un nombre al gadget y seleccionar nuestro ThinKP, que nos dará visibilidad a la conexión con la ontología que hayamos configurado.



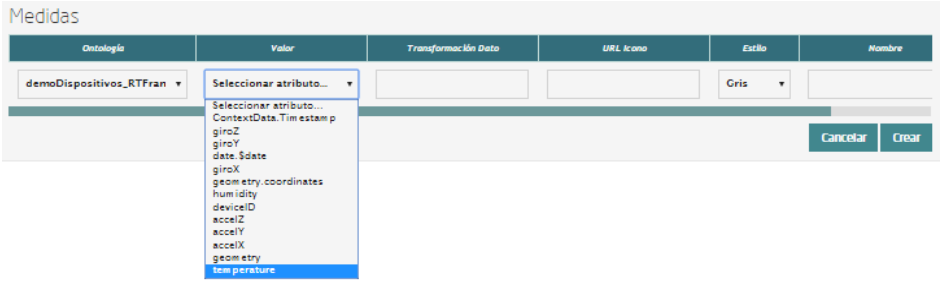
Una vez seleccionado el ThinKP, tendremos dos opciones para obtener los datos:

- Obtener los datos en directo: Esto es, el gadget se mantendrá suscrito a la ontología, actualizando el valor representado en el mismo momento en que un nuevo valor de ésta entra en el repositorio.
- Obtener datos por query: Definiremos un intervalo de tiempo para el refresco del gadget, transcurrido el cual se lanzará la consulta que definamos contra la base de datos en tiempo real o bien contra la base de datos histórica.

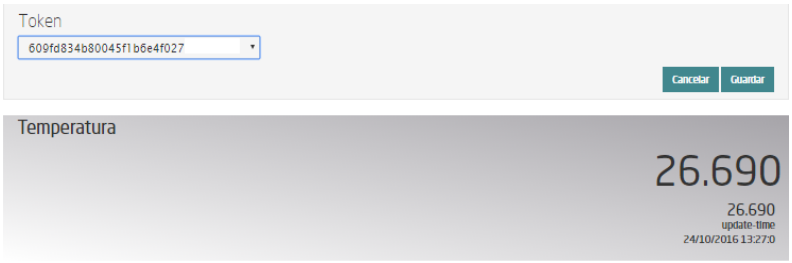
En el caso de los valores simples, elegiremos la segunda opción, lanzando cada 20 segundos la siguiente query a la base BDTR (que nos devuelve el último registro insertado en la ontología):

db.demoDispositivos_RTFrame.find().sort({'demoDispositivos_RTFrame.date':1}).sort({'contextData.timestamp':-1})

Con los datos seleccionados en nuestro gadget, solo necesitaremos seleccionar cuál de los campos de la instancia de ontología recuperada queremos representar, asignarle un nombre en la gráfica y opcionalmente una transformación del dato recuperado de la ontología:

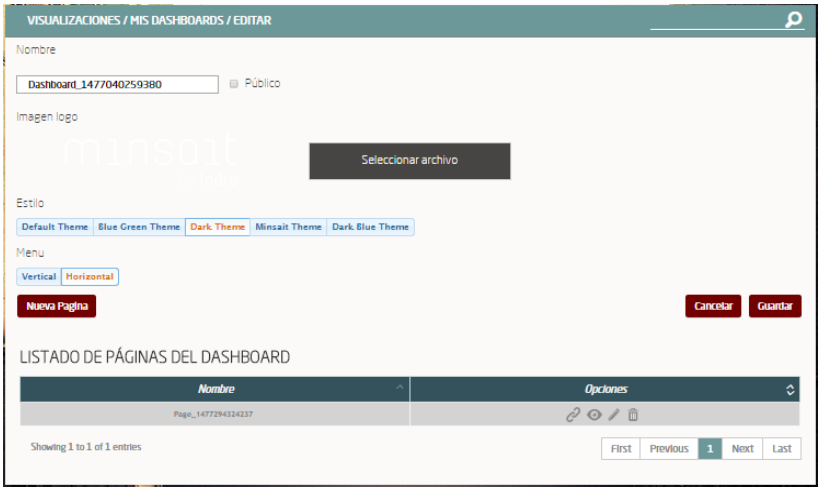


Con todos estos pasos, queda seleccionar un token de seguridad de los disponibles en el ThinKP y guardar el gadget creado.

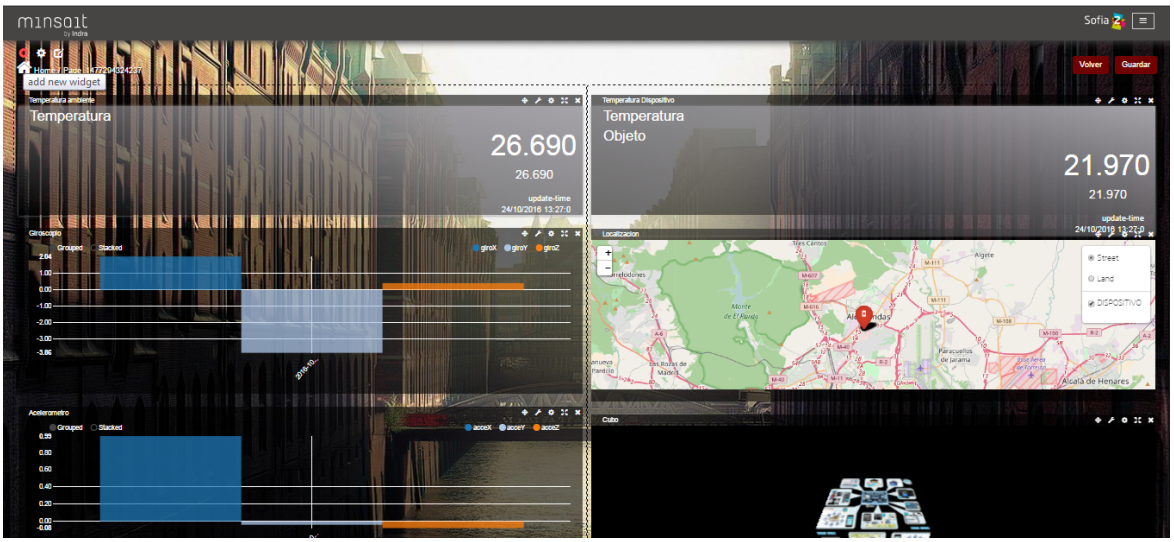


Ya con el conjunto de gadgets creados, podremos componer nuestro dashboard de una manera sencilla, accediendo al menú de visualización, submenú dashboards.

Primero configuraremos el estilo general, icono, tipo de menú y crearemos una primera página.



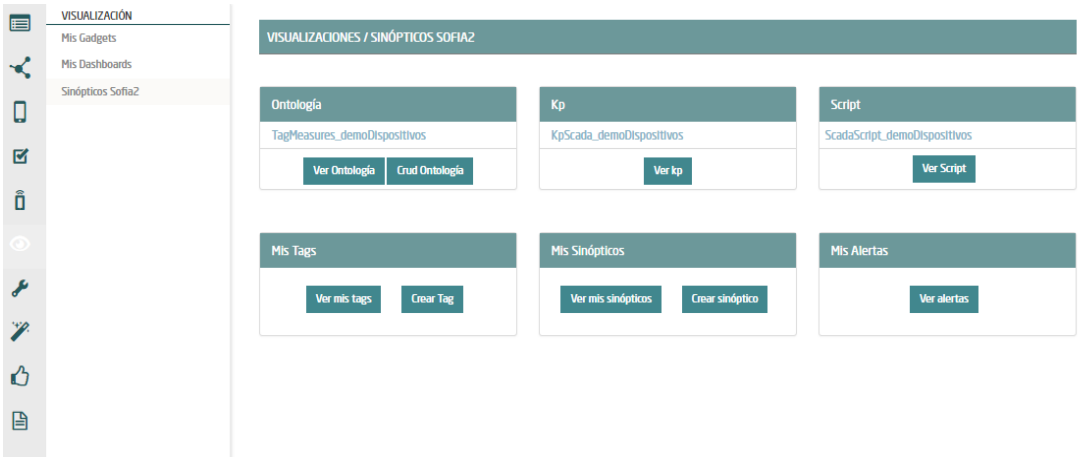
Accediendo a la nueva página recién creada del dashboard, podremos añadir los gadgets creados, y arrastrarlos al área donde queramos que se visualice. Nuestro dashboard para este tutorial es el siguiente:



Composición de Sinóptico

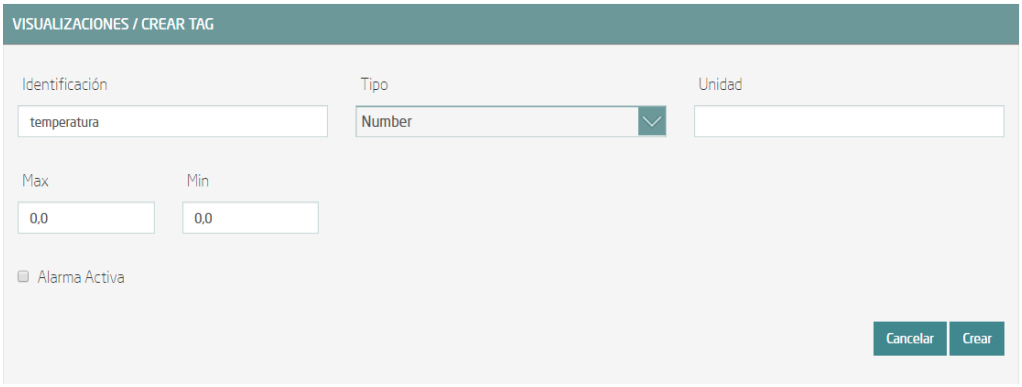
Sofia2 contiene un módulo Scada que permite la creación y configuración de sinópticos. Para nuestro ejemplo, se creará un sinóptico sencillo que permite visualizar la actualización de datos en tiempo real.

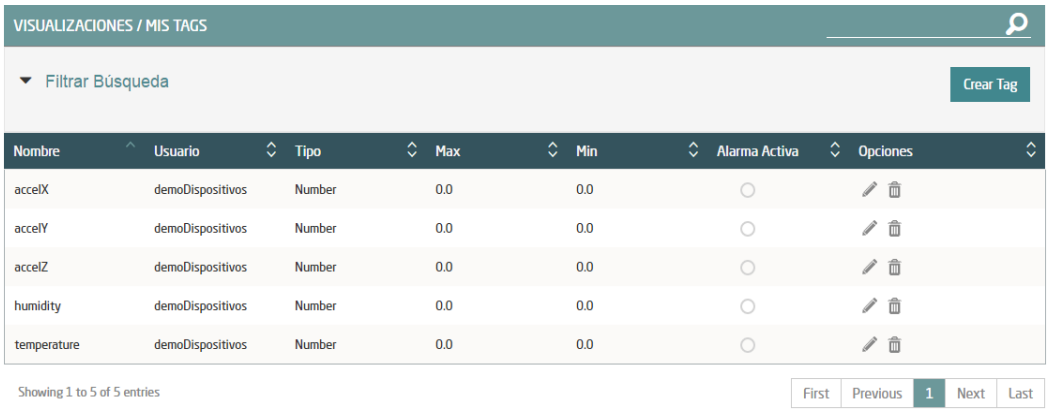
Para comenzar, se deberá acceder al menú de Visualización y al submenú Sinópticos Sofia2.



A continuación, teniendo en cuenta los atributos de la ontología creada que se quieren mostrar en el sinóptico, se definirán los tags asociados de la siguiente forma:

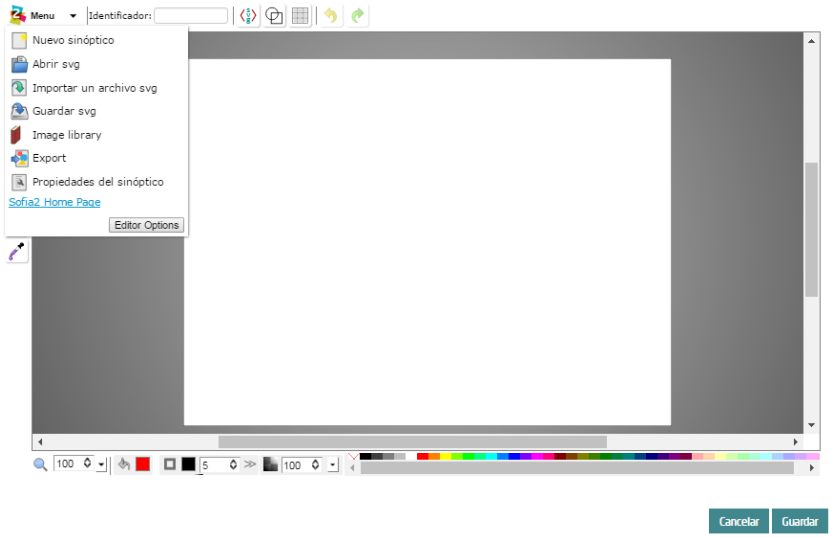
- En el apartado **Mis tags**, se seleccionará **Crear Tag**.
- Se añadirá el nombre o identificación que se quiera dar al tag.
- Se seleccionará el tipo que tomará el tag. En este caso todos los atributos serán de tipo **Number**.



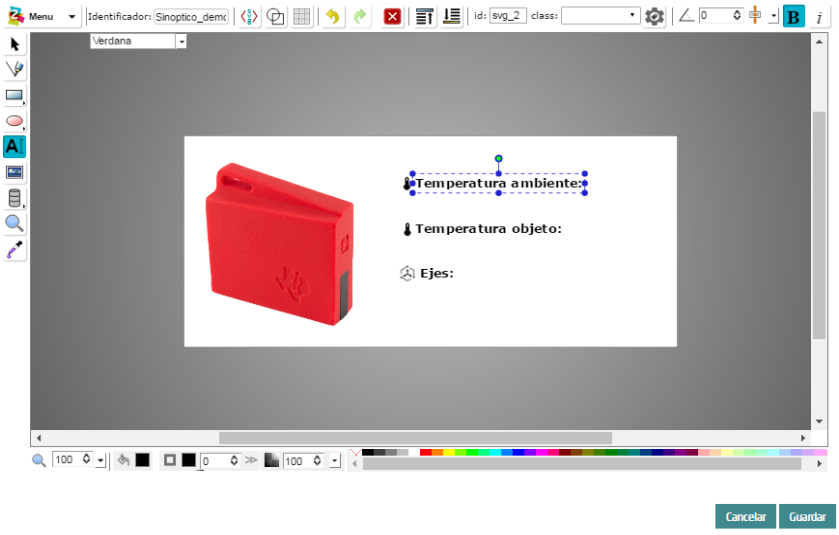


Tras concluir esta definición de tags, se volverá a la pantalla de Visualización de sinópticos y en el apartado Mis sinópticos se seleccionará Crear sinóptico.

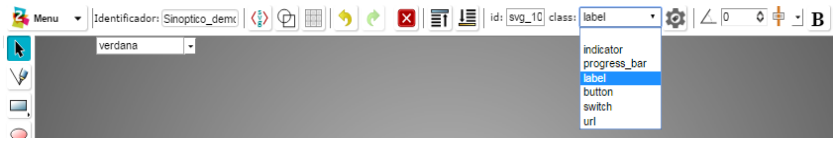
Para añadir imágenes o archivos svg se deberá acceder al menú y seleccionar Importar archivo svg.



Para añadir texto y etiquetas para los valores de los atributos, se deberá ir al menú lateral izquierdo y se seleccionará la opción de Insertar texto, insertando uno a uno todos los elementos de texto que se deseen.



A continuación se añadirán las etiquetas para la visualización de los atributos de la ontología que se quieren mostrar. Para cada una se creará un elemento de texto con un asterisco, y después se seleccionará en el menú superior la clase de tag que se quiere mostrar, para este ejemplo todos serán de clase label.



Una vez seleccionada la clase se seleccionará el botón de **tag properties** para asignar el tag que va a asociar a ese elemento

Para cada elemento de texto de tipo **label**, se deberá seleccionar el **tag** asociado, temperature, acceIX,..

El módulo Scada contiene una ontología (TagMeasures_demoDispositivos), un Thinkp (KpScada_demoDispositivos) y un script (ScadaScript_demoDispositivos) asociados a los sinópticos. Estos elementos son los que permiten la visualización de los datos en tiempo real.

Para este caso se creará un script que lo que haga sea insertar una instancia en la ontología TagMeasures_demoDispositivos cada vez que se inserte en la ontología demoDispositivos_RTFrame. Para la generación del script se deberán tener en cuenta únicamente los campos tagId (que se corresponderá con la identificación del tag) y measure (que se corresponderá con el valor que toma el tag) de la ontología TagMeasures_demoDispositivos. De tal forma que por cada atributo que se quiera mostrar, se insertará una instancia en la ontología del sinóptico.

```
12 /////valores ontología  demoDispositivos_RTFrame ///////
13 def accelX=ontologyJson.demoDispositivos_RTFrame.accelX;
14 def accelY=ontologyJson.demoDispositivos_RTFrame.accelY;
15 def accelZ=ontologyJson.demoDispositivos_RTFrame.accelZ;
16 def temperature =ontologyJson.demoDispositivos_RTFrame.temperature;
17 def humidity=ontologyJson.demoDispositivos_RTFrame.humidity;
18
19
20 String [] valuesNames=["accelX","accelY","accelZ","temperature","humidity"];
21 String [] measures=[accelX,accelY,accelZ,temperature,humidity];
22 pw.println("${valuesNames}/////////////// ${measures}");
23 pw.flush();
24 for(int i=0;i<5;i++){
25     def ontologia="{\"Feed\":{ \"tagId\":\"${valuesNames[i]}\",\"tagType\":\"tagType1\",\"tagSource\":\"ta
26     pw.println(ontologia);
27     pw.flush();
28     def result= apirtdb.insertInBDTR(token,Kp,nombreontologia,ontologia);
29     pw.println(result);
30     pw.flush();
31   }
32 pw.close();
```

Para más detalle sobre la composición de un sinóptico se pueden visitar los siguientes enlaces:

Versión inicial del Módulo Sofia2-SCADA

Vídeo Demostrador Editor SVG

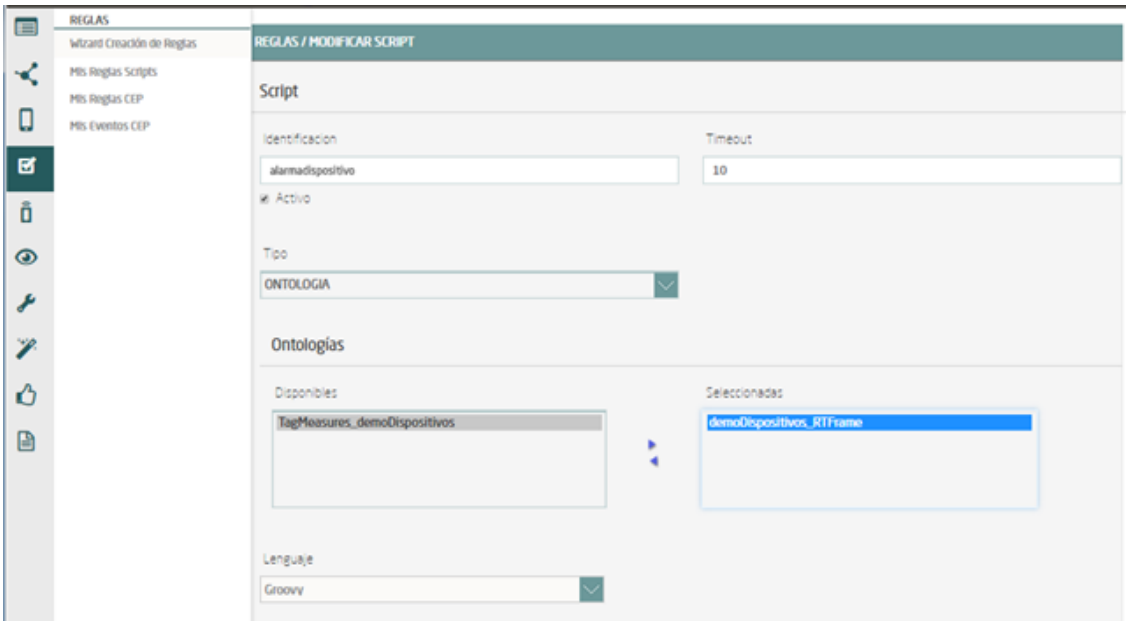
Jugando con los datos

Continuando en la línea del tutorial, si en los apartados anteriores veíamos simplemente como representarlos, tal cual o bajo alguna transformación, de diversas maneras según el uso del dato, en este apartado vamos a configurar una regla, que se ejecutará por cada dato recibido, y vamos a configurar un API para ofrecer una interfaz de acceso a los datos de una manera controlada.

Configurando reglas en tiempo real

En el contexto de nuestra prueba, que consiste en recibir datos de los sensores configurados en un beacon, vamos a controlar que uno de los valores no excede de un valor, y en caso de que esto ocurra, mandaremos un SMS al teléfono del administrador.

Para ello, vamos crear una regla de ontología, que se ejecutará por cada dato que se inserte en nuestra ontología **demoDispositivos_RTFrame** Esta opción la podemos encontrar en el menú **Reglas** submenú **Wizard de Creación de Reglas**



Una vez seleccionados los campos de nombre, timeout (valor obligatorio), Tipo de regla (Ontología), ontología a la que queremos asociar la ejecución del código (en nuestro caso, demoDispositivos_RTFrame), y lenguaje (en nuestro ejemplo seleccionaremos groovy), podemos comenzar a introducir el código.

Para facilitar la estructura del código, localizaremos la condición de la regla en la sección **IF**, que en caso de devolver un **true** continuará ejecutando la sección escrita en la pestaña **THEN** y en caso opuesto, ejecutará la sección codificada en la pestaña **ELSE**. La pestaña **ERROR** contendrá el código para la gestión de los errores de ejecución del script.

En nuestro caso, codificaremos las secciones **IF** y **THEN**

En la sección IF (podéis ver el código en la imagen anterior), se importan las librerías a utilizar, se declaran las variables de instancia de nuestro ThinKP, y su token, cargamos la ontología en la variable ontologyJson, y evaluamos el valor de humedad (si es superior a 30, la evaluación devolverá un true, y el script continuará ejecutando el código de la pestaña THEN)

En la sección THEN utilizaremos un proveedor de SMS para enviar un mensaje indicando la alerta del dispositivo. A través de una conexión httpGET

If Then Else Error

```
def apiutils = new APIUtils();
def apirtdb = new APIRTDB();
def apihttpconnection = new APIHttpConnection();
def apiurlencoder = new java.net.URLEncoder();
def token = "xxxxxxxxxxxxxxxxxxxxxxxx";
def kpInstance = "demoDispositivos_KP:demoDispositivos_instance";
def msg = "ALERTA TEMPERATURA DISPOSITIVO";
def telefonos = "34636903617";
def Correo = "mail@sofia.com";
def Passwd = "XXxxxxx";
def Mensaje = apiurlencoder.encode(msg,"UTF-8");
def url ="http://api.mensatek.com/v5/enviar.php?Destinatarios=${telefonos}&Correo=${Correo}&Passwd=${Passwd}&Mensaje=${
def resp = apihttpconnection.httpGET(url);
```

Así de simple. Una vez guardado y activado el script, éste se ejecutará por cada dato insertado o modificado en la ontología.

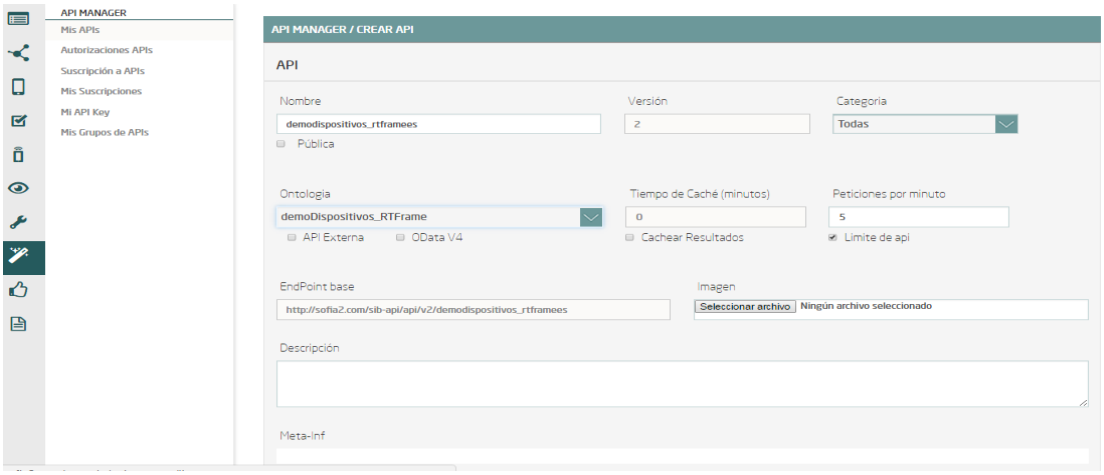
Si vas a jugar con scripting en Sofia2, te interesará saber los siguientes **trucos:**

- Con el cursor en la sección de edición de scripts, pulsa F11 para conseguir que se amplíe a toda la pantalla y así tendrás más espacio para escribir.
- Para verificar si la ejecución del script ha sido correcta, o detalles de cualquier error de ejecución, puedes ir al menú **Herramientas** submenú **visualización de estado de procesos** y verás toda la información necesaria para poner en marcha tu script.

Publicando APIs de acceso a los datos

Ahora vamos a definir interfaces sobre los datos que estamos gestionando. Para ello, una de las opciones disponibles es la configuración de servicios REST a través del API Manager de Sofia2, que nos permitirá definir operaciones de lectura, escritura (*POST*), actualización (*PUT*), borrado (*DELETE*), búsqueda básica o búsquedas avanzadas (*GET*).

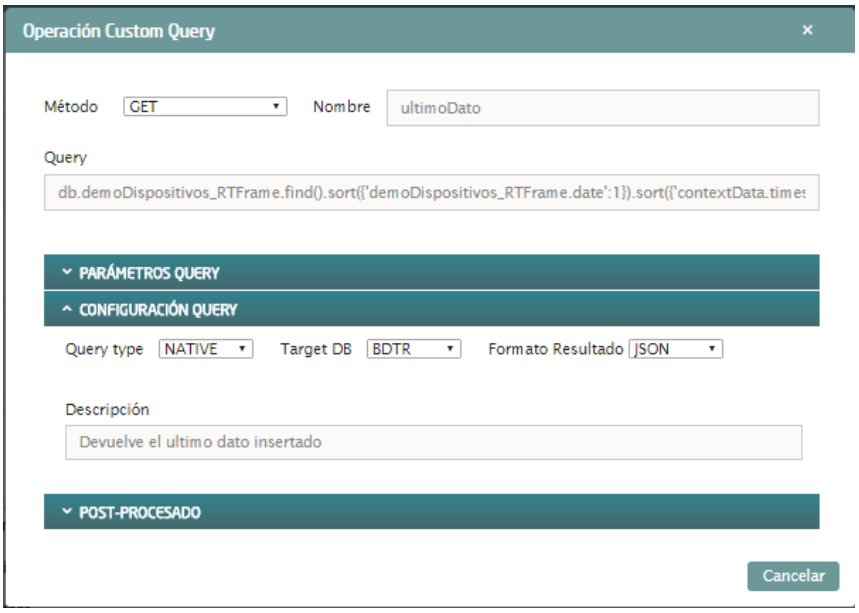
Para crear un conjunto de operaciones sobre la ontología *demoDispositivos_RTFrame* que estamos usando en este ejemplo, accederemos al menú 'Mis APIS', opción 'Crear API'



Por defecto, el formulario propone la configuración de un API externa. En nuestro ejemplo, desmarcaremos esta opción, para poder seleccionar la ontología *demoDispositivos_RTFrame* en el combo, y esta acción nos propondrá un

nombre para el API (*demoDispositivos_RTFramees* que podremos actualizar a lo que prefiramos)

Para habilitar cada una de las operaciones disponibles (GET, POST, PUT, DELETE...), solo tendremos que seleccionarlas en el listado, y añadirle una descripción. En nuestro caso, haremos un ejemplo sencillo de la operación CUSTOM QUERY



Donde la invocación al método 'ultimoDato', lanzará la query:

db.demoDispositivos_RTFrame.find().sort({\'demoDispositivos_RTFrame.date\':1}).sort({\'contextData.timestamp\':-1})

Devolviendo el último dato insertado en la ontología.

Para información más detallada de todas las posibilidades del API Manager en Sofia2, os recomentamos que visitéis los siguientes post del Blog:

- Nuevo ciclo de vida en API Manager
- API Manager: Metodos Custom Query
- Control Throtling en API Manager
- API Manager: Clonado de APIs

Gestión de dispositivos en Sofia2

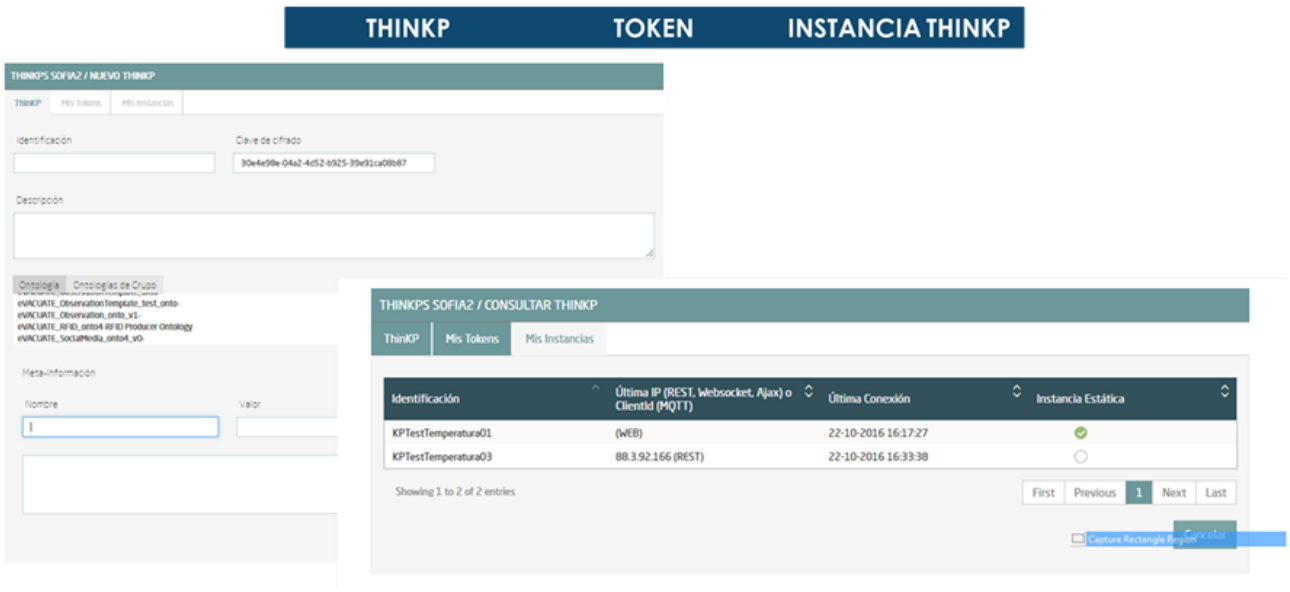
La gestión de los dispositivos conectados es una de las 10 tecnologías más relevantes en el ámbito IoT para los próximos años, según la estimación de Gartner

En este apartado, y a modo de compendio de todas las capacidades presentadas anteriormente, se encarga de presentar las capacidades actuales y futuras de gestión de dispositivos por parte de Sofia2.

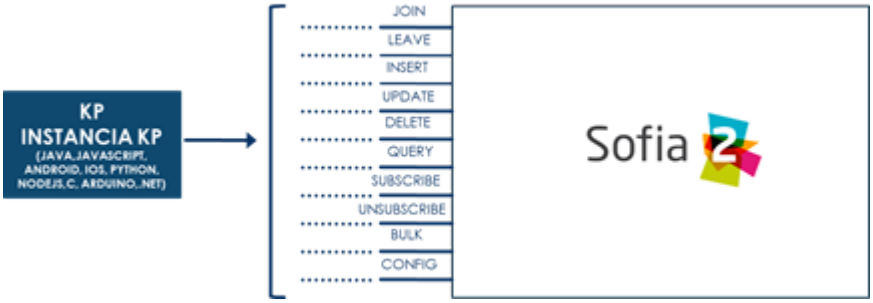
Antes de entrar en materia, un breve repaso de los conceptos que manejamos en Sofia2 para la configuración de las conexiones IoT:

- **Spaces (Proyectos):** Un Space (Proyecto Sofia2) representa un entorno colaborativo virtual donde los usuarios pueden crear sus aplicaciones, por ejemplo creando Things, modelando sus entidades, aplicando algoritmos o creando visualizaciones.
- **Ontología (Entities):** Una Entity (Ontología en terminología Sofia2 / **Thintology**) representa el Modelo de Dominio que maneja una Thing.
  - Las Ontologías se representan en JSON y pueden representar un modelo básico (como si fuera una Tabla) o un modelo complejo con relaciones (como si tuviésemos un conjunto de tablas relacionadas).
  - Cuando un Dispositivo (Thing) envía una medida hablamos de **instology** (Instancia de Ontología).
  - Las Entities pueden crearse de diversas formas: visualmente en un diagrama de clases UML, a través de un esquema JSON o XML, campo a campo o a partir de un CSV/XLS.
- **ThinKP:** Una ThinKP (en terminología Sofia2 hablamos de KP: Knowledge Processor o de ThinKP) representa a cada uno de los elementos que interactúan con la plataforma, bien publicando, bien consumiendo información.
  - Una Thing puede representar desde un dispositivo sencillo (un Arduino o un iBeacon) a un Gateway (una Raspberry) o un Sistema Empresarial (Backend Java u otro).
  - Una Thing puede manejar una o varias Entities (ontologías).
  - Una Thing al ponerse en ejecución crea una **instathing** (Instancia de KP), asociado a una Thing pueden crearse varias Instancias.
  - Todas las comunicaciones con Sofia2 están securizadas. En el caso de las comunicaciones desde una instathing a la plataforma, tendremos un **token** de autenticación que garantizará que la thing conectada está registrada y autorizada para hacer la operación.
- **Asset (Think Type):** Un Asset me permite definir las características estáticas de una Thing. Puede usarse para definir tipos de dispositivos (p. e. farolas en una ciudad o motores en una planta) o hacer una gestión de activos.

Desde el menú ThinKPs SOFIA2, submenú **'Mis ThinKPs'** podremos gestionar el alta, modificación y eliminación de los ThinKPs, sus tokens y las instancias de cada uno de ellos.



Con esta configuración, podremos comunicar nuestras "things" con la plataforma, siguiendo el protocolo SSAP.



Para ello, Sofia2 provee una serie de APIs de desarrollo (disponibles en nuestra web), de tal manera que esta comunicación se pueda implementar tanto en distintos lenguajes de programación, como en distintos protocolos de comunicación (MQTT, rest, websockets. . .)

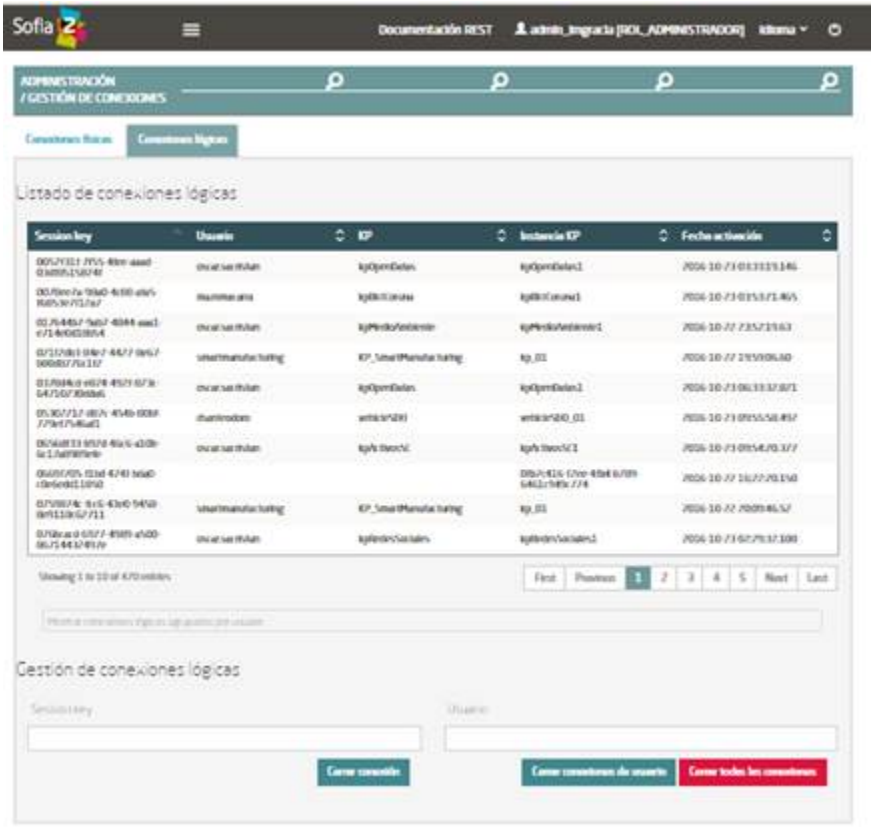
Una vez puesto en marcha nuestro proyecto, podremos controlar la actividad de nuestras conexiones desde varios puntos de vista:

KPs Activos

Desde el menú ThinKP Sofia2, submenú **'ThinKPs conectados'** podremos visualizar las conexiones activas, junto con sus datos (identificación, sessionKey y fecha de activación)

Gestión de conexiones

Por otra parte, desde el menú de Administración, submenú **'Gestión de Conexiones'**, podremos visualizar las conexiones tanto desde el punto de vista físico como lógico, pudiendo hacer búsquedas, y cerrar conexiones o incluso bloquear clientes específicos, tal y como se muestra en la siguiente figura.

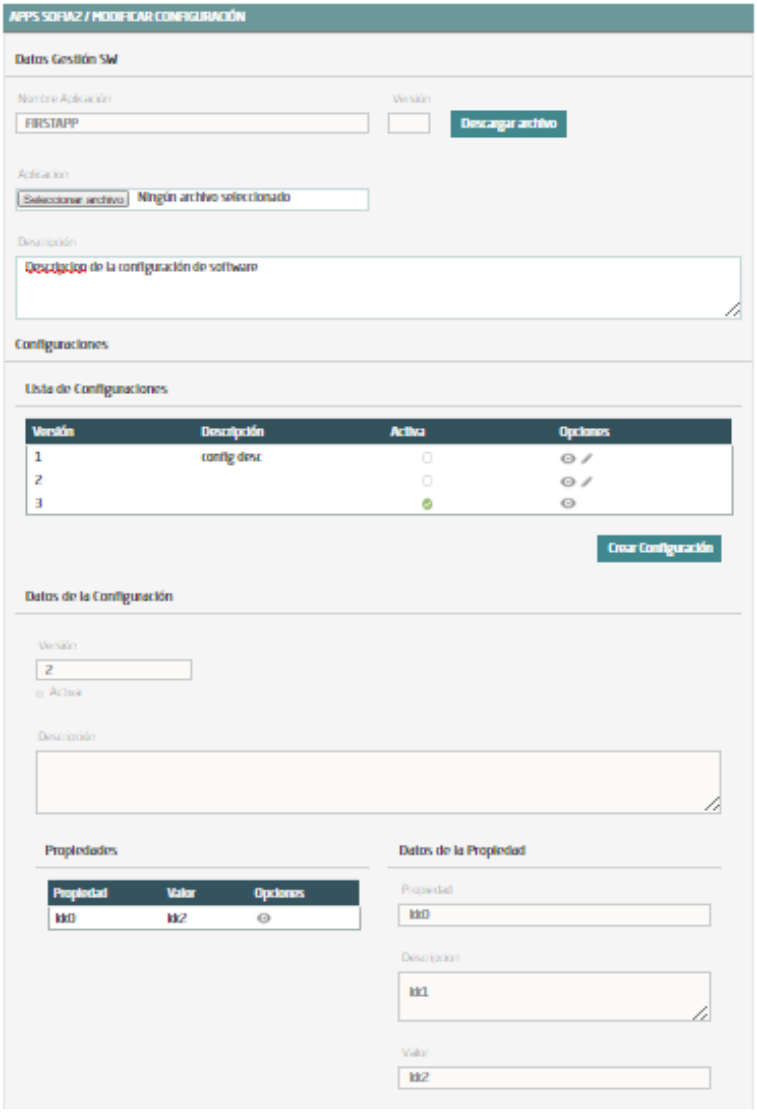


Gestión de Configuraciones SW

Además, podremos controlar las versiones de los clientes desplegados en nuestros things, y su configuración, mediante la gestión de configuraciones, donde podremos asociar SW y parámetros de configuración a nuestros ThinKPs o a instancias de ThinKPs.

De esta manera, si queremos actualizar la versión del SW con la que se conectan nuestros things, actualizaremos la configuración de SW asignada, y la próxima vez que el dispositivo compruebe la versión de SW, se le informará que hay una nueva versión, pudiendo lanzar la descarga y actualización en cliente de manera automática.

Esta funcionalidad es muy útil en escenarios en que tenemos cientos de dispositivos conectados a nuestra plataforma (por ejemplo, una smartCity, o una fábrica), y queremos hacer actualizaciones remotas de todos ellos.



Gestión de assets

A todas estas capacidades, y como funcionalidad adicional, podemos añadir la gestión de los assets (los elementos del mundo real conectados a través de nuestros ThinKPs), con funcionalidades como la geolocalización de cada uno de ellos, categorización y gestión de sus propiedades.

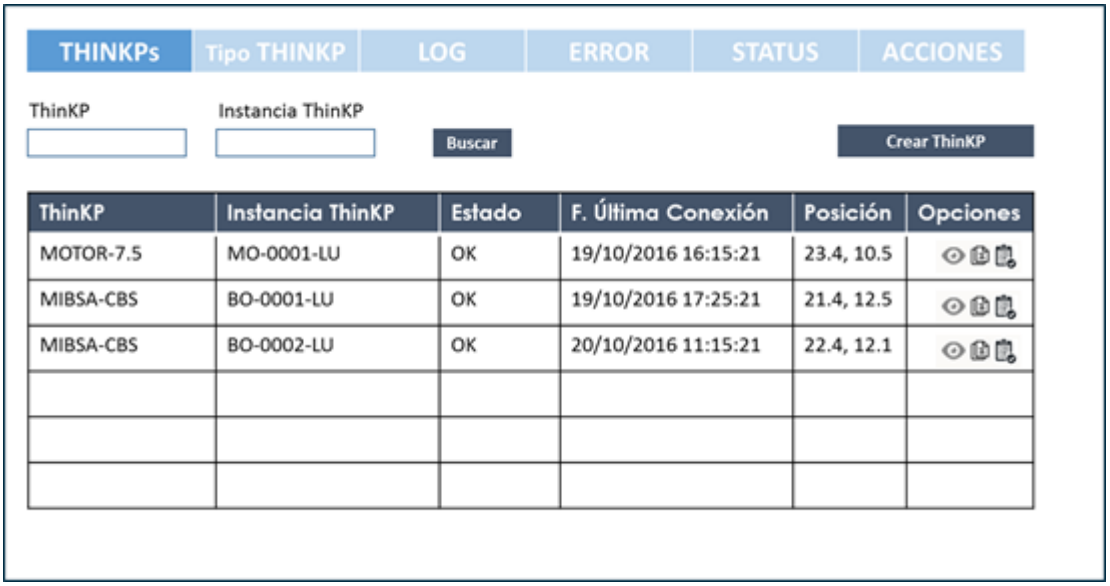
En resumen, Sofia2 cuenta con un completo conjunto de funcionalidades para dar cobertura a las necesidades de gestión de los dispositivos conectados, tanto para controlar su actividad como para asegurar su correcta configuración.

Lo que viene en la próxima versión

En cualquier caso, y siguiendo nuestro espíritu de mejora continua, estamos preparando novedades en las próximas versiones. Entre otras, estamos trabajando en una **gestión avanzada de dispositivos (ThinKPs)** que permitirá:

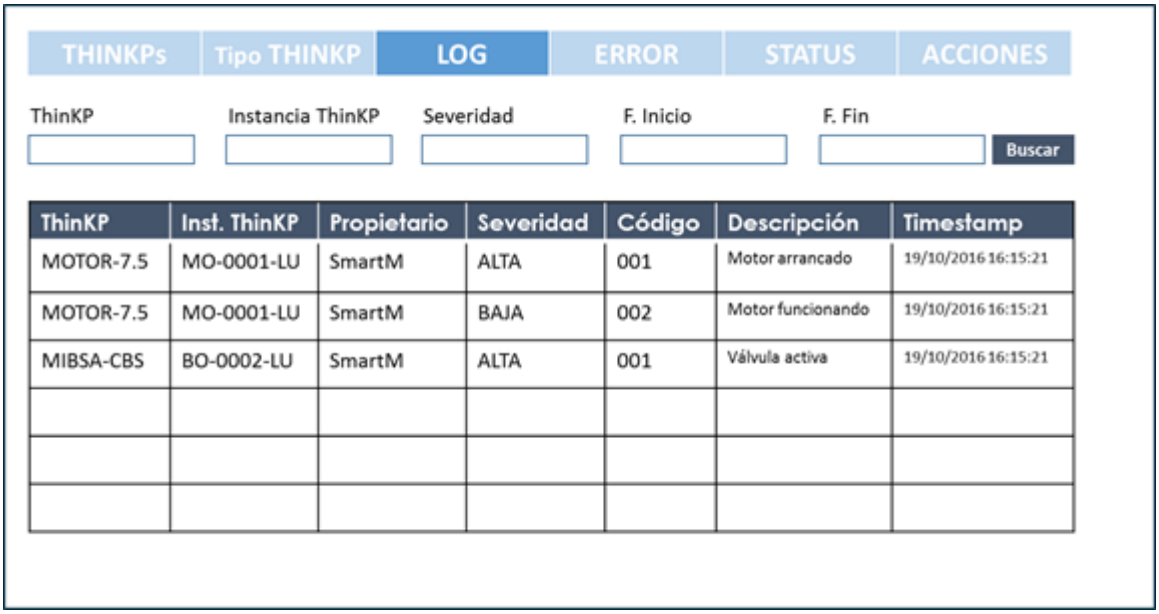
- Obtener una visión centralizada del estado de todos los dispositivos y componentes conectados a la plataforma (desde sus distintas configuraciones, localización, mensajes y errores producidos hasta el estado de sus componentes HW). Para ello se incorporarán nuevos tipos de mensaje de comunicación:
  - Error
  - Logs
  - Location
  - Status
- Comunicación y actuación directas desde la plataforma con cualquiera de los thinKP conectados, para obtener información y efectuar operaciones remotas.
- Control de todas las funcionalidades (las nuevas y las ya existentes) desde un único punto de la consola, para lo que se reestructurará en una única pantalla, con las siguientes pestañas (incluimos los drafts conceptuales!):

Gestión de ThinKPs: Para la gestión de los thinKPs: Visualización, búsqueda, modificación y borrado.

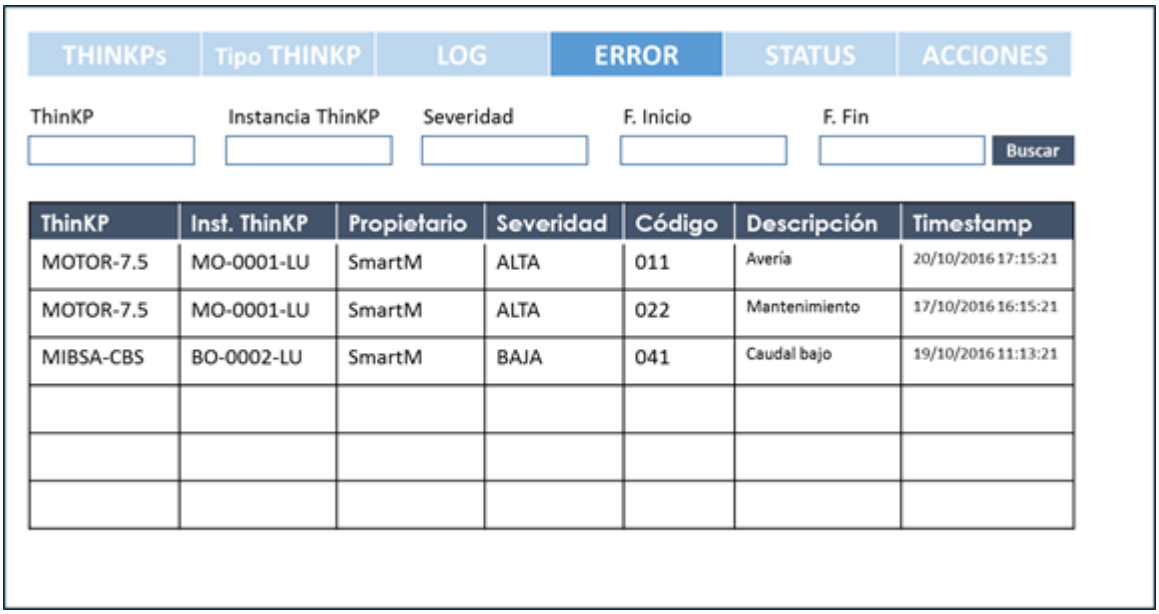


Tipo de ThinKPs: Para la gestión de lo que ahora se presenta como assets. Podremos buscar y gestionar toda esta información relacionándola con los thinKPs.

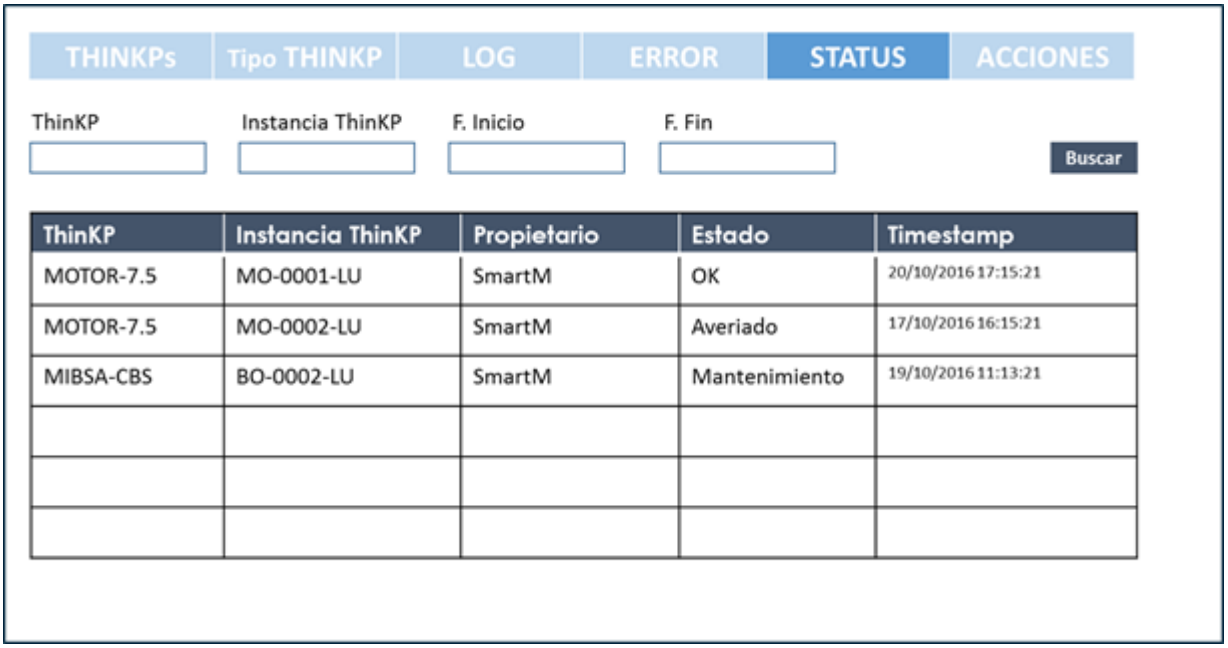
Logs: La plataforma recibirá todas las trazas que se consideren relevantes desde cada uno de los dispositivos, permitiendo tener un único punto centralizado de control de logs de todos los thinKPs integrados, con distintos criterios de búsqueda.



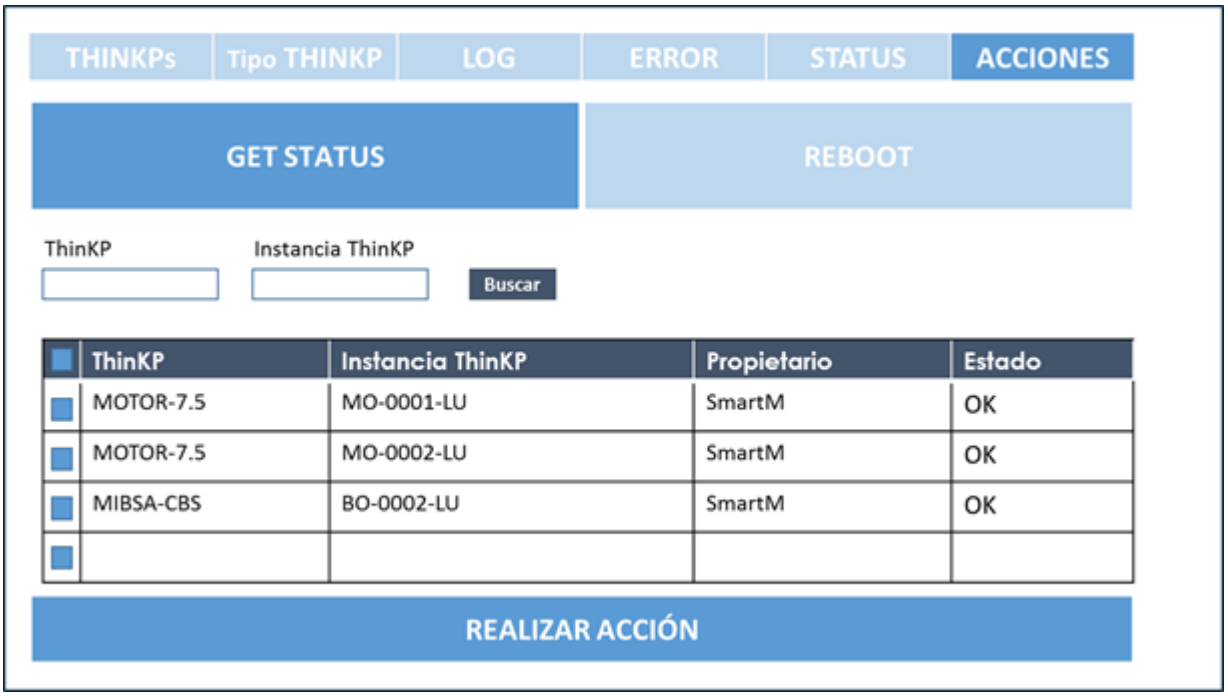
Error y estado: Igualmente, tendremos un único punto desde el que controlar y buscar todos los mensajes de error identificados en los dispositivos.



Igualmente para el estado de cada ThinKP conectada.



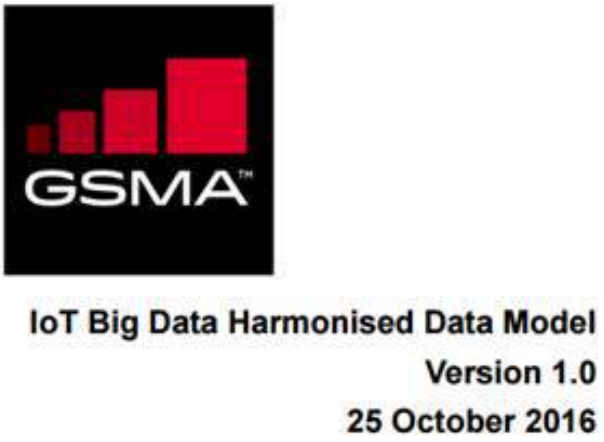
Lanzar acciones: Por último, podremos lanzar acciones sobre uno o varios dispositivos a la vez (búsqueda y multiselección), tales como actualizar el estado del dispositivo o hacer un reinicio en remoto.



DATA MODEL FIWARE/GSMA EN SOFIA2

Introducción

La asociación GSMA (asociación de operadores móviles) está trabajando en un IoT Big Data Harmonised Data Model,



que define estas entidades:

AgriCrop	AirQualityObserved	MachineModel	VehicleType
AgriGreenHouse	Building	MachineOperation	WaterQualityObserved
AgriParcel	BuildingOperation	PointOfInterest	WeatherForecast
AgriParcelOperation	BuildingType	Road	WeatherObserved
AgriParcelRecord	Device	RoadSegment	
AgriPest	DeviceModel	Subscriber	
AgriProduct	DeviceOperation	SubscriptionService	
AgriProductType	EnvironmentObserved	Vehicle	
AgriSoil	Machine	VehicleFault	

Por otro lado en la iniciativa FIWARE se han inspirado en el Data Model GSMA para crear los FIWARE Data Models, donde además se han seleccionado un conjunto de Entidades sobre estas de GSMA:

- **Alarms**. Events related to risk or warning conditions which require action taking.
- **Parks & Gardens**. Data models intended to make an efficient, effective and sustainable management of green areas.
- **Environment**. Enable to monitor air quality and other environmental conditions for a healthier living.
- **Point of Interest**. Specific point locations that someone may find useful or interesting. For instance, weather stations, touristic landmarks, etc.
- **Civic Issue tracking**. Data models for civic issue tracking interoperable with the de-facto standard Open311.
- **Street Lighting**. Modeling street lights and all their controlling equipment towards energy-efficient and effective urban illuminance.
- **Device**. IoT devices (sensors, actuators, wearables, etc.) with their characteristics and dynamic status.
- **Transportation**. Transportation data models for smart mobility and efficient management of municipal services.
- **Indicators**. Key performance indicators intended to measure the success of an organization or of a particular activity in which it engages.

- **Waste Management**. Enable efficient, recycling friendly, municipal or industrial waste management using containers, litters, etc.
- **Parking**. Real time and static parking data (on street and off street) interoperable with the EU standard DATEX II.
- **Weather**. Weather observed, weather forecasted or warnings about potential extreme weather conditions.

Entidades Sofia2

La Plataforma sofia2 gestiona dos tipos de entidades:

- **Assets**. Un asset es todo elemento (físico o virtual) capaz de generar o consumir información de carácter sensorial y gestionarla a través de la Plataforma SOFIA2
- **Ontologías**. Los assets generan información y dicha información se modela por medio de ontologías (JSON) en la Plataforma.

(Recomendamos al respecto leer el documento Gobierno de Ontologías o el conjunto de posts al respecto).

Los Data Models GSMA y FIWARE se definen en JSON por lo que **su representación como Ontología Sofia2 es inmediata**. Para ello, haremos uso de las plantillas predefinidas que soportan los anteriores modelos.

Plantillas (Modelo GSMA/FIWARE)

En Sofia2 se soportan las entidades GSMA vía Plantillas (una plantilla sirve para crear ontologías partiendo de una definición):

Esquemas JSON (JSON-SChema)

Para la definición de Plantillas se utiliza una simplificación del estándar de datos AMON (http://amee.github.io/AMON/). Estas, se basan en el formato JSON-Schema, el cual ofrece un contrato para definir los datos requeridos para una aplicación dada y la forma de interactuar con él. Los tipos de datos que encontraremos en un JSON-Schema son:

- string : Cadena de texto
- number: Numérico
- object: Objeto
- char: Caracteres Unicode válidos
- array: Colección de valore
- null: Nulo
- boolean: Valores true o false

Para hacernos una idea veamos un ejemplo de un esquema JSON sencillo:

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "Product",
    "description": "A product from Acme's catalog",
    "type": "object",
    "properties": {
        "id": {
            "description": "The unique identifier for a product",
            "type": "integer"
        },
        "name": {
            "description": "Name of the product",
            "type": "string"
        },
        "price": {
            "type": "number",
            "minimum": 0,
            "exclusiveMinimum": true
        }
    },
    "required": ["id", "name", "price"]
}
```

Que validaría como válidos JSONs como este:

```
{
    "id": 1,
    "name": "A green door",
    "price": 12.50,
    "tags": ["home", "green"]
}
```

Y como inválido este por no tener el atributo price:

```
{
    "id": 1,
    "name": "A green door",
    "tags": ["home", "green"]
}
```

Atributos de un esquema JSON

Podemos ver la referencia completa de la especificación JSON aquí: http://json-schema.org/latest/json-schema-core.html

```
{
"$schema": "http://json-schema.org/draft-04/schema#",
"title": "Product",
"description": "A product from Acme's catalog",
"type": "object",
"properties": {
    ...
    ...
    ...
    ...
},
"required": ["id", "name", "price"]
}
```

Los atributos más utilizados en un esquema JSON son:

- **"$schema"**: Nos permite indicar la versión del Schema JSON que queremos usar: 0.4 o 0.3, SOFIA2 se apoya en la versión 0.4 (http://json-schema.org/draft-04/schema#)
- **"title"**: indicar un título con el que identificar el esquema.
- **"description"**: Se puede utilizar este atributo para incluir una descripción de lo que va a representar el esquema JSON.
- **"type"**: Para indicar el tipo que va a representar el esquema.
- **"properties"**: Este atributo es un objeto con las definiciones de propiedades que definen los valores estáticos de una instancia de objeto. Es una lista no ordenada de propiedades. Los nombres de las propiedades se deben cumplir y el valor de las propiedades se definen a partir de un esquema, que debe cumplirse también.
- **"patternProperties"**: Este atributo es un objeto con las definiciones de propiedades que definen los valores de una instancia de objeto. Es una lista desordenada de propiedades. Los nombres de las propiedades son patrones de expresiones regulares, las instancias de las propiedades deben cumplir con el patrón definido y el valor de la propiedad con el esquema que define esa propiedad.
- **"additionalProperties"**: Permite indicar si la instancia JSON puede contener propiedades que no hayan sido definidas en el esquema. Tiene dos posibles valores (true o false), para indicar si se admite cualquier propiedad o no. Si no se añade la propiedad, se podrá incluir cualquier otra propiedad.
- **"required"**: Permite indicar todas las propiedades que son obligatorias para una instancia JSON y que como mínimo debe incluir. Las propiedades se incluirán entre corchetes y separadas por el carácter ",".

  (Este propiedad es obligatoria incluirla en el esquema).
- **"$ref"**: Define una URI de un esquema que contienen la completa representación para esa propiedad.

Para ampliar la información sobre cómo funcionan los JSON-Schema podemos consultar el documento Modelado de Ontologías

Veamos a continuación el esquema que siguen algunas de las plantillas existentes en Sofia2 que soportan el modelo GSMA:

GSMA-Air Quality Observed

Con esta plantilla caracterizaremos la observación de las condiciones de calidad del aire en un determinado lugar y tiempo.

El JSON-Schema que seguiremos para definir el modelo de datos es el siguiente:

{

"$schema": "http://json-schema.org/draft-04/schema#",

"title": "Air Quality Observed",

"type": "object",

"required": [

“id”,
“type”,
“address”,
“dateObserved”,
“location”,
“source”,
“precipitation”,
“relativeHumidity”,
“temperature”,
“windDirection”,
“windSpeed”,
“measurand”,
“CO”,
“NO”,
“NO2”,
“NOx”,
“SO2”,
“refPointOfInterest”
],
“properties”: {
“id”: {
“type”: “string”
},
“type”: {
“type”: “string”
},
“address”: {
“type”: “object”,
“properties”: {
“addressCountry”: {
“type”: “string”
},
“addressLocality”: {
“type”: “string”
},
“streetAddress”: {

“type”: “string”
}
},
“required”: [
“addressCountry”,
“addressLocality”,
“streetAddress”
]
},
“dateObserved”: {
“type”: “string”
},
“location”: {
“type”: “object”,
“properties”: {
“type”: {
“type”: “string”
},
“coordinates”: {
“type”: “array”,
“items”: {
“type”: “number”
}
}
},
“required”: [
“type”,
“coordinates”
]
},
“source”: {
“type”: “string”
},
“precipitation”: {
“type”: “integer”
},

“relativeHumidity”: {
“type”: “number”
},
“temperature”: {
“type”: “number”
},
“windDirection”: {
“type”: “integer”
},
“windSpeed”: {
“type”: “number”
},
“measurand”: {
“type”: “array”,
“items”: {
“type”: “string”
}
},
“CO”: {
“type”: “integer”
},
“NO”: {
“type”: “integer”
},
“NO2”: {
“type”: “integer”
},
“NOx”: {
“type”: “integer”
},
“SO2”: {
“type”: “integer”
},
“refPointOfInterest”: {
“type”: “string”
}

}

}

Ejemplo de Uso:


```
{
  "id": "Madrid-AmbientObserved-28079004-2016-03-15T11:00:00",
  "type": "AirQualityObserved",
  "address": {
    "addressCountry": "ES",
    "addressLocality": "Madrid",
    "streetAddress": "Plaza de España"
  },
  "dateObserved": "2016-03-15T11:00:00/2016-03-15T12:00:00",
  "location": {
    "type": "Point",
    "coordinates": [-3.712247222222222, 40.423852777777775]
  },
  "source": "http://datos.madrid.es",
  "precipitation": 0,
  "relativeHumidity": 0.54,
  "temperature": 12.2,
  "windDirection": 186,
  "windSpeed": 0.64,
  "measurand": [
     "CO, 500, GP, Carbon Monoxide",
     "NO, 45, GQ, Nitrogen Monoxide",
     "NO2, 69, GQ, Nitrogen Dioxide",
     "NOx, 139, GQ, Nitrogen oxides",
     "SO2, 11, GQ, Sulfur Dioxide"
  ],
  "CO": 500,
  "NO": 45,
  "NO2": 69,
  "NOx": 139,
  "SO2": 11,
  "refPointOfInterest": "28079004-Pza. de España"
}
```


Ver en FIWARE-DATAMODELS

GSMA-Air Quality Station

Con esta plantilla caracterizaremos un punto de interés: Una Estación de Calidad del Aire.

El JSON-Schema que seguiremos para definir el modelo de datos es el siguiente:

{

“$schema”: “http://json-schema.org/draft-04/schema#”,
“title”: “Air Quality Station”,
“type”: “object”,
“properties”: {
“address”: {
“type”: “object”,
“properties”: {
“addressCountry”: {
“type”: “string”
},
“addressLocality”: {
“type”: “string”
},
“streetAddress”: {
“type”: “string”
}
},
“required”: [
“addressCountry”,
“addressLocality”,
“streetAddress”
]
},
“category”: {
“type”: “string”
},
“location”: {
“type”: “object”,
“properties”: {
“type”: {
“type”: “string”
},
“coordinates”: {
“type”: “array”,
“items”: {
“type”: “number”

}
}
},
“required”: [
“type”,
“coordinates”
]
},
“name”: {
“type”: “string”
},
“source”: {
“type”: “string”
},
“type”: {
“type”: “string”
},
“id”: {
“type”: “string”
}
},
“required”: [
“address”,
“category”,
“location”,
“name”,
“source”,
“type”,
“id”
]
}

Ejemplo de Uso:

```
{
  "address": {
    "addressCountry": "ES",
    "addressLocality": "Madrid",
    "streetAddress": "Plaza de España"
  },
  "category": "AirQualityStation",
  "location": {
    "type": "Point",
    "coordinates": [
      -3.712247222222222,
      40.423852777777775
    ]
  },
  "name": "Pza. de España",
  "source": "http://datos.madrid.es",
  "type": "PointOfInterest",
  "id": "AirQualityStation-ES-Madrid-004"
}
```


Ver en FIWARE-DATAMODELS

GSMA-Device

Con esta plantilla caracterizaremos un Device (Dispositivo). Una Estación de Calidad del Aire. Un dispositivo es un objeto tangible que contiene alguna lógica y es productor y/o consumidor de datos. Siempre se supone que un dispositivo es capaz de comunicarse electrónicamente a través de una red.

El JSON-Schema que seguiremos para definir el modelo de datos es el siguiente:

{

“$schema”: “http://json-schema.org/draft-04/schema#”,

“title”: “Device”,

“type”: “object”,

“properties”: {

“id”: {

“type”: “string”

},

“type”: {

“type”: “string”

},

“category”: {

“type”: “array”,
“items”: {
“type”: “string”
}
},
“controlledProperty”: {
“type”: “array”,
“items”: {
“type”: “string”
}
},
“controlledAsset”: {
“type”: “array”,
“items”: {
“type”: “string”
}
},
“ipAddress”: {
“type”: “string”
},
“mcc”: {
“type”: “string”
},
“mnc”: {
“type”: “string”
},
“batteryLevel”: {
“type”: “number”
},
“serialNumer”: {
“type”: “string”
},
“refDeviceModel”: {
“type”: “string”
},
“value”: {

“type”: “string”

},

“deviceState”: {

“type”: “string”

},

“dateFirstUsed”: {

“type”: “string”

}

},

“required”: [

“id”,

“type”,

“category”,

“controlledProperty”,

“controlledAsset”,

“ipAddress”,

“mcc”,

“mnc”,

“batteryLevel”,

“serialNumer”,

“refDeviceModel”,

“value”,

“deviceState”,

“dateFirstUsed”

]

}

Ejemplo de Uso:

```
{
  "id": "device-9845A",
  "type": "Device",
  "category": ["sensor"],
  "controlledProperty": ["fillingLevel","temperature"],
  "controlledAsset": ["wastecontainer-Osuna-100"],
  "ipAddress": "192.14.56.78",
  "mcc": "214",
  "mnc": "07",
  "batteryLevel": 0.75,
  "serialNumer": "9845A",
  "refDeviceModel": "myDevice-wastecontainer-sensor-345",
  "value": "l=0.22;t=21.2",
  "deviceState": "ok",
  "dateFirstUsed": "2014-09-11",
}
```


Ver en FIWARE-DATAMODELS

GSMA-Key Performance Indicator

Con esta plantilla caracterizaremos un Key Performance Indicator (KPI), o lo que es lo mismo, un Indicador Clave de Rendimiento, un tipo de medición del desempeño. Los KPIs evalúan el éxito de una organización o de una actividad particular en la que se involucra.

El JSON-Schema que seguiremos para definir el modelo de datos es el siguiente:

{

“$schema”: “http://json-schema.org/draft-04/schema#”,

“title”: “Key Performance Indicator”,

“type”: “object”,

“properties”: {

“id”: {

“type”: “string”

},

“type”: {

“type”: “string”

},

“name”: {

“type”: “string”

},

“description”: {

“type”: “string”
},
“category”: {
“type”: “array”,
“items”: {
“type”: “string”
}
},
“organization”: {
“type”: “object”,
“properties”: {
“name”: {
“type”: “string”
}
},
“required”: [
“name”
]
},
“provider”: {
“type”: “object”,
“properties”: {
“name”: {
“type”: “string”
}
},
“required”: [
“name”
]
},
“kpiValue”: {
“type”: “integer”
},
“currentStanding”: {
“type”: “string”
},

“calculationPeriod”: {
“type”: “object”,
“properties”: {
“from”: {
“type”: “string”
},
“to”: {
“type”: “string”
}
},
“required”: [
“from”,
“to”
]
},
“calculationMethod”: {
“type”: “string”
},
“calculationFrequency”: {
“type”: “string”
},
“dateModified”: {
“type”: “string”
},
“dateNextCalculation”: {
“type”: “string”
},
“address”: {
“type”: “object”,
“properties”: {
“addressLocality”: {
“type”: “string”
},
“addressCountry”: {
“type”: “string”
}

},
“required”: [
“addressLocality”,
“addressCountry”
]
}
},
“required”: [
“id”,
“type”,
“name”,
“description”,
“category”,
“organization”,
“provider”,
“kpiValue”,
“currentStanding”,
“calculationPeriod”,
“calculationMethod”,
“calculationFrequency”,
“dateModified”,
“dateNextCalculation”,
“address”
]
}

Ejemplo de Uso:

```
{
  "id": "kpi-2016-Ciudad-containers-faults",
  "type": "KeyPerformanceIndicator",
  "name": "Incidencias-Contenedores-Mensual",
  "description": "Number of incidences raised on containers per month",
  "category": ["quantitative"],
  "organization": {
     "name": "Ayuntamiento de Ciudad"
  },
  "provider": {
     "name": "Cleaning Service Provider S.A."
  },
  "kpiValue": 20,
  "currentStanding": "good",
  "calculationPeriod": {
    "from": "2016-06-01",
    "to":   "2016-06-30",
  },
  "calculationMethod": "automatic",
  "calculationFrequency": "monthly",
  "dateModified": "2016-06-29T15:59:09.224Z",
  "dateNextCalculation": "2016-07-31",
  "address": {
    "addressLocality": "Ciudad",
    "addressCountry": "ESP"
  }
}
```


Ver en FIWARE-DATAMODELS

GSMA-Parking Access

Con esta plantilla caracterizaremos un punto de acceso a un parking, normalmente un parking fuera de la calle.

El JSON-Schema que seguiremos para definir el modelo de datos es el siguiente:

{

“$schema”: “http://json-schema.org/draft-04/schema#”,

“title”: “Parking Access”,

“type”: “object”,

“properties”: {

“id”: {

“type”: “string”

},

“type”: {

“type”: “string”

},
“name”: {
“type”: “string”
},
“location”: {
“type”: “object”,
“properties”: {
“coordinates”: {
“type”: “array”,
“items”: {
“type”: “number”
}
},
“type”: {
“type”: “string”
}
},
“required”: [
“coordinates”,
“type”
]
},
“category”: {
“type”: “array”,
“items”: {
“type”: “string”
}
},
“refOffStreetParking”: {
“type”: “string”
},
“features”: {
“type”: “array”,
“items”: {
“type”: “string”
}

}

},

“required”: [

“id”,

“type”,

“name”,

“location”,

“category”,

“refOffStreetParking”,

“features”

]

}

Ejemplo de Uso:

Ver en FIWARE-DATAMODELS

GSMA-Streetlight

Con esta plantilla caracterizaremos un punto de iluminación urbano.

El JSON-Schema que seguiremos para definir el modelo de datos es el siguiente:

{

“$schema”: “http://json-schema.org/draft-04/schema#”,

“title”: “Streetlight”,

“type”: “object”,

“properties”: {

“id”: {

“type”: “string”
},
“type”: {
“type”: “string”
},
“location”: {
“type”: “object”,
“properties”: {
“type”: {
“type”: “string”
},
“coordinates”: {
“type”: “array”,
“items”: {
“type”: “number”
}
}
},
“required”: [
“type”,
“coordinates”
]
},
“areaServed”: {
“type”: “string”
},
“status”: {
“type”: “string”
},
“refStreetlightGroup”: {
“type”: “string”
},
“refStreetlightModel”: {
“type”: “string”
},
“circuit”: {

“type”: “string”
},
“lanternHeight”: {
“type”: “integer”
},
“locationCategory”: {
“type”: “string”
},
“powerState”: {
“type”: “string”
},
“controllingMethod”: {
“type”: “string”
},
“dateLastLampChange”: {
“type”: “string”
}
},
“required”: [
“id”,
“type”,
“location”,
“areaServed”,
“status”,
“refStreetlightGroup”,
“refStreetlightModel”,
“circuit”,
“lanternHeight”,
“locationCategory”,
“powerState”,
“controllingMethod”,
“dateLastLampChange”
]
}

Ejemplo de Uso:

```
{
  "id": "streetlight:guadalajara:4567",
  "type": "Streetlight",
  "location": {
    "type": "Point",
    "coordinates": [  -3.164485591715449, 40.62785133667262 ]
  },
  "areaServed": "Roundabouts city entrance",
  "status": "ok",
  "refStreetlightGroup": "streetlightgroup:G345",
  "refStreetlightModel": "streetlightmodel:STEEL_Tubular_10m",
  "circuit": "C-456-A467",
  "lanternHeight": 10,
  "locationCategory" : "centralIsland",
  "powerState": "off",
  "controllingMethod": "individual",
  "dateLastLampChange": "2016-07-08T08:02:21.753Z"
}
```


Ver en FIWARE-DATAMODELS

GSMA-Weather Observed

Con esta plantilla caracterizaremos la observación de las condiciones climáticas en un lugar y tiempo determinados.

El JSON-Schema que seguiremos para definir el modelo de datos es el siguiente:

{

“$schema”: “http://json-schema.org/draft-04/schema#”,

“title”: “Weather Observed”,

“type”: “object”,

“properties”: {

“id”: {

“type”: “string”

},

“type”: {

“type”: “string”

},

“address”: {

“type”: “object”,

“properties”: {

“addressLocality”: {

“type”: “string”

},
“addressCountry”: {
“type”: “string”
}
},
“required”: [
“addressLocality”,
“addressCountry”
]
},
“atmosfericPressure”: {
“type”: “number”
},
“dataProvider”: {
“type”: “string”
},
“dateObserved”: {
“type”: “string”
},
“location”: {
“type”: “object”,
“properties”: {
“type”: {
“type”: “string”
},
“coordinates”: {
“type”: “array”,
“items”: {
“type”: “number”
}
}
},
“required”: [
“type”,
“coordinates”
]

},
“precipitation”: {
“type”: “integer”
},
“pressureTendency”: {
“type”: “number”
},
“relativeHumidity”: {
“type”: “integer”
},
“source”: {
“type”: “string”
},
“stationCode”: {
“type”: “string”
},
“stationName”: {
“type”: “string”
},
“temperature”: {
“type”: “number”
},
“windDirection”: {
“type”: “integer”
},
“windSpeed”: {
“type”: “integer”
}
},
“required”: [
“id”,
“type”,
“address”,
“atmosfericPressure”,
“dataProvider”,
“dateObserved”,

“location”,

“precipitation”,

“pressureTendency”,

“relativeHumidity”,

“source”,

“stationCode”,

“stationName”,

“temperature”,

“windDirection”,

“windSpeed”

]

}

Ejemplo de Uso:

```
{
            "id": "Spain-WeatherObserved-2422-2016-11-30T08:00:00",
            "type": "WeatherObserved",
            "address":
            {
                "addressLocality": "Valladolid",
                "addressCountry": "ES"
            },
            "barometricPressure": 938.9,
            "dataProvider": "TEF",
            "dateObserved": "2016-11-30T07:00:00.00Z",
            "location":
            {
                "type": "Point",
                "coordinates":
                [
                    -4.754444444,
                    41.640833333
                ]
            },
            "precipitation": 0,
            "pressureTendency": 0.5,
            "relativeHumidity": 1,
            "source": "http://www.aemet.es",
            "stationCode": "2422",
            "stationName": "Valladolid",
            "temperature": 3.3,
            "windDirection": -45,
            "windSpeed": 2
}
```


Ver en FIWARE-DATAMODELS

GSMA-Weather Station

Con esta plantilla caracterizaremos el punto de interés: Estación meteorológica.

El JSON-Schema que seguiremos para definir el modelo de datos es el siguiente:

{

“$schema”: “http://json-schema.org/draft-04/schema#”,

“title”: “Weather Station”,

“type”: “object”,

“properties”: {

“category”: {

“type”: “string”

},
“location”: {
“type”: “object”,
“properties”: {
“type”: {
“type”: “string”
},
“coordinates”: {
“type”: “array”,
“items”: {
“type”: “number”
}
}
},
“required”: [
“type”,
“coordinates”
]
},
“name”: {
“type”: “string”
},
“postalAddress”: {
“type”: “object”,
“properties”: {
“addressCountry”: {
“type”: “string”
},
“addressLocality”: {
“type”: “string”
},
“addressRegion”: {
“type”: “string”
}
},
“required”: [

“addressCountry”,
“addressLocality”,
“addressRegion”
]
},
“source”: {
“type”: “string”
},
“type”: {
“type”: “string”
},
“id”: {
“type”: “string”
}
},
“required”: [
“category”,
“location”,
“name”,
“postalAddress”,
“source”,
“type”,
“id”
]
}

Ejemplo de Uso:

```
{
  "category": "WeatherStation",
  "location": {
    "type": "Point",
    "coordinates": [
      -7.684722222222222,
      43.78611111111111
    ]
  },
  "name": "Estaca de Bares",
  "postalAddress": {
    "addressCountry": "ES",
    "addressLocality": "Mañón",
    "addressRegion": "A Coruña"
  },
  "source": "http://aemet.es",
  "type": "PointOfInterest",
  "id": "WeatherStation-ES-1351"
}
```


Ver en FIWARE-DATAMODELS

HANDS ON

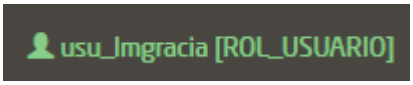
5.1. Creación de Ontologías (Modelo GSMA/FIWARE)

A continuación veremos cómo Sofia2 permite trabajar con estas entidades. Pongamos el ejemplo de la entidad **WeatherObserved** :

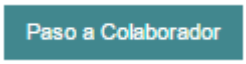
Su aspecto es este:

```
{
        "id": "Spain-WeatherObserved-2422-2016-11-30T08:00:00",
        "type": "WeatherObserved",
        "address":
        {
            "addressLocality": "Valladolid",
            "addressCountry": "ES"
        },
        "atmosfericPressure": 938.9,
        "dataProvider": "TEF",
        "dateObserved": "2016-11-30T07:00:00.00Z",
        "location":
        {
            "type": "Point",
            "coordinates":
            [
                -4.754444444,
                41.640833333
            ]
        },
        "precipitation": 0,
        "pressureTendency": 0.5,
        "relativeHumidity": 1,
        "source": "http://www.aemet.es",
        "stationCode": "2422",
        "stationName": "Valladolid",
        "temperature": 3.3,
        "windDirection": -45,
        "windSpeed": 2
}
```


1. Comenzaremos por acceder al Panel de Control de Sofia2. Para ello podremos crear un usuario o acceder desde aquí.

Una vez hecho el LOGIN, si nuestro rol es USUARIO, 

deberé solicitar el



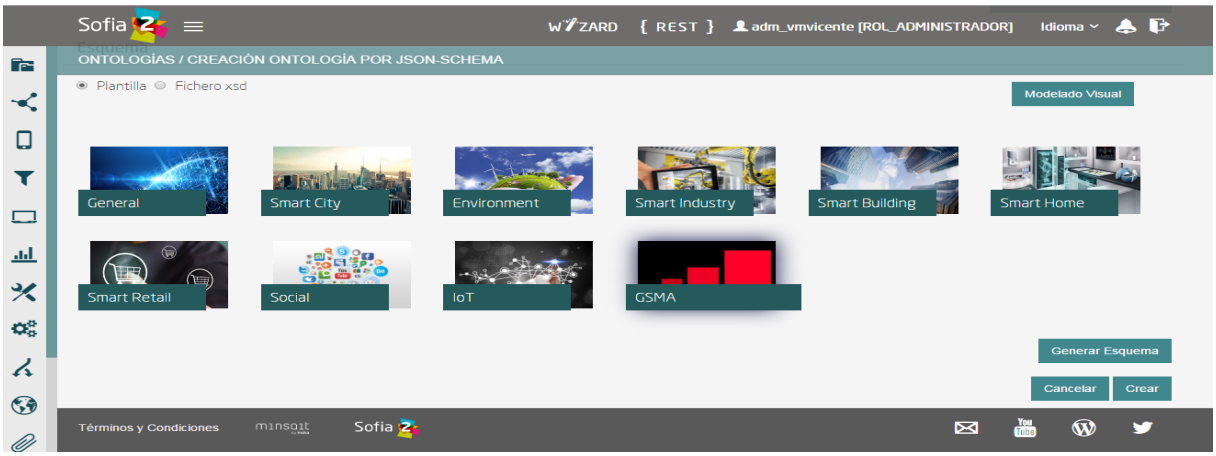
para poder crear Ontologías.

2. Una vez mi usuario tiene rol COLABORADOR podré crear la Ontología que representa la Entidad WeatherObserved.

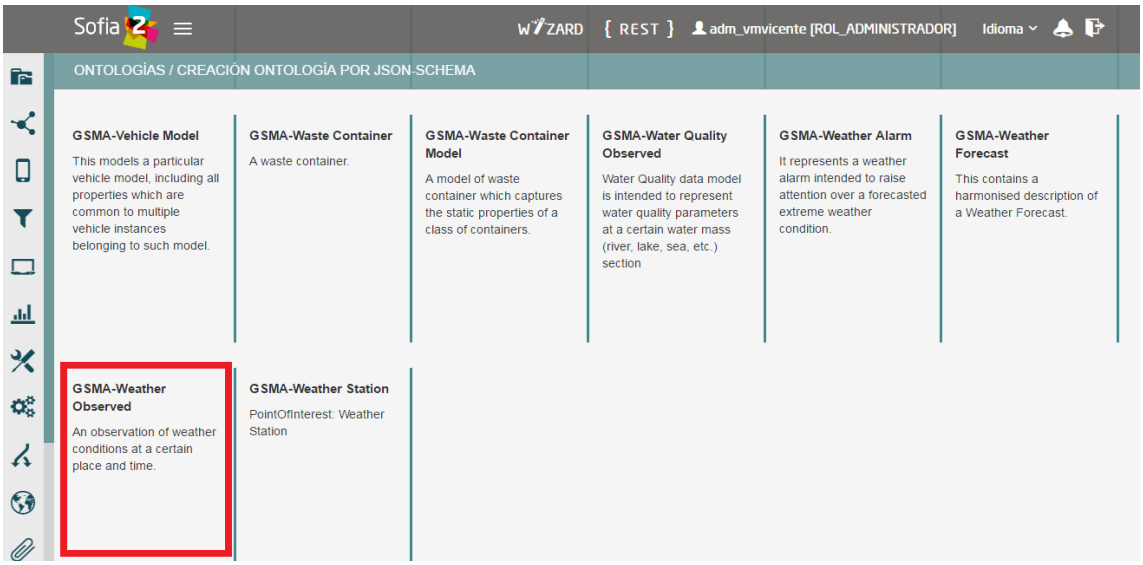
En el menú **Ontologías>Crear Ontología>crear Ontología mediante JSON**

Selecciono un nombre como **GSMA_WeatherObserved_Ontology**

Voy a la sección de Esquema, selecciono la categoría GSMA:



y selecciono como Plantilla:



Finalmente selecciono **Crear**

Con esto ya tengo creada mi Ontología conforme el Modelo Data Model GSMA.

3. Lo siguiente que haré será cargar unos datos para poder hacer consultas.

Puedo hacer esto desde el simulador de datos para simular instancias de la ontología, pero también puedo acceder a la gestión CRUD para crear datos desde un formulario. Desde **Ontologías>Gestión CRUD de Instancias:**

Selecciono

en la tabla:

Si me fijo en el ejemplo que ponía antes: WeatherObserved

{

“id”: “Spain-WeatherObserved-2422-2016-11-30T08:00:00”,

“type”: “WeatherObserved”,

“address”: { “addressLocality”: “Valladolid”, “addressCountry”: “ES” },

“atmosfericPressure”: 938.9,

“dataProvider”: “TEF”,

“dateObserved”: “2016-11-30T07:00:00.00Z”,

“location”: { “type”: “Point”, “coordinates”: [-4.754444444, 41.640833333] },

“precipitation”: 0,

“pressureTendency”: 0.5,

“relativeHumidity”: 1,

“source”: “http://www.aemet.es”,

“stationCode”: “2422”,

“stationName”: “Valladolid”,

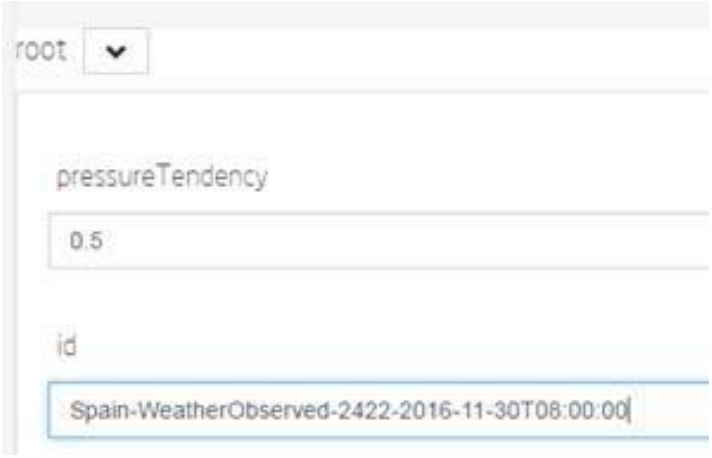
“temperature”: 3.3,

“windDirection”: -45,

“windSpeed”: 2

}

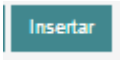
Y lo voy copiando en el formulario:



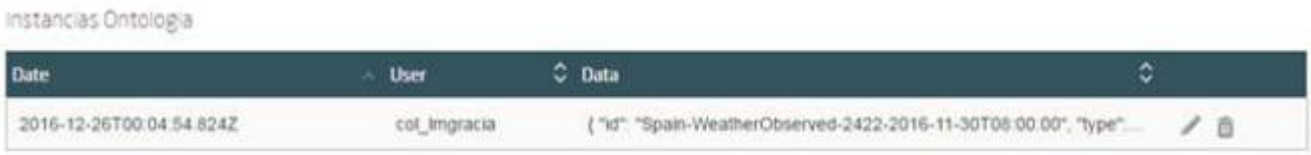
Sigo:

...

Al final del formulario puedo ver lo que se va a guardar:

Y finalmente seleccionaré 

Si voy al comienzo de la pantalla veré que ya se ha insertado:



5.2. Consulta de los datos de la Ontología creada (Modelo GSMA/FIWARE)

Una vez cargado este dato ya podré consultarlo desde los mecanismos de Sofia2:

A través de los conectores, publicarlo como API, en el Visor Open Data… Veamos:

Voy a **Herramientas>Consola BDTR y BDH.**

Selecciono mi Ontología y Generar Query (o la pongo: **select * from GSMA_WeatherObserved_Ontology limit 3**)

Que me devuelve los datos de la instancia insertada (además de los datos de Contexto como usuario, KP, momento de inserción), en este caso podemos ver los datos introducidos:

```
[
  {
    "_id": {
      "$oid": "58605ea8e4b0706036d2d711"
    },
    "contextData": {
      "session_key": "d2dc246f-9e24-4860-ac2c-9ff77fd0a083",
      "user": "col_lmgracia",
      "kp": "KP_SIB_API",
      "kp_instancia": "341e1716-245d-4fb0-9565-a9336cec5eb9",
      "timestamp": {
        "$date": "2016-12-26T00:04:54.824Z"
      }
    },
    "id": "Spain-WeatherObserved-2422-2016-11-30T08:00:00",
    "type": "WeatherObserved",
    "address": {
      "addressLocality": "Valladolid",
      "addressCountry": "ES"
    },
    "atmosfericPressure": 938.9,
    "dataProvider": "TEF",
    "dateObserved": "2016-11-30T07:00:00.00Z",
    "location": {
      "type": "Point",
      "coordinates": [
        -4.754444444,
        41.640833333
      ]
    },
    "precipitation": 0,
    "pressureTendency": 0.5,
    "relativeHumidity": 1,
    "source": "http://www.aemet.es",
    "stationCode": "2422",
    "stationName": "Valladolid",
    "temperature": 3.3,
    "windDirection": -45,
    "windSpeed": 2
  }
]
```

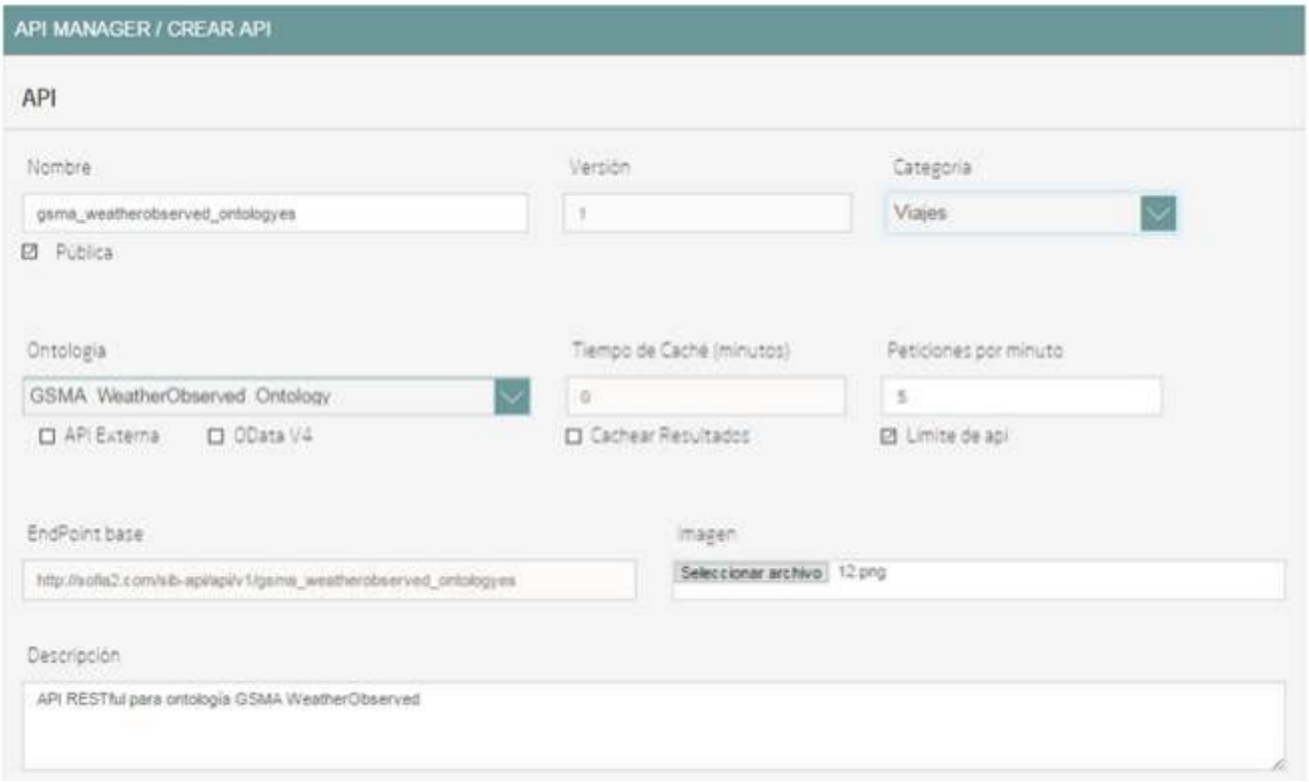

Esta información ya es accesible a través del SIB de Sofia2, a través de cualquiera de las APIs que ofrece.

5.3. Publicar Ontología como API RESTFUL

También puedo publicar esta Ontología como un API RESTFul para acceder a ella en una url de tipo:

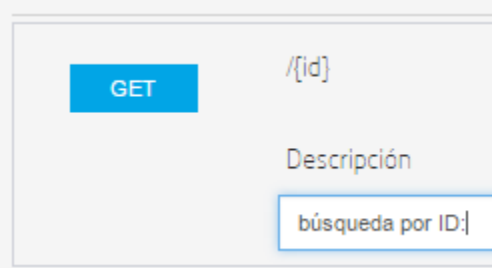
http://sofia2.com/sib-api/api/v1/gsma_weatherobserved_ontologyes

Para eso iré a **API Manager>APIs> Crear API:**



Habilitaré los métodos:

GET:

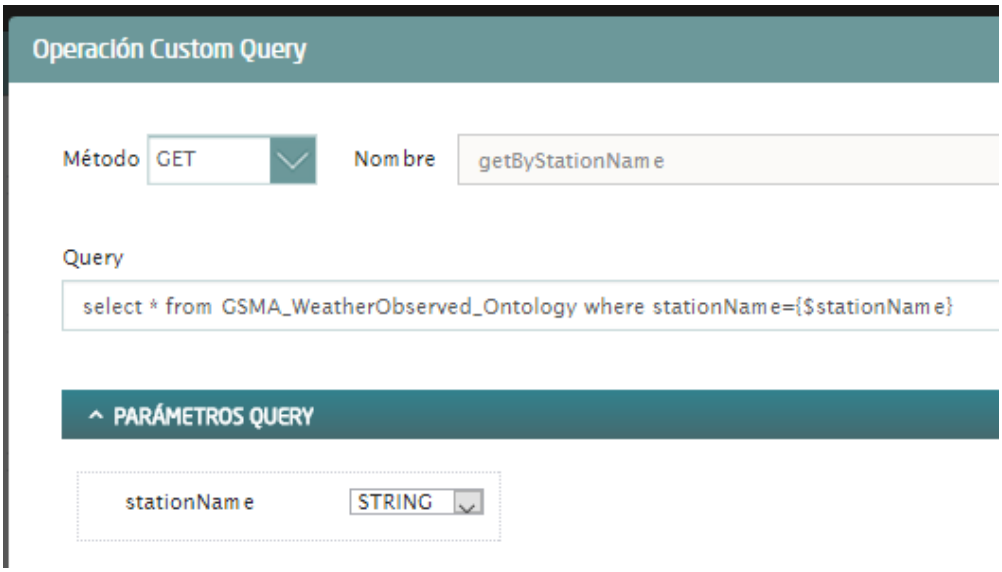


POST para INSERT, PUT para UPDATE y CUSTOM QUERY

En la CUSTOM QUERY quiero poder sacar los datos para una estación, por tanto la consulta es como esta:

select * from GSMA_WeatherObserved_Ontology where stationName='Valladolid'

En el UI debe registrarse así:



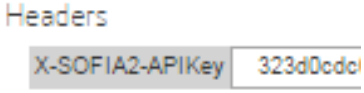
Es decir, el parámetro debe ir entre { } y con un $delante:

select * from GSMA_WeatherObserved_Ontology where stationName={$stationName}

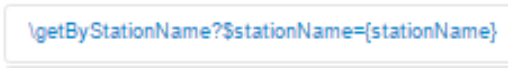
Tras esto, puedo probar mi API desde la opción **Mis suscripciones>Test&Doc**

(antes debo extraer la API Key desde **Mi API Key**)

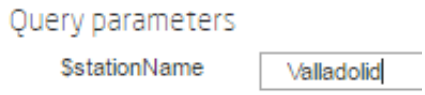
En la ventana de invocación en la parte de Headers pondré mi API Key:



Luego seleccionaré



Y en Query Parameters pondré Valladolid



Al invocarlo veo esto:

Esta misma invocación se puede realizar vía curl con una invocación de este estilo (escapamos el $por %24):

curl -v –H "-X-SOFIA2-APIKey:<my_token>" "http://sofia2.com/sib-api/api/v1/gsma_weatherobserved_ontologyes/getByStationName?%24stationName=Valladolid"

APIS y librerías Sofia2

Descargar PDF

Desarrollo de un cliente

Descargar PDF


Sofia 2

CHAPTER 25

Documentación para Desarrolladores Avanzados.

Creación Reglas Script

Descargar PDF

APIs Reglas Script

Descargar PDF

Motor CEP RI Sofia2

Descargar PDF

Cliente Sofia2 Arq. Kp-Modelo

Descargar PDF

KP gestionado (KP/APP Modelo)

Descargar PDF

Guía CEP Paso a Paso

Descargar PDF

CHAPTER 26

Demostradores

El objetivo de los demostradores es mostrar a través de ejemplos lo sencillo que es trabajar con Sofia2. Todos los ejemplos están accesibles junto a su código fuente.

Visor Geográfico

Representación de diferente información en diversos mapas, desde situación de autobuses en diversas ciudades en tiempo real, a información sensórica o representación de tuits sobre un tema concreto.

Ver sitio

Demo Twitter Streaming

Muestra las capacidades de la Plataforma para recibir en tiempo real información de Twitter a la vez que su representación una vez almacenado en la plataforma.

Ver sitio

Ver vídeo Demo

Demo API Streaming Twitter

Utilizando los Gadgets de la Plataforma se ha realizado un nuevo demostrador de las capacidades de integración de Sofia2 con RRSS, en este caso Twitter. El demostrador permite recibir en tiempo real información de Twitter y visualizar esta información de diversas formas: nube de palabras y diferentes gráficas.

 Ver sitio

Dashboard Smart Health

Cuadro de mando en el que asociado a cada paciente tenemos un wearable tipo pulsera capaz de medir información sobre pasos andados, sueño, oximetría,... Información que se cruza con histórica del paciente.

 Ver sitio

 Ver vídeo Demo

Dashboard Smart Retail

Ejemplo de panel de mando de todas las tiendas físicas de una empresa dedicada a la venta de ropa y en la que para cada tienda se pueden saber visitas, zonas más visitadas, ropa más probada y comprada,. . . y todo en tiempo real.

 Ver sitio

 Ver vídeo Demo

Gasolineras de España

Ejemplo en el que se representa en un mapa información sobre las gasolineras de España y los precios de combustibles actualizados diariamente, permite buscar por diversos criterios e identifica las gasolineras más caras y más baratas.

 Ver sitio

 Versión Developers

Visor Opendata

Permite tener acceso a la información Opendata de la plataforma.

 Ver sitio

Smart Home

Demostrador Smart Home en el que a través de un interfaz Web se controlan diversos dispositivos domóticos conectados con Sofia2.

 Ver sitio

iViewer Capas Sofia2

Ejemplo que integra en el Visor GIS iViewer de Indra la capacidad de cargar capas construidas en base a ontologías Sofia2.

 Ver sitio

Dashboard Estación Metereológica

Ejemplo de Dashboard creado vía Gadgets de la Plataforma. Requiere un usuario en Sofia2 CloudLab para poder visualizarlo.

 Ver sitio

Demostrador de Control de Paso con Beacons

A través de 2 beacons previamente situados y seleccionados en el cliente Android, se pueden ver las entradas y salidas de personas en un determinado lugar en el DashBoard Web, todo ello interconectado gracias a Sofia2.

 Ver sitio

Smart Agriculture

El siguiente dashboard utiliza distintos sensores para tomar medidas en tiempo real en una explotación agrícola. En este caso tomamos medidas de la presión atmosférica, temperatura , la evolución de la humedad del terreno y su temperatura...

 Ver sitio

Smart Distribution

SmartDistribution es un proyecto en cesium.js que muestra las paginas html desde los enlaces del GIS de cesium.js

 Ver sitio

 Ver vídeo Demo

Smart Drive

Demostrador Pay As You Drive(PAYD) Gracias al equipo de Vodafone y Oysta tenemos operativa una primera versión de este demostrador que incluye un enfoque social y de gamificación con el objetivo de reducir el número de siniestros y el precio del seguro

 Ver sitio

Telepizza

Demo para realizar el seguimiento de Pedidos.

Ver sitio

CHAPTER 27

Vídeos Tutoriales

- Registro en Sofia2 y Paso a Rol Colaborador
- Primeros pasos en Sofia2
- Tutorial de Modelado visual de Ontologías
- Tutorial de Creación de Ontologías
- Creación de una Ontología
- Tutorial de Creación KP y generación de token de autenticación
- CRUD: Create, Read, Update y Delete operaciones en Ontologías
- Tutorial de Uso del CRUD para la inserción de información
- Carga de datos desde Excel y generación de una ontología Sofia2
- Inserción de información en lenguaje Nativo desde la Consola de Sofia2
- Tutorial Sofia2 Dataflow
- Tutorial Subir Ficheros a Hadoop
- Tutorial Crawling Web
- Tutorial de Notebook en lenguaje Hive
- Tutorial de Notebook en lenguaje Spark
- Tutorial consultas SQL Excel con conexión a DataLink
- Tutorial Monitorización Sofia2
- Reglas Sofia2: Deteccion Ausencia de Eventos
- Tutorial Ejecución Script temporizados
- Scripting Rules Debugging
- Tutorial Contenedor de KPs
- Tutorial Generación de Informes

- Automatic HTML5 Client Generation Sofia2
- Twitter Real time Monitoring
- Gestión de Conexiones
- Búsqueda en Facebook, carga en Ontología y consulta de información
- Búsqueda en Twitter, carga en Ontología y consulta de información
- Tutorial Creación y lanzamiento de datos sobre un Widget
- Tutorial Creacion de Gadgets
- Tutorial Creacion Gadget Tabla e Interaccion otros Gadgets
- Tutorial Creacion de un Dashboard
- Nuevos Dashboards

CHAPTER 28

Blog

Puedes encontrar más noticias, información, ayuda y tutoriales sobre la Plataforma Sofia2 en nuestro Blog

Hazte seguidor y conseguirás estar al día de todo lo que acontece.

CHAPTER 29

Twitter

Mantente informado de las últimas noticias, novedades y eventos de la plataforma Sofia2 haciendote seguidor de nuestro canal en Twitter: @SOFIA2_Platform

CHAPTER 30

Comunidad

Visita la Comunidad de Sofia2

Registrate y colabora con nosotros.

CHAPTER 31

Descargas

La Plataforma disponibiliza las herramientas necesarias para poder desarrollar servicios, aplicaciones, extensiones o personalizaciones sobre la Plataforma.

Para ello la plataforma provee un set de herramientas para desarrolladores que facilitará y promoverá el trabajo con la plataforma.

La plataforma Sofia2 ofrece diferentes APIs y un SDK que permite su integración con cualquier tipo de dispositivo.

Descargue de nuestro repositorio aquellas librerías y recursos que necesite para el uso de nuestros productos:

API Java

APIs Java para el desarrollo de plugins de extensión para Sofia2-Core para poder ampliar las capacidades de la plataforma, pudiendo también realizar las adaptaciones necesarias para su interactuación con otros sistemas.

 Descargar

API Javascript

 Descargar

API Android

 Descargar

API IOS

 Descargar

API Phython

 Descargar

API Node.js

 Descargar

API C

 Descargar

API Arduino

 Descargar

API .NET

 Descargar

KP Modelo

 Descargar

Además facilita al desarrollador un IDE personalizado basado en Eclipse para el desarrollo simplificado de integraciones sobre la Plataforma:

SOFIA2 SDK (Windows)

Descargar SOFIA2 SDK (Windows)

SOFIA2 SDK (Mac)

Descargar SOFIA2 SDK (Mac)

SOFIA2 SDK (Linux)

Descargar SOFIA2 SDK (Linux)