UNIVERSIDAD POLITÉCNICA DE MADRID

GRADO EN INGENIERIA DE COMPUTADORES

DISEÑO E IMPLEMENTACIÓN DE UNA PLATAFORMA ESCALABLE PARA LA GESTION DE SISTEMAS IOT

AUTOR: **Francisco Javier García Álvarez**

TUTOR: **Bernardo Tabuenca Archilla**

Resumen

En este proyecto se diseña e implementa un sistema complejo para la trazabilidad y monitorización de sistemas IoT en el contexto de la ETSI Sistemas Informáticos de la Universidad Politécnica de Madrid. Para ello, se han desarrollado diferentes elementos, los cuales se integran entre ellos generando un sistema final que cumple el objetivo de este proyecto, mantener en un estado óptimo la planta.

Para evaluar el sistema, se ha realizado un macetero formado por una placa ESP32 y diferentes tipos de sensores. Dicha placa será la encargada de recoger los valores medidos por los sensores, como puede ser la temperatura ambiente, la luminosidad, la humedad de la tierra, etcétera. Además de recoger dichos valores, los envía a la plataforma “Internet of Things “(IoT), ThingsBoard, la cual desempeña un papel imprescindible.

Una vez llegan los datos a dicha plataforma, ésta se encarga de realizar la lógica necesaria para atender las necesidades básicas de la planta. La plataforma IoT recibirá cada X minutos los datos recogidos por los sensores, los cuales pasarán por una serie de reglas que los analizarán y modificarán los valores de los actuadores en consecuencia. Los valores de estos actuadores serán leídos por la placa ESP32 y ésta misma modificará los dispositivos correspondientes. Dichas reglas también se encargan de realizar llamadas al servicio REST para persistir los valores y las acciones en la base de datos.

El último elemento del sistema es una aplicación Web, que integra una interfaz gráfica con un servicio REST (back-end). El objetivo de esta aplicación, es permitir al usuario llevar un control más detallado sobre la planta y poder conocer el estado de ésta de una forma más visual. El servicio REST integra una base de datos relacional postgres, la cual permite persistir los valores de los sensores, los clientes, las acciones realizadas sobre la maceta, etc.

Palabras clave: IoT, API REST, Interfaz, Servicio.

Índice

1. Introducción

1.1. Contexto

Para entender el marco en el que se desarrolla este proyecto es necesario saber que es el Internet de las cosas o IoT. El internet de las cosas o '*Internet of Things*', es un concepto, una enorme red de dispositivos electrónicos capaces de enviar y recibir información entre ellos. De esta forma se abre un amplio abanico de oportunidades para una mejor integración del mundo físico en los sistemas. Según el estudio '*Internet of Thing for Smart Cities*' el IoT podrá incorporar de manera transparente y sin problemas un gran número de sistemas finales diferentes y heterogéneos, proporcionando al mismo tiempo acceso abierto a determinados subconjuntos de datos para el desarrollo de una plétora de servicios digitales.(Zanella et al., 2014)

En 2019 la empresa de telefonía llamada 'Telefónica', presentó un estudio sobre el internet de las cosas, '*Things Matter 2019: la experiencia del usuario de IoT en España', éste* estudio constata que se ha producido un aumento del uso de esta tecnología en un 66% en dos años. Además '*A medida que IoT se conoce más, crece la confianza de los usuarios (más de un 50%) en términos de seguridad. La eficiencia energética y la sostenibilidad de los dispositivos y soluciones IoT aparecen como una prioridad tanto a nivel industrial como personal*.'(Telefónica, 2019)

En un reciente artículo, se dice que la disminución del coste de los sensores durante los últimos años, y la llegada de la 5ª generación de tecnología móvil beneficiará enormemente a la innovación del Internet de las Cosas (IoT). Por consiguiente, el uso de IoT en nuevas prácticas agronómicas podría ser una parte vital para mejorar la calidad del suelo, optimizar el uso del agua o mejorar el medio ambiente. No obstante, todavía no se ha explorado la aplicación de sistemas IoT para fomentar la conciencia ambiental en los entornos educativos.(Tabuenca et al., 2020)

Es una realidad que el mundo IoT está adquiriendo un papel fundamental en la creación de nuevos dispositivos electrónicos de todo tipo. Más concretamente, y con el objetivo de mejorar los conocimientos sobre este sector en la asignatura '*Sistemas Basados en Computadores*', correspondiente al título de Ingeniería de Computadores de la Universidad Politécnica de Madrid, se propuso a los alumnos la realización de un

dispositivo electrónico conectado con una plataforma IoT, capaz de intercambiar información con dicha plataforma.

¿Pero que es una plataforma IoT? Según una publicación del Instituto Español de Estudios Estratégicos (IEEE), una plataforma IoT es el elemento que proporciona la capacidad de gestionar de forma centralizada los dispositivos IoT necesarios, mediante el suministro de herramientas y servicios destinados a este propósito(Asemani et al., 2019). Una plataforma se puede considerar como el middleware de un sistema IoT, la cual se va a encargar de comunicarse tanto con el servicio web como con los dispositivos IoT.

El sistema basado en computador propuesto por la asignatura fue un macetero 'inteligente' formado por un micro-controlador y una serie de sensores encargados de monitorizar las variables de planta.

Puesto que se iban a crear un número de sistemas considerable, surgió la idea de implementar un servicio que fuese capaz de englobar todos estos maceteros y poder llevar un control sobre ellos. La idea de este servicio evolucionó para crear la base de este proyecto, un sistema IoT que integrara una plataforma IoT y el servicio web para llevar el control de todos estos sistemas basados en computador.

1.2. Motivación

Este trabajo está motivado en la necesidad inicial de desarrollar un sistema exclusivamente para las 6 macetas que se realizarían en la universidad, sin embargo, la posibilidad de añadir cualquier tipo de dispositivo en este sistema era una idea muy atractiva.

La Escuela Técnica Superior de Ingeniería de Sistemas Informáticos de la Universidad Politécnica de Madrid empieza a trabajar en diferentes proyectos relacionados con el mundo IoT, como por ejemplo, la medición de las condiciones que se dan en las aulas del campus sur para el control del gasto energético. Estos proyectos, los cuales tienen también una relación con los Objetivos de Desarrollo Sostenible (ODS), presentaban la oportunidad de concienciarme y concienciar más sobre este tema a los alumnos de la universidad, lo que resulta una labor beneficiosa en muchos aspectos.

Estos diferentes proyectos que se estaban llevando desde la escuela daban la posibilidad de realizar un sistema escalable y que sirviese para el conjunto de la universidad completa lo que era un gran aliciente para implementarlo.

La realización de este proyecto daba la oportunidad de afrontar un gran reto y aprender en cuanto a arquitecturas de aplicaciones web, lo que personalmente me resulta muy interesante y me permite adquirir una visión global en este tipo de proyectos.

Por otro lado implementar este sistema daba la posibilidad de optar a la ayuda que otorga la Universidad Politécnica De Madrid para los TFG/TFM dentro de la convocatoria de campus sostenible, lo que supone otra fuerte motivación.

1.3. Trabajo relacionado

En la subsección de 'Contexto' se describe qué es una plataforma IoT, no obstante y puesto que existen un gran número de ellas, se va a realizar una comparación entre ocho de las más reconocidas,tomando como referencia seis de revistas relevantes en este campo; Software Testing Help[1], UbuntuPit[2], Un poco de Java[3], DZone[4]

Para realizar esta comparación se ha recopilado información de diferentes comparativas realizadas en otras páginas web y de la propia documentación de las plataformas seleccionadas. Se ha realizado una tabla comparativa que considera seis factores diferentes, reflejados en cada una de las columnas de la tabla 1.

- **Integración** Esta columna indica las formas con las que se puede integrar un servicio con esa plataforma. Hoy en día es muy común

[1] https://www.softwaretestinghelp.com/best-iot-platforms/

[2] https://www.ubuntupit.com/choose-the-right-iot-platform-top-20-iot-cloud-platforms-reviewed/

[3] https://unpocodejava.com/2016/11/02/tabla-comparativa-plataformas-iot-azure-iot-hub-vs-aws-iot-vs-watson-iot-foundation-vs-sofia2-iot-platform/

[4] https://dzone.com/articles/10-cloud-platforms-for-internet-of-things-iot

para implementar los servicios WEB, como estas plataformas, utilizar servicios API REST gracias a la facilidad que aportan a la hora de la integración con ellos y la escalabilidad que ofrece.

- **Seguridad** Especifica los protocolos de seguridad soportados por esa plataforma. Por lo que se puede observar la mayoría de estas plataformas deciden utilizar el 'Transport Layer Security' (TLS), para la comunicación entre el usuario y la plataforma." *Se trata de protocolos criptográficos que proporcionan privacidad e integridad en la comunicación entre dos puntos en una red de comunicación. Esto garantiza que la información transmitida por dicha red no pueda ser interceptada ni modificada por elementos no autorizados, garantizando de esta forma que sólo los emisores y los receptores legítimos sean los que tengan acceso a la comunicación de manera íntegra."*(*Qué Es El Protocolo SSL/TLS* | *Redalia*, n.d.)

- **Protocolos** Referencia a los protocolos de comunicación de los dispositivos IoT con la plataforma. Cabe destacar que la mayoría de plataformas soportan el protocolo MQTT ya que éste fue diseñado especialmente para la comunicación en el Internet de las Cosas y es un protocolo muy ligero basado en publicación/suscripción. No obstante, también suelen soportar un protocolo más común de comunicación, HTTP. "*Los dispositivos IoT se comunican mediante protocolos de IoT. El protocolo de Internet (IP) es un conjunto de reglas que determina cómo se envían los datos a Internet. Los protocolos de IoT garantizan que un dispositivo o sensor lea y comprenda la información enviada por otro dispositivo o sensor. Dada la gran diversidad de dispositivos IoT disponibles, es importante usar el protocolo adecuado en el contexto adecuado*."(*Protocolos y Tecnologías de IoT* | *Microsoft Azure*, n.d.)

- **Base de Datos** Refleja si la plataforma IoT tiene una base datos donde persistir los datos. En el caso de grande empresas como Google o Amazon cuentan con su propio motor de base de datos lo que le proporciona un valor añadido.

- **Precio** Otro factor que se valora en esta comparación es el precio, es decir, el coste que supone el uso de esta plataforma. Actualmente es tendencia el ‘pago por uso’, es decir, según el uso que le des a la plataforma y lo que necesites, el precio varía.

- **Interfaz** Por último, la interfaz. Esta columna señala si la plataforma cuenta con una interfaz gráfica para el usuario o por lo contrario debe utilizarse mediante comandos. ”*La interfaz gráfica de usuario, conocida también como GUI (del inglés graphical user interface) es un programa informático que actúa de interfaz de usuario, utilizando un conjunto de imágenes y objetos gráficos para representar la información y acciones disponibles en la interfaz. Su principal uso, consiste en proporcionar un entorno visual sencillo para permitir la comunicación con el sistema operativo de una máquina o computador.*”(Luis et al., n.d.)

	Integración	Seguridad	Protocolos	Base de Datos	Precio	Interfaz
AWS IoT Core	API REST	Autenticación protocolo TLS	MQTT, HTTP	AWS S3	Por uso	SI
IBM Watson Internet of Things	Real-time API's	Autenticación protocolo TLS *'Single Sign On'* LDAP	MQTT, HTTP	----------	Por uso	SI
Google Cloud IoT Core	API REST	Protocolo TLS 1.2, Certificados CA firmados	HTTP y MQTT	Cloud Pub/Sub	Por uso	SI
ThingWork IoT Platform	API REST	ISO 27001, LDAP	MQTT, AMQP, XMPP, CoAP, DDS, WebSockets	----------	Contactar	SI
Microsoft Azure IoT Suite	API REST	Autenticación protocolo TLS	HTTP, AMPQ, MQTT	Azure Cosmos DB	Por uso	SI
Oracle Internet of Things	API REST	Autenticación protocolo TLS	MQTT, CoAP, AMQP	Oracle Databse	Por uso	SI
ThingsBoard	API REST	Autenticación protocolo TLS , Access Token	HTTP, MQTT	Postgres	Gratis	SI
Cisco Kinect	API REST	----------	MQTT, CoAP, AMQP	----------	Contactar	SI
Sofia2 IoT Platform	API REST	Autenticación protocolo TLS	HTTP, MQTT, AMQP, JMS, OPC	----------	Por uso	SI

Tabla 1: Comparación plataformas IoT. (Elaboración propia)

1.4. Definición del problema a resolver

Los Objetivos de Desarrollo Sostenible (ODS) pretenden combatir la pobreza, proteger el planeta y asegurar la prosperidad para todos como parte de una nueva agenda de desarrollo sostenible (*Objetivos y Metas de Desarrollo Sostenible – Desarrollo Sostenible*, n.d.). Estos objetivos que adoptaron la mayoría de países en el año 2015 es uno de los puntos importantes en la lucha contra el cambio climático en la actualidad, por ello desde la asignatura de Sistemas Basados en Computadores querían concienciar a los alumnos sobre estos objetivos y más concretamente sobre el objetivo 'Producción y Consumo responsable'.

Para realizar esta concienciación se pidió la elaboración de unos dispositivos IoT, que tratarán de ayudar en mayor o menor medida en este objetivo de desarrollo sostenible. En consecuencia se elaborarían una serie de dispositivos IoT independientes, sin embargo, no había nada para realizar un almacenamiento de los datos recogidos para su posterior procesamiento, ni había forma de controlar esta serie de dispositivos.

En primer lugar está la necesidad de ser propietarios de todos los datos recogidos por estos dispositivos IoT o por cualquier otro tipo de dispositivos que la escuela genere. Esto es muy importante puesto que hoy en día el procesamiento masivo de datos es uno de los factores más relevantes en el mundo IoT y para ello es necesario tener los datos almacenados en nuestra propiedad y con el modelo de datos que nos facilite dicho procesamiento masivo. Este almacenamiento permite estudiar el comportamiento de los diferentes dispositivos que se añadan a este sistema además de dar la posibilidad de implementar algoritmos de machine learning para la predicción de estos comportamientos. Por lo que era necesario cubrir esta necesidad de almacenamiento de información.

En segundo lugar, es necesario cubrir la posibilidad de que el número de estos dispositivos crezca en gran medida por lo que es preciso realizar un sistema escalable y que se pueda integrar fácilmente con otro sistema que pudiera tener la universidad anteriormente o que se genere posteriormente.

En tercer lugar, la posibilidad de ver el estado de los dispositivos IoT de una forma visual, cierto es que la plataforma IoT permite definir '*dashboard'* pero realmente no es una forma muy visual de ver la información. Se necesitaba una interfaz que permitiría ver el estado de todos los dispositivos a la vez y de una forma más interpretable.

2. Herramientas implementadas

A lo largo de esta sección se van a desarrollar cada uno de los elementos que componen la solución final planteada por este proyecto. En esta introducción se pretender ayudar a entender mejor la arquitectura global del proyecto para que en las siguientes subsecciones se pueda comprender el funcionamiento de cada elemento.

En la ilustración 1 se representan cada uno de los componentes y como se relacionan entre ellos. Como se ve en la imagen, la mayoría del sistema está montado sobre un servidor en el Centro de Supercomputación y Visualización de Madrid (CESVIMA)[5]el cual contiene otro servidor Tomcat donde se alojan las dos interfaces gráficas. Fuera de este servidor Tomcat se encuentran ThingsBoard y el servicio API REST. Estas dos interfaces y estos dos servicios están expuestos a internet y se puede acceder a ellos mediante la URL que aparece en la ilustración 1. Además en dicha ilustración se puede ver que la solución cuenta con una base de datos Postgres para el almacenamiento masivo de los datos recabados por el dispositivo IoT.

Al margen de estos elementos tenemos el dispositivo IoT que servirá para comprobar el funcionamiento de la solución, en este caso más particular se ha seleccionado una maceta representada en la parte derecha de la imagen.

[5] https://www.cesvima.upm.es/

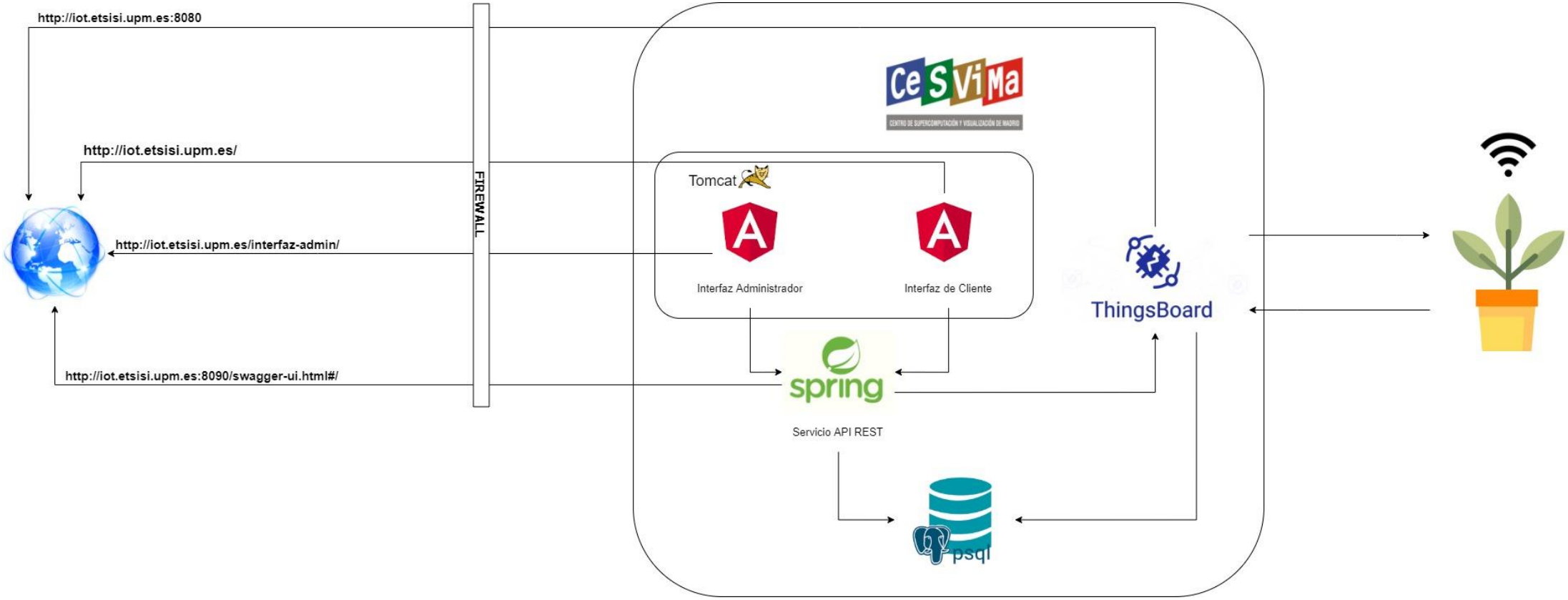


Ilustración 1: Arquitectura Global. (Elaboración propia)

2.1. Arquitectura

Como se ilustra en la figura anterior el sistema final cuenta con una serie de elementos que se integran entre sí. Para alojar todo este sistema se ha decido utilizar una máquina/servidor, en un servicio en nube, lo que además nos proporciona una ip estática para poder acceder a cada uno de los elementos del sistema desde el exterior. Para ello la Universidad Politécnica de Madrid ha proporcionado un servidor en el Centro de Supercomputación de Madrid (CESVIMA). Esta máquina cuenta con los recursos necesarios, tanto de cómputo como de memoria, para soportar todos los elementos del sistema, añadiendo además los protocolos de seguridad necesarios. Otra característica de este servidor es que viene con un SO Linux preinstalado.

En cuanto a la seguridad la máquina cuenta con un '*cortafuegos'* para el control de las conexiones entrantes o salientes. Un '*cortafuegos*' es un elemento que viene integrado en la mayoría de los propios equipos de tal forma que permiten el tráfico de internet que sea seguro y bloquean las conexiones maliciosas. La mayoría de los cortafuegos actuales permiten establecer al usuario cuales son las conexiones seguras y cuáles no, mediante la definición de reglas. "*Las reglas de cortafuegos definen qué tipo de tráfico de Internet se permite o bloquea*" (*¿Qué Son Las Reglas de Cortafuegos? | Client Security for Windows | 13.10 | F-Secure User Guides,* n.d.)Estas reglas permiten a los cortafuegos controlar tantos las conexiones entrantes como salientes de la máquina.

Puesto que se trata de un sistema operativo Linux la definición de estas reglas se realiza mediante comando. Para este proyecto se han habilitado las conexiones por los puertos 8090 y 80, además del puerto 8080 que ya viene abierto por defecto.

Además de añadir esta configuración se ha instalado un servidor Tomcat para que actúe a modo de '*proxy invertido'*, es decir, redirija las conexiones entrantes al sub-sistema correspondiente.

Para entender un poco mejor como es la infraestructura de la solución final se detalla cómo se ha desplegado cada elemento de la misma.

En la página de aterrizaje del sistema, *iot.etsisi.upm.es*[6], se encuentra la página web donde se verá reflejado el estado de la planta (Interfaz de Cliente), de esta

[6] http://iot.etsisi.upm.es/

forma el usuario podrá controlar la planta de una forma rápida y sencilla. Esta interfaz se encuentra desplegada sobre el servidor Tomcat en el puerto 80, que viene abierto por defecto y es donde te redirige Tomcat cuando no se especifica ningún puerto concreto. Este elemento de la solución se detalla en la subsección 2.1.1 'Interfaz del Cliente'.

Otro sub-sistema que se encuentra desplegado sobre este Tomcat es la interfaz de administrador, '*iot.etsisi.upm.es/interfaz-administrador*'[7]. Esta interfaz permite dar de alta un nuevo usuario en la plataforma o bien permite modificar lo necesario una vez se ha creado. El detalle de este sub-sistema se encuentra en la sección 2.1.2

Por otro lado tenemos el servicio de la plataforma IoT y el servicio web desarrollado para este proyecto. Estos elementos no se encuentran desplegados sobre el servidor Tomcat, ya que la tecnología usada para implementarlos, la cual detallaremos más adelante, levanta un servidor embebido. El servicio de la plataforma IoT proporciona una interfaz gráfica, a la cual se puede acceder por el puerto 8080 del dominio de la máquina. En cuanto al servicio desarrollado, se encuentra desplegado en el puerto 8090 y este está integrado con las dos interfaces desplegadas sobre el servidor Tomcat. Estos dos elementos se describen detalladamente en las secciones 2.1.3 'Servicio API REST' y 2.1.4 'Plataforma IoT'

Ajeno a toda esta configuración en la máquina del CESVIMA, se encuentra el sistema hardware. Este sistema es completamente independiente puesto que no necesita estar ejecutándose en un servidor, sino que está corriendo sobre el propio micro controlador. El micro controlador cuenta con una tarjeta de red para poder conectarse a la red y poder comunicarse con el servidor y enviar los datos recuperados por los sensores.

2.1.1. Interfaz del Cliente

La 'Interfaz del Cliente' permite al usuario poder ver la información de su dispositivo de una forma más visual que la plataforma IoT. Se considera cliente cualquier dispositivo electrónico compuesto por sensores y actuadores, que comunican información a la plataforma IoT. Un ejemplo de cliente es el dispositivo IoT

[7] http://iot.etsisi.upm.es/interfaz-admin

desarrollado para la evaluación de este proyecto y que puede verse en la sección 3.1 'Evaluación del Sistema con un Cliente IoT'. Esta interfaz permite al usuario llevar el control de su dispositivo IoT de una forma más cómoda además de poder ver los valores recogido por los sensores.

Pero antes de entrar en el detalle de este caso concreto, y cómo se ha implementado, se va a explicar con qué tecnología se ha desarrollado. Angular es un framework de diseño de aplicaciones y una plataforma, de desarrollo para crear aplicaciones eficientes y sofisticadas de una sola página. (*Angular - Introduction to the Angular Docs*, n.d.). Este framework utiliza el lenguaje de programación TypeScript, el cual se considera como un superset de JavaScript. Un superset es un lenguaje escrito sobre otro lenguaje, es decir, TypeScript es un lenguaje escrito sobre JavaScript, lo que permite que cualquier navegador entienda el código desarrollado en TypeScript como si fuera original de JavaScript. Este lenguaje de programación permite implementar la lógica dentro de la propia interfaz gráfica.

Para la parte visual Angular utiliza el lenguaje HTML o '*Hyper Text Markup Language',* éste es un lenguaje de marcado estándar para la creación de páginas web. En HTML se describe la estructura de la página web con una serie de elementos y estos son los que le dicen al navegador como debe mostrar el contenido.

En Angular se manejan 3 conceptos fundamentales módulos, componentes y servicios junto con la inyección de dependencias.

- **Módulos:** Las aplicaciones de Angular son modulares y esta herramienta proporciona un sistema propio. NgModules son contenedores utilizados para conjuntos de código que trabajen dentro del mismo ámbito de la aplicación, un flujo de trabajo o conjuntos de funcionalidades ligadas entre sí. Este sistema de modularización puede contener componentes, servicios y otros archivos de código. Estos *NgModules* pueden importar las funcionalidades entre sí.

  Un módulo permite declarar el contexto para la compilación de un componente, cada aplicación angular mínimo ha de tener el modulo raíz, el cual proporciona el mecanismo para arrancar la aplicación. Una aplicación suele tener varios módulos, los cuales importan las funcionalidades unos de otros.

- **Componentes:** Los componentes se encargan de controlar una vista, definen la lógica que esa vista debe implementar. La clase interactúa con la vista a través de una API de propiedades y métodos. Como se aprecia en la ilustración 2, un componente define también los datos que serán pasados a la vista.

src / app / hero-list.component.ts (clase)

```
export class HeroListComponent implements OnInit {
  heroes: Hero[];
  selectedHero: Hero;

  constructor(private service: HeroService) { }

  ngOnInit() {
    this.heroes = this.service.getHeroes();
  }

  selectHero(hero: Hero) { this.selectedHero = hero; }
}
```

Ilustración 2: Definición de componente Angular

- **Servicios:** Los servicios son clases con un propósito específico, para cubrir una necesidad de la aplicación. Angular distingue entre servicios y componentes, separando la lógica relaciona con una vista de cualquier otro tipo de procesamiento, de esta forma aumenta la reutilización y la modularidad. Un componente debe contener propiedades y métodos para gestionar la lógica de la vista, sin embargo puede delegar ciertas tareas en servicios como puede ser la llamada a servidor, validar la entrada, etc. Estos servicios se definen como inyectables y permite que cualquier otro componente pueda inyectarlo y utilizar sus funcionalidades. En la ilustración 3 se puede ver la definición de un servicio en Angular.

src / app / hero.service.ts (clase)

```
export class HeroService {
  private heroes: Hero[] = [];

  constructor(
    private backend: BackendService,
    private logger: Logger) { }

  getHeroes() {
    this.backend.getAll(Hero).then( (heroes: Hero[]) => {
      this.logger.log(`Fetched ${heroes.length} heroes.`);
      this.heroes.push(...heroes); // fill cache
    });
    return this.heroes;
  }
}
```

Ilustración 3: Definición de Servicio

Para la implementación de esta interfaz gráfica se han definido 5 componentes uno por cada una de las vistas que tiene la 'Interfaz del Cliente'.

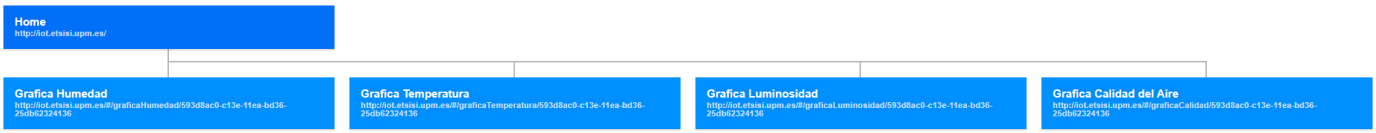


Ilustración 4: SiteMap Componentes (Elaboración propia)

En primer lugar, tenemos la página '*home*' la cual permite al usuario ver los diferentes parámetros de su dispositivo IoT, para concretar más el dispositivo IoT utilizado es una maceta, como se puede observar en la ilustración 5. En esta ilustración se ven algunos de los parámetros que se miden en la planta, la calidad del aire, la humedad, la luz y la temperatura, de esta forma cuando el usuario quiera controlar el estado lo puede hacer directamente desde la página principal.

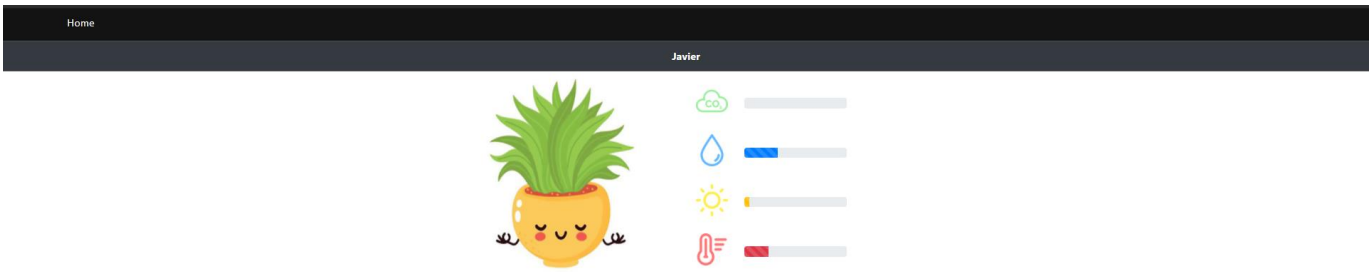


Ilustración 5: Interfaz de Control. (Elaboración propia[8])

Se han definido otros 4 componentes, uno para cada parámetro que se muestra en la página de '*home*'. Cuando el cliente quiere ver en detalle los valores tomados por cualquiera de estos cuatro sensores selecciona el parámetro concreto que desea ver y se abrirá otra vista donde se podrán ver los valores recogidos por el sensor en una gráfica y en una tabla, como se puede ver en la ilustración 6. De esta forma el cliente podrá también ver de forma detallada el estado de la planta.

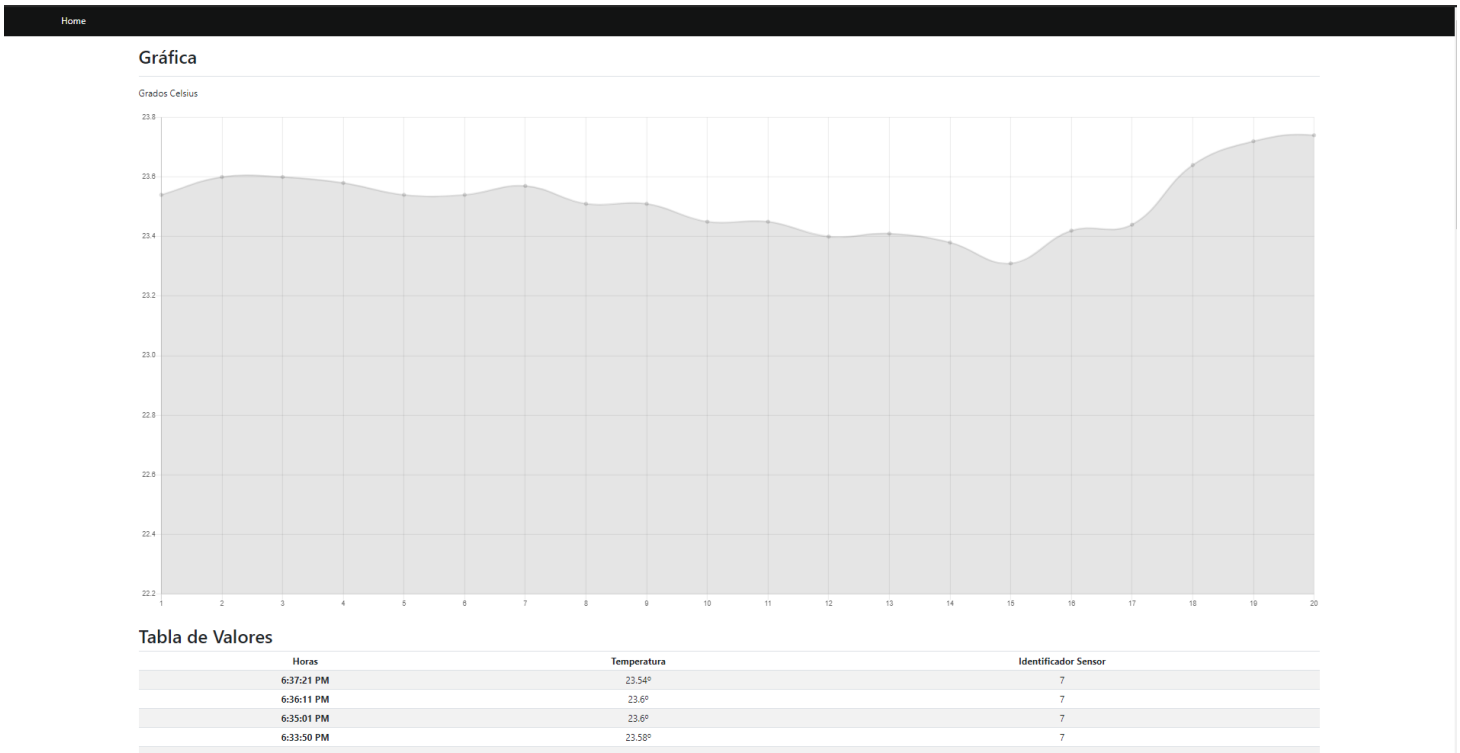


Ilustración 6: Grafica. (Elaboración propia)

[8] http://iot.etsisi.upm.es/#/home

2.1.2. Interfaz Administrador

Se considera administrador de la plataforma, aquella persona encargada de crear y modificar clientes en el sistema. Además este administrador tendrá acceso a la base de datos por si fuese necesario realizar alguna acción. Al igual que la 'Interfaz del Cliente' ésta se ha desarrollado con la tecnología Angular, no obstante, su función es completamente diferentes. Como ya se ha comentado en la sección de Introducción este proyecto busca ser escalable para poder alojar los dispositivos IoT que desarrolle en el futuro la Universidad Politécnica de Madrid. Para facilitar la labor del administrador a la hora de crear un nuevo usuario en el sistema se ha implementado una interfaz que automatice en la medida de lo posible esta función, la 'Interfaz del Administrador'.

Esta interfaz consta de un formulario que el administrador debe cumplimentar para dar de alta un nuevo cliente. Los campos de este formulario son:

- **ThingsBoardId:** Este campo representa el identificador que se le haya otorgado al cliente por la plataforma ThingsBoard, por tanto es imprescindible dar de alta un cliente nuevo en la plataforma IoT.

- **Descripción:** Texto para describir el cliente creado.

- **Sensores del Cliente:** En este aparatado del formulario se pueden añadir tantos sensores como se quieran, estos sensores se relacionarán directamente con el cliente. Los sensores deben tener algún tipo de los que se muestran en el catálogo, no obstante, ese catálogo se pude ampliar si fuese necesario. Una característica imprescindible para la creación de estos sensores, es que debe asignar el mismo nombre que el que tienen en la plataforma IoT, es decir, si los datos del sensor le llegan a la plataforma con el nombre 'XX', el nombre del sensor en el sistema ha de ser 'XX'.

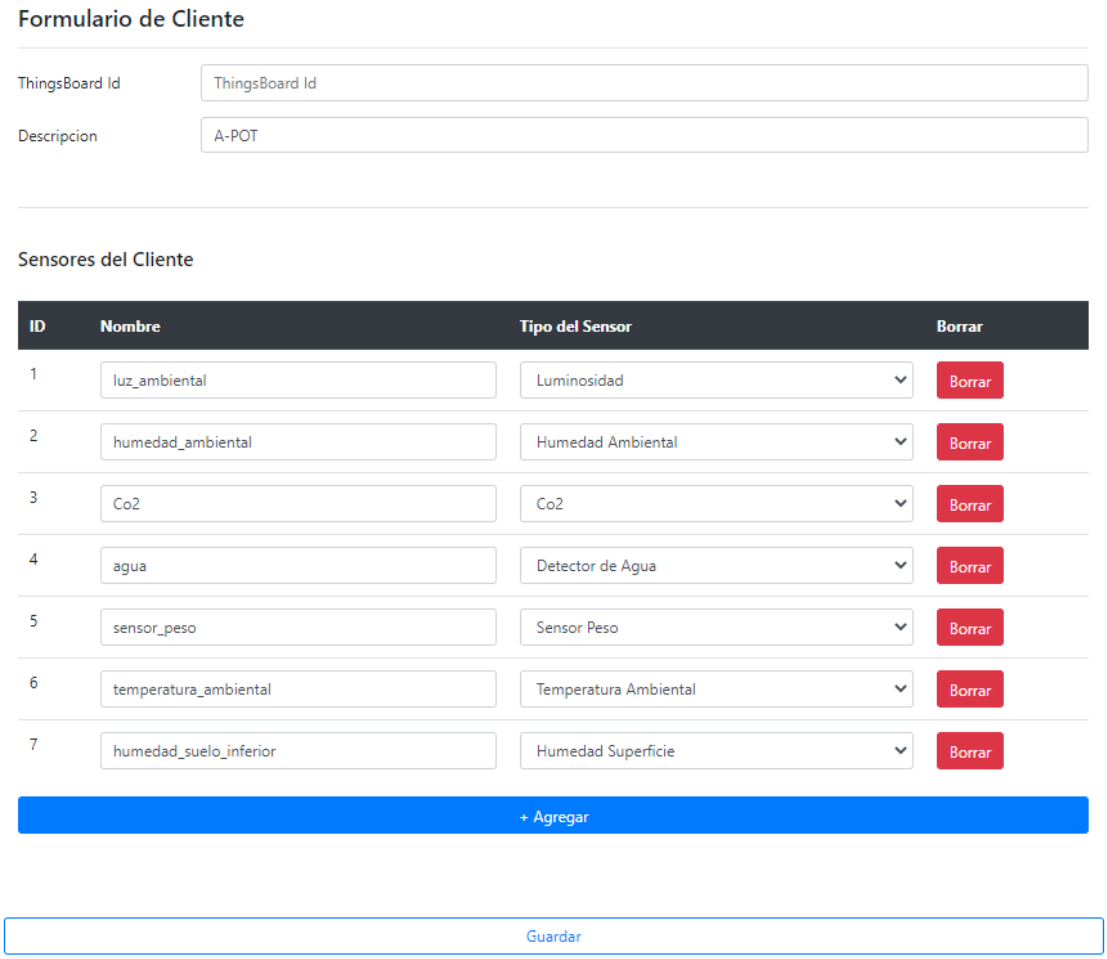


Ilustración 7: Interfaz Administrador. (Elaboración propia)

2.1.3. Servicio API REST

El elemento encargado de gestionar gran parte del sistema es este servicio API REST. Se encarga de realizar la comunicación con la base de datos para recuperar toda la información que necesitarán las interfaces, además de realizar toda la lógica para su funcionamiento. Por otro lado este servicio se encarga de persistir en base de datos todos los valores recogidos por los sensores del dispositivo, gracias a que expone una API para permitir al usuario u otro sistema utilizar funcionalidades del servicio. En la sección posterior de 'Plataforma IoT' se explica cómo la plataforma consume esta API para persistir los datos que le llegan de los sensores. Antes de explicar la API creada para este servicio se va a explicar que es un servicio REST.

Antes del año 2000 el concepto de REST no existía y se trabajaba con otro modelo de servicios que era el modelo SOAP. Este tipo de servicios eran bastante potentes pero sin embargo su complejidad iba paralela a su potencia.

REST o "REpresentational STate Transfer" permite desarrollar un servicio de forma más sencilla. La característica principal es que este estilo de arquitectura está orientado a recursos por lo que las peticiones sobre este este servicio actúan directamente sobre los propios recursos del sistema. Además este tipo de servicios permite la comunicación mediante las interfaces XML o JSON.

Un ejemplo para diferenciar mejor estos dos estilos de arquitectura de servicios. Cuando un cliente realiza una llamada, para obtener todos los usuarios listarUsuarios (), a un servicio SOAP, éste realizará otra petición al sub-servicio de usuarios para solicitarle todos los usuarios y poder mostrárselos al cliente. En cambios en los servicios REST el usuario realizará la petición listarUsuarios () y accederá directamente al recurso de usuarios para obtener todos los disponibles en base de datos.

Puesto que para este sistema se utiliza un servicio REST se va a explicar algunos principios de diseño (por Microsoft). REST es independiente de cualquier protocolo subyacente y no está necesariamente unido a HTTP, sin embargo, para este proyecto se ha utilizado el protocolo HTTP, para realizar las peticiones.

Como ya se ha mencionado un servicio API REST está diseñado entorno a recurso y cada uno de estos recursos tendrá un identificador único, para poder acceder a él mediante una petición directa, del tipo '*recurso/identificador*'. Para la comunicación con el servicio se utiliza el intercambio de interpretaciones de recursos,

para este proyecto se utiliza la interfaz JSON, esto quiere decir que si se ejecuta la petición para recuperar un recurso, éste se devolverá en formato JSON. Las API REST usan una interfaz uniforme, lo que permite el uso de verbos HTTP estándar para realizar operaciones en los recursos. Las operaciones más comunes son GET, POST, PUT y DELETE. Por ejemplo para que un cliente modificase un recurso debería realizar una petición POST, enviando en el cuerpo de la petición los valores cambiados del recurso en formato JSON.

Un servicio REST es '*stateless'*, es decir, cada petición es independiente y se puede ejecutar en cualquier orden, lo que no permite almacenar información de una petición a otra. Sin embargo, se almacena dicha información en el propio recurso permitiendo implementar un servicio mucho más escalable.

Para el desarrollo del servicio API REST de este proyecto se ha utilizado la tecnología Spring Boot y Spring Framework. Spring framework proporciona un modelo completo de programación y configuración para aplicaciones basadas en java (*Spring Framework*, n.d.).Spring Boot es una herramienta que facilita la tarea de configurar y desplegar una aplicación.

Este framework cuenta con numerosas características que lo hace una tecnología excepcional para el desarrollo de servicios API REST, nos centraremos en las más relevantes para este proyecto.

La inyección de dependencias, es un proceso que trata de reducir el acoplamiento y aumentar la independencia entre clases, evitando la creación de nuevas instancias dentro las clases y sustituyéndolas por la inyección de dicha clase. El objeto no busca sus dependencias y no conoce la ubicación o la clase de las mismas, de esta forma las clases se vuelven más fáciles de probar.

Spring soporta '*Data Access Object*' (DAO) lo que facilita el trabajo a las tecnologías de acceso a datos como Hibernate, JPA, JDBC, etc. DAO separa completamente la lógica de acceso de a datos de la lógica de negocio, para ello es necesario crear una estructura de proyecto en la cual existan por un lado los objetos de la lógica de negocio llamados DOT y por otra las clases que definan entidades en base de datos o DAO. Estas clases DAO definen el modelo de datos que queremos tener en nuestro servicio.

Para realizar esta inyección de dependencias y utilizar DAO en un proyecto, es necesario el uso de anotaciones, Spring cuenta con una gran cantidad de anotaciones que permiten controlar los comportamientos de las clases definidas. Por ejemplo, para

realizar la inyección de una clase dentro de otra es necesario utilizar @Autowired, para representar con una clase una entidad de base de datos es necesario utilizar la anotación @Entity. Existen anotaciones de muchos tipos diferentes.

Una vez se ha explicado cómo funciona un servicio API REST en grandes rasgos y algunas características importantes de spring, se va a concretar como se ha definido la API y que endpoints expone el servicio.

Interfaz de programación de aplicaciones (API)

"Una API es un conjunto de definiciones y protocolos que se utiliza para desarrollar e integrar el software de las aplicaciones. API significa interfaz de programación de aplicaciones." (*¿Qué Es Una API?*, n.d.)

Las Apis permiten que cualquier otro sistema haga uso de las funcionalidades de este servicio sin la necesidad de saber cómo se realiza la lógica. De esta forma se facilita en gran medida la integración con servicios externos. El servicio desarrollado para este proyecto provee una API con 7 controladores, los cuales exponen los endpoints correspondientes con su funcionalidad. La mayoría de elementos de esta solución hacen uso de esta API para poder realizar su propia lógica. La interfaz del cliente, donde se muestra el estado de los maceteros, recupera la información necesaria mediante este servicio y sus endpoints expuestos. La interfaz del administrador se comunica con dicho servicio para persistir los datos que el administrador ha introducido a la hora de crear o modificar un cliente. La plataforma IoT persiste en la base de datos la información recibida mediante esta API. A continuación se van a detallar cada uno de los endpoints expuestos.

Client Controller

Listado de operación API REST implementadas para el recurso Client del servicio web. Se entiende como la entidad Client la persona que se va a encargar de gestionar un dispositivo IoT. En el caso concreto de este proyecto, se ha necesitado dar de alta un Client; para de poder monitorizar el estado del macetero A-POT, con el que se evalúa este proyecto en la sección 3 ‘Evaluación de la plataforma‘. Puesto que los Endpoint de este controlador siempre van a trabajar con el recurso Client se quiere describir cada uno de los parámetros de este recurso, en la ilustración 8.

```
[ClienteDTO {
  descripcion        string
                     Descripción del cliente

  id                 integer($int64)
                     Campo autocompletado
  letra              string
                     Campo para identificar el cliente en la interfaz
  thingsboardId      string
                     Identificador asignado por la plataforma ThingsBaord
}]
```

Ilustración 8: DTO Cliente

Identificador	EPAñadirCliente
Método	PUT
Endpoint	/api/client
Descripción	Operación para añadir un nuevo cliente en la Base de Datos.

Ejemplo Entrada	{
	"descripcion": "macetero A-POT grupo 3 " : String
	"letra": "A" : String
	"thingsboardId": " 593d8ac0-c13e-11ea-bd36-25db62324136" : String
	}

Ejemplo Salida	{ "descripcion": "macetero A-POT grupo 3 " : String

	"letra": "A" : String “id”: “1” : Integer "thingsboardId": " 593d8ac0-c13e-11ea-bd36-25db62324136" : String }

Tabla 1: Añadir Cliente. (Elaboración propia)

Identificador	**EPListarCliente**
Método	GET
Endpoint	/api/client
Descripción	Operación para listar todos los clientes de la Base de Datos.
Ejemplo Entrada	---

Ejemplo Salida	[
	{
	"descripcion": "macetero A-POT grupo 3 " : String
	
	"letra": "A" : String
	
	“id”: “1” : Integer
	
	"thingsboardId": " 593d8ac0-c13e-11ea-bd36-25db62324136" : String
	}
	]

Tabla 2: Obtener Clientes. (Elaboración propia)

Identificador	**EPObtenerCliente**
Método	GET
Endpoint	/api/client/{client_id}
Descripción	Operación para buscar un cliente concreto por su identificador del Thingsboard.
Ejemplo Entrada	---
Ejemplo Salida	{ "descripcion": "macetero A-POT grupo 3 " : String "letra": "A" : String "id": "1" : Integer "thingsboardId": " 593d8ac0-c13e-11ea-bd36-25db62324136" : String }

Tabla 3: Obtener Cliente. (Elaboración propia)

Identificador	**EPModificarCliente**
Método	POST
Endpoint	/api/client/{client_id}
Descripción	Operación para realizar la modificación de un cliente.
Ejemplo Entrada	{ "descripcion": "macetero A-POT grupo 3 modificado " : String "letra": "B" : String "thingsboardId": " 593d8ac0-c13e-11ea-bd36-25db62324136" : String }
Ejemplo Salida	{ "descripcion": "macetero A-POT grupo 3 modificado " : String "letra": "B" : String "thingsboardId": " 593d8ac0-c13e-11ea-bd36-25db62324136" : String }

Tabla 4: Modificar Cliente. (Elaboración propia)

Identificador	**EPEliminarCliente**
Método	DELETE
Endpoint	/api/client/{client_id}
Descripción	Operación para eliminar un cliente concreto por su identificador del Thingsboard.
Ejemplo Entrada	----

Ejemplo Salida	{
	"descripcion": "macetero A-POT grupo 3 modificado " : String
	"letra": "B" : String
	“id”: “1”: Integer
	"thingsboardId": " 593d8ac0-c13e-11ea-bd36-25db62324136" : String
	}

Tabla 5: Eliminar Cliente. (Elaboración propia)

Identificador	**EPObtenerClientedeSenor**
Método	GET
Endpoint	/api/client/sensor/{sensor_id}
Descripción	Operación para buscar el cliente donde está instalado el sensor.
Ejemplo Entrada	---

Ejemplo Salida	{
	"descripcion": "macetero A-POT grupo 3" : String
	"letra": "A" : String
	“id”: “1”: Integer
	"thingsboardId": " 593d8ac0-c13e-11ea-bd36-25db62324136" : String
	}

Tabla 6: Obtener Cliente de Sensor. (Elaboración propia)

Configuración Controller

Identificador	**EPObtenerConfiguración**
Método	GET
Endpoint	/config/{device_token}
Descripción	Operación para recuperar los datos de configuración del dispositivo.
Ejemplo Entrada	---
Ejemplo Salida	String (JSON)

Tabla 7: Obtener Configuración Dispositivo. (Elaboración propia)

Identificador	**EPHistorialRiego**
Método	GET
Endpoint	/irrigation/{client_id}
Descripción	Operación para recuperar el historial de riego de un cliente.
Ejemplo Entrada	---

Ejemplo Salida	[
	{
	"action": {
	"actionDesc": "cerrar riego",
	
	"id": 0
	},
	"cliente": {
	"descripcion": "macetero A-POT grupo 3 modificado " : String
	
	"letra": "B" : String
	
	“id”: “1”: Integer
	
	"thingsboardId": " 593d8ac0-c13e-11ea-bd36-25db62324136" : String
	},
	"id": 0,
	"timestamp": "2020-06-02T16:30:59.065Z"
	}
	]

Tabla 8: Obtener Historial Riego. (Elaboración propia)

Sensor Controller

Listado de operación API REST implementadas para el recurso Sensor del servicio web. Se entiende como Sensor el dispositivo electrónico que mide los valores de los parámetros del dispositivo. En el caso concreto del dispositivo IoT utilizado para evaluar este proyecto, se tienen 7 sensores que se encargan de recoger la temperatura, humedad, luminosidad etc. Estos sensores se asocian al cliente cuando este se da de alta en la plataforma. Puesto que los Endpoint de este controlador siempre van a trabajar con el recurso Sensor se quiere describir cada uno de los parámetros de este recurso, en la ilustración 9.

```
[SensorDTO {
  id                  integer($int64)
                      Campo autocompletado
  nombre              string
                      Nombre del sensor
  tipoSensorId        string
                      Tipo de Sensor asignado
}]
```

Ilustración 9: DTO Sensor

Identificador	EPAñadirSensor
Método	PUT
Endpoint	/api/sensors
Descripción	Operación para añadir un nuevo sensor a la Base de datos.
Ejemplo Entrada	{ "nombre": "temperatura_ambiental" : String "tipoSensorId": "Temperatura Ambiental" : SensorType }
Ejemplo Salida	{ “id”: “1” : Integer "nombre": "temperatura_ambiental" : String "tipoSensorId": "Temperatura Ambiental" : SensorType }

Tabla 9: Añadir Sensor. (Elaboración propia)

Identificador	**EPObtenerSensores**
Método	GET
Endpoint	/api/sensors
Descripción	Operación para listar todos los sensores de la Base de Datos.
Ejemplo Entrada	---

Ejemplo Salida	]
	{
	“id”: “1” : Integer
	"nombre": "temperatura_ambiental" : String
	"tipoSensorId": "Temperatura Ambiental" : SensorType
	}
	]

Tabla 10: Obtener Sensores. (Elaboración propia)

Identificador	**EPObtenerSensor**
Método	GET
Endpoint	/api/sensors/{sensor_id}
Descripción	Operación para obtener sensor concreto.
Ejemplo Entrada	---

Ejemplo Salida	{
	“id”: “1” : Integer
	"nombre": "temperatura_ambiental" : String
	"tipoSensorId": "Temperatura Ambiental" : SensorType
	}

Tabla 11: Obtener Sensor. (Elaboración propia)

Identificador	EPObtenerSensoresdeCliente
Método	GET
Endpoint	/api/sensors/client/{cliente_id}
Descripción	Operación para listar todos los sensores de un cliente.
Ejemplo Entrada	---

Ejemplo Salida	[
	{
	“id”: “1” : Integer
	"nombre": "temperatura_ambiental" : String
	"tipoSensorId": "Temperatura Ambiental" : SensorType
	},
	{
	“id”: “2” : Integer
	"nombre": "temperatura_superficial" : String
	"tipoSensorId": "Temperatura Superficial" : SensorType
	}
	]

Tabla 12: Listar Sensores de Cliente. (Elaboración propia)

Sensor_Cliente Controller

Listado de operación API REST implementadas para el recurso Sensor_Cliente del servicio web. Se entiende como Sensor_Cliente la relación entre el recurso Client y el recurso Sensor, de esta forma el sistema conoce a quien pertenece cada sensor. En el caso de este proyecto para la monitorización del macetero A-POT se tendrían 7 entidades 'Sensor_Cliente'en las cuales se relacionaría el cliente con los sensores del dispositivo. Puesto que los Endpoint de este controlador siempre trabajan con el recurso Sensor _Cliente se describe cada uno de los parámetros de este recurso, en la ilustración 10.

```
Sensor_ClienteDTO ⌄ {
    cliente                 ClienteDTO ⌄ {
                                descripcion         string
                                                    Descripción del cliente

                                id                  integer($int64)
                                                    Campo autocompletado
                                letra               string
                                                    Campo para identificar el cliente en la interfaz
                                thingsboardId       string
                                                    Identificador asignado por la plataforma ThingsBaord
                            }
    sensores                SensorDTO ⌄ {
                                id                  integer($int64)
                                                    Campo autocompletado
                                nombre              string
                                                    Nombre del sensor
                                tipoSensorId        string
                                                    Tipo de Sensor asignado
                            }
}
```

Ilustración 10: DTO Sensor_Cliente

Identificador	**EPCrearRelaciónCliente-Sensor**
Método	PUT
Endpoint	/api/sensor-client
Descripción	Operación para asignar a un cliente un sensor de Base de Datos.
Ejemplo Entrada	{ "cliente": { "descripcion": "macetero A-POT grupo 3 " : String "letra": "A" : String “id”: “1” : Integer "thingsboardId": " 593d8ac0-c13e-11ea-bd36-25db62324136" : String } { “id”: “1” : Integer "nombre": "temperatura_ambiental" : String "tipoSensorId": "Temperatura Ambiental" : SensorType }, }
Ejemplo Salida	{ "cliente": { "descripcion": "macetero A-POT grupo 3 " : String "letra": "A" : String “id”: “1” : Integer "thingsboardId": " 593d8ac0-c13e-11ea-bd36-25db62324136" : String } { “id”: “1” : Integer "nombre": "temperatura_ambiental" : String "tipoSensorId": "Temperatura Ambiental" : SensorType }, }

Tabla 13: Crear Relación Sensor – Cliente. (Elaboración propia)

Event Controller

Listado de operación API REST implementadas para el recurso Event del servicio web. Se entiende por Event cualquier acción que se realice sobre el sistema. Un ejemplo de esta entidad, es cuando la plataforma IoT registra un evento de ‘abrir riego’ o de ‘cerrar riego’, este evento se persiste en la base de datos mediante esta entidad. Puesto que los Endpoint de este controlador siempre trabajan con el recurso Event se describe cada uno de los parámetros de este recurso, en la ilustración 11.

```
EventsDTO ⌄ {
    action                  Actions > {...}
    cliente                 Cliente > {...}
    id                      integer($int64)
                            Campo autocompletado
    timestamp               string($date-time)
                            Instante en el que se registro el evento
}
```

Ilustración 11: DTO Events

Identificador	**EPRegistrarEvento**
Método	PUT
Endpoint	/api/event/{action}/{client_id}
Descripción	Operación para registrar un nuevo evento en Base de Datos.
Ejemplo Entrada	---

Ejemplo Salida	{
	"action": {
	"actionDesc": "cerrar riego",
	"id": 1
	},
	{
	"cliente": {
	"descripcion": "macetero A-POT grupo 3 " : String
	
	"letra": "A" : String
	
	“id”: “1” : Integer
	
	"thingsboardId": " 593d8ac0-c13e-11ea-bd36-25db62324136" : String
	}
	}

Tabla 14: Registrar Evento. (Elaboración propia)

TimeLine Controller

Listado de operación API REST implementadas para el recurso TimeLine del servicio web. Se entiende por TimeLine el registro del valor tomado por un sensor concreto en un instante de tiempo. Esta entidad se persiste mediante las reglas implementadas en la plataforma IoT, en el caso de este proyecto, se almacenan las variables tomadas por los sensores del dispositivo descrito en la sección 3.1 'Evaluación del sistema con un único cliente IoT' Puesto que los Endpoint de este controlador siempre trabajan con el recurso TimeLine se describe cada uno de los parámetros de este recurso, en la ilustración 12.

```
[TimeLineDTO {
  fecha        string($date-time)
               Instante en el que se almacena el valor
  id           integer($int64)
               Campo autocompletado
  sensor       SensorDTO > {...}
  valor        string
               Valor recogido por el sensor
}]
```

Ilustración 12: DTO TimeLine

Identificador	**EPObtenerValoresSensor**
Método	GET
Endpoint	/api/timeLine/{sensor_id}
Descripción	Operación para listar los 100 últimos valores recogidos por un sensor.
Ejemplo Entrada	---

Ejemplo Salida	[
	{
	"fecha": "2020-06-22T20:15:01.139Z",
	"id": 1 : string,
	"sensor": {
	"id": "1" : Integer
	"nombre": "temperatura_ambiental" : String
	"tipoSensorId": "Temperatura Ambiental" : SensorType
	},
	"valor": "20" : String
	}
	]

Tabla 15: Obtener valores de Sensor

Sensor Type Controller

Listado de operación API REST implementadas para el recurso Sensor Type del servicio web. Se entiende como Sensor_Type el catálogo precargado de tipos de sensores que se pueden dar de alta en el sistema. Puesto que los Endpoint de este controlador siempre trabajan con el recurso Sensor_Type se describe cada uno de los parámetros de este recurso, en la ilustración 13.

```
[TipoSensorDTO {
  descripcion        string
                     Descripción del tipo de sensor

  id                 integer($int64)
                     Campo autocompletado
  nombre             string
                     Nombre del tipo de sensor
                     Enum:
                       Array [ 9 ]
}]
```

Ilustración 13: DTO TipoSensor

Identificador	**EPListarTiposSensor**
Método	GET
Endpoint	/api/tipoSensor
Descripción	Operación para listar todos los tipos de sensores de la Base de Datos.
Ejemplo Entrada	---

Ejemplo Salida	[
	{
	"descripcion": "Sensor detector de agua" : String,
	"id": 0 : Integer,
	"nombre": "agua" : String
	}
	]

Tabla 16: Obtener Tipos de Sensor. (Elaboración propia)

Identificador	EPObtenerTipoSensor
Método	GET
Endpoint	/api/tipoSensor/{tipoSensor_id}
Descripción	Operación para buscar un tipo de sensor concreto de la Base de Datos.
Ejemplo Entrada	---
Ejemplo Salida	{ "descripcion": "Sensor detector de agua" : String, "id": 0 : Integer, "nombre": "agua" : String }

Tabla 17: Obtener Tipo Sensor Concreto. (Elaboración propia)

2.1.4. Plataforma IoT

Como ya se ha comentado anteriormente ThingsBoard es la plataforma IoT seleccionada para este proyecto.

ThingsBoard tiene varias funciones en esta solución, no obstante el propósito general de esta plataforma es realizar toda la lógica que mantendrá la planta en su estado óptimo. Para ello recibe los datos recogidos por el sistema hardware, los cuales pasarán por un flujo definido por la el motor de reglas de Thingsboard. ¿Qué es este motor de reglas? El motor de reglas es un framework fácil de usar para construir flujos de trabajo basados en eventos.(The ThingsBoard Authors, 2020). Este motor tiene 3 elementos principales:

1. los **mensajes** o cualquier evento entrante proveniente de un dispositivo, de otro servicio o de un evento del propio thingsboard;
2. los **nodos**, que son el elemento encargado de realizar una función sobre los datos de este mensaje entrante, es decir, el mensaje recorrerá los nodos correspondientes y en cada uno de ellos se realizará una función. Existe una gran variedad de nodos, lo que permite implementar cualquier tipo de lógica.
3. las **cadenas de reglas**, las cuales alojan un número de nodos cualesquiera relacionados entre sí y a su vez estas cadenas de reglas pueden relacionarse entre sí, permitiendo realizar cualquier tipo de flujo.

Para el desarrollo de la lógica que debe realizar ThingsBoard se han implementado seis cadenas de reglas, cada una de ellas realiza una serie de funciones diferentes, desde el control del estado de la planta hasta el aviso de algún fallo en el sistema a través de una alerta. De esta forma si el riego de planta se ha roto o no se está midiendo bien algún parámetro, el usuario será advertido y podrá realizar las acciones que considere necesarias. Otra función que realizan algunas de estas reglas es la de registrar eventos, es decir, cada vez que se abre o se cierra el riego, se modifica algún parámetro, se activa el riego manual o cualquier otro tipo de acción la plataforma se comunica con el servicio API REST para registrar estas acciones.

Por último, estas reglas implementadas también se encargan de persistir los datos en la base de datos creada y explicada en la sección de ‘Base de Datos’. Para conseguir persistir estos datos realiza una petición al servicio API REST de igual forma que cuando se registra un evento.

Reglas

Se han diseñado e implementado siete reglas para realizar todas las funcionalidades de plataforma IoT. Pero antes de entrar en el detalle de estas reglas es importante destacar que en el desarrollo de las mismas se han definido seis **atributos asociados al flujo de los datos para parametrizar ciertos aspectos de la lógica**.

Estos atributos se pueden dividir en dos grupos:

- aquellos que el cliente puede controlar y modificar
- los que se modifican durante el flujo de vida de los datos.

Los siguientes parámetros permiten al usuario controlar algunos factores del funcionamiento del sistema. '*T_DESCONEXION_MAX*' define el tiempo que debe pasar antes de afirmar que el dispositivo no tiene conexión wifi. Por ejemplo, en el caso del macetero implementado para este proyecto era necesario enviar un mensaje al bedel en el caso de que el dispositivo perdiera la conexión con la plataforma. Para ello se ha establecido el tiempo máximo de desconexión en 30 minutos, de esta forma el bedel será avisado si el dispositivo no envía información en 30 minutos, pudiendo ir al dispositivo y reiniciarlo. '*T_MUESTREO*', este parámetro establece la frecuencia entre las lecturas que realiza la maceta de los sensores y envía los datos a la plataforma, es decir, es el tiempo que la placa ESP32 permanecerá dormida entre lectura y lectura. En el caso del macetero A-POT se estableció el tiempo de muestreo en 60 segundos, esto quiere decir que este dispositivo enviará información a la plataforma IoT cada 60 segundos. Esto permite a la plataforma llevar el control de si el dispositivo sigue conectado o no.

Este grupo de parámetros se modificado durante la ejecución de la lógica. '*T_DESCONEXION'* es un contador que se encarga de controlar el tiempo que pasa entre lectura y lectura del dispositivo IoT. '*T_RIEGO_MACETA_X*' este parámetro se establece con el menor tiempo de riego posible que necesite la maceta mediante las reglas implementadas. '*T_RIEGO_ACTUAL*' es un contador que mide el tiempo que se está regando la planta y que junto con el atributo anterior controlan y modifican el tiempo de riego para optimizarlo al máximo.

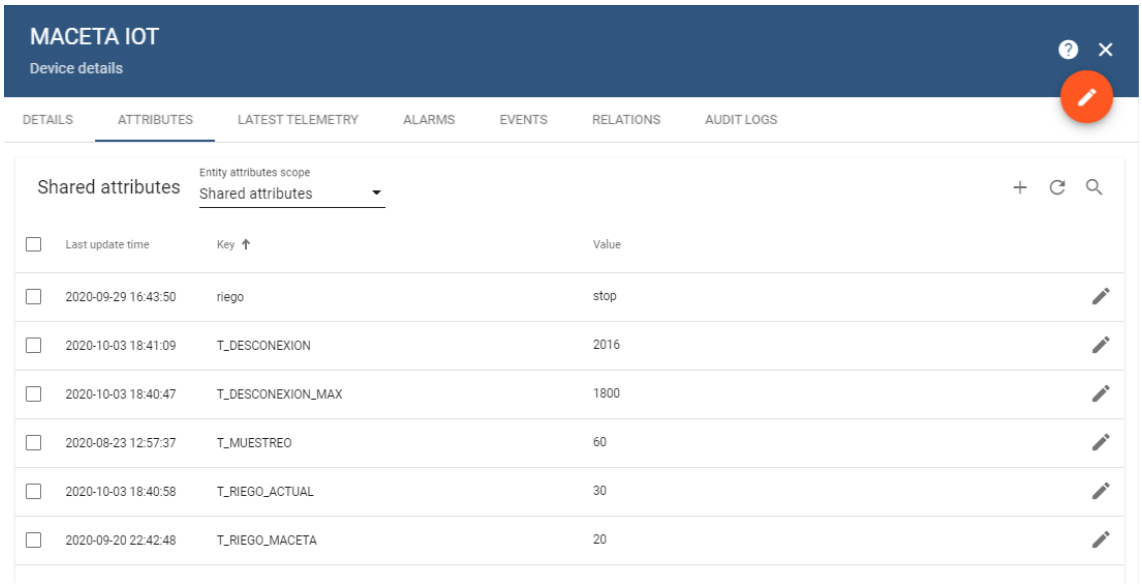


Ilustración 14: Atributos del dispositivo

Todo este conjunto de parámetros controlan, en cierta manera, el comportamiento del sistema ejecutado por las siete reglas que se definen a continuación:

Regla Principal

Esta cadena de reglas es la encargada de recibir los datos entrantes del dispositivo IoT y enviarle dicho mensaje al resto de cadenas de reglas. Además de comunicar el menaje entrante a las otras reglas, esta regla principal reflejada en la ilustración 15, se encarga de enviar un mensaje cada segundo a la regla 5 para el control del tiempo de desconexión y de realizar la petición al servicio REST para que persista la información en la base de datos.

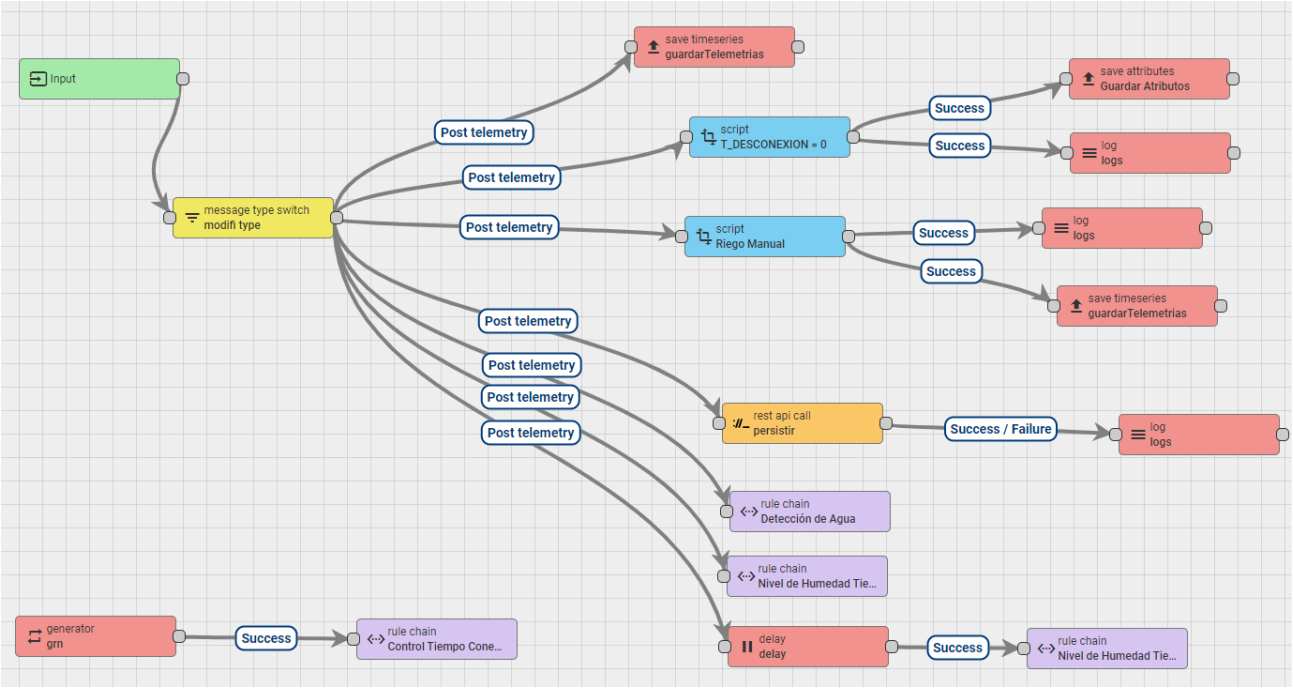


Ilustración 15: Regla principal. (Elaboración propia)

Para llevar el contador del tiempo tiene un generador que cada segundo envía un mensaje a la regla 5, 'Control Tiempo de Conexión', la cual se describe en esta sub sección y se encargará de realizar la lógica necesaria para saber si el dispositivo esta desconectado o no. Para ello la regla 5 incrementará en 1 el valor del atributo 'T_DESCONEXION' al recibir el mensaje de este generador, este contador se reseteará a 0 mediante el script "T_DESCONEXION = 0" de la regla principal. Por otro lado esta misma regla establece la telemetría 'riegoManual' a '*false*', para indicar que el riego manual no es necesario ya que la plataforma recibe información y es esta la encarga de controlar el riego. En la ilustración 16 se observan los dos scripts y el generador que se encargan de realizar estas tareas descritas.

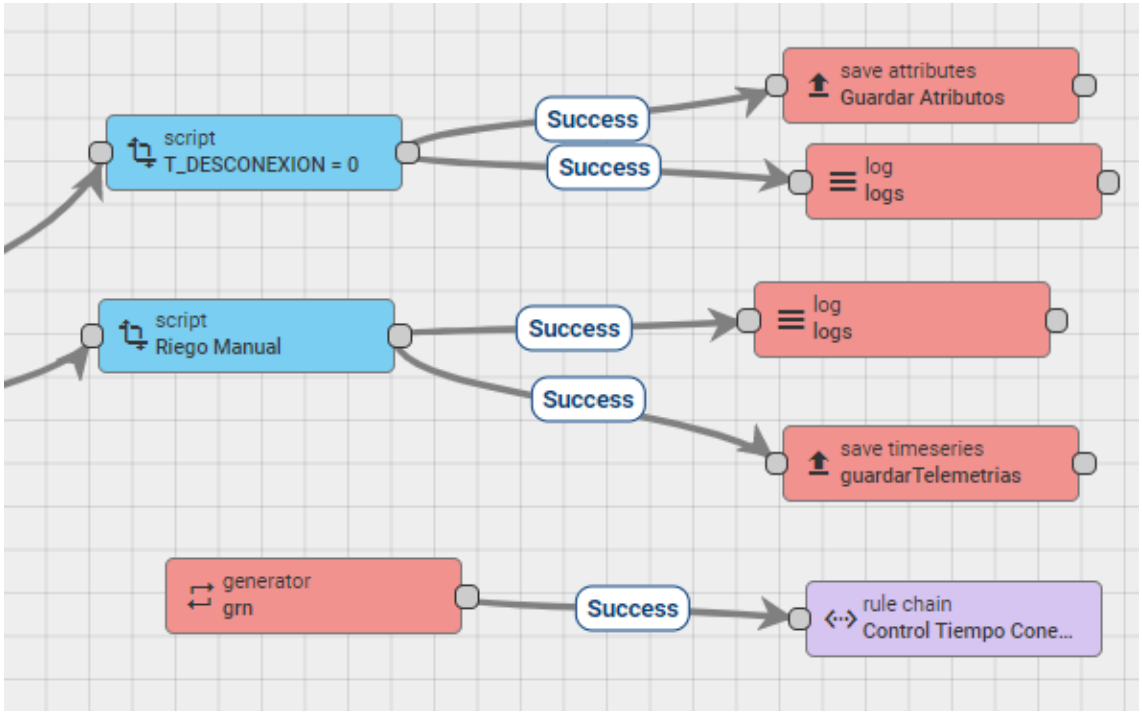


Ilustración 16: Scripts de control de desconexión. (Elaboración propia)[9]

Como se ha comentado anteriormente esta regla también se encarga de realizar la petición http al servicio REST para persistir los datos, para ello se ejecuta el nodo de tipo 'rest api call' que se puede observar en la ilustración 17. En dicha ilustración de se puede ver el endpoint al que se 'ataca' y con el tipo de petición correspondiente, en este caso, una petición de POST y con las cabeceras necesarias para que el servicio sea capaz de almacenar los datos.

[9] http://iot.etsisi.upm.es:8080/ruleChains/d027ae40-d0de-11ea-b07b-a35d5148ed00

Name *
persisitir

☐ Debug mode

Endpoint URL pattern *
http://localhost:8090/api/persistir
HTTP URL address pattern, use ${metaKeyName} to substitute variables from metadata

Request method
POST

☐ Use simple client HTTP factory

Read timeout in millis
0
The value of 0 means an infinite timeout

Max number of parallel requests
0
The value of 0 specifies no limit in parallel processing

Headers

Use <code>${metaKeyName}</code> in header/value fields to substitute variables from metadata

Header	Value	
device	75a6f2f0-c13e-11ea-bd36-25db62324136	×
cliente	593d8ac0-c13e-11ea-bd36-25db62324136	×

Ilustración 17: HTTP Request para persistir los datos. (Elaboración propia)

Regla 1: Detección de agua

La función de esta cadena de reglas es evitar que la planta se riegue en exceso. Para ello, el sensor de agua situado debajo de la maceta identifica cuando el agua rebosa por debajo de la maceta, y por tanto, es hora de cerrar el riego. El dispositivo A-POT se encuentra instalado en la casa del cliente, las lecturas de los sensores informan que la plata necesita regarse, por tanto el sistema realiza la lógica necesaria para abrir el riego. Pasado un tiempo de riego X, empieza a salir agua por debajo de la maceta, por lo que el sensor detector de agua detecta agua y envía la información a la plataforma IoT, donde se realiza la lógica para cerrar el riego. Para implementar esta funcionalidad en la ilustración 18 se aprecia cómo se conectan una serie de nodos en cada uno de los cuales se realiza una función.

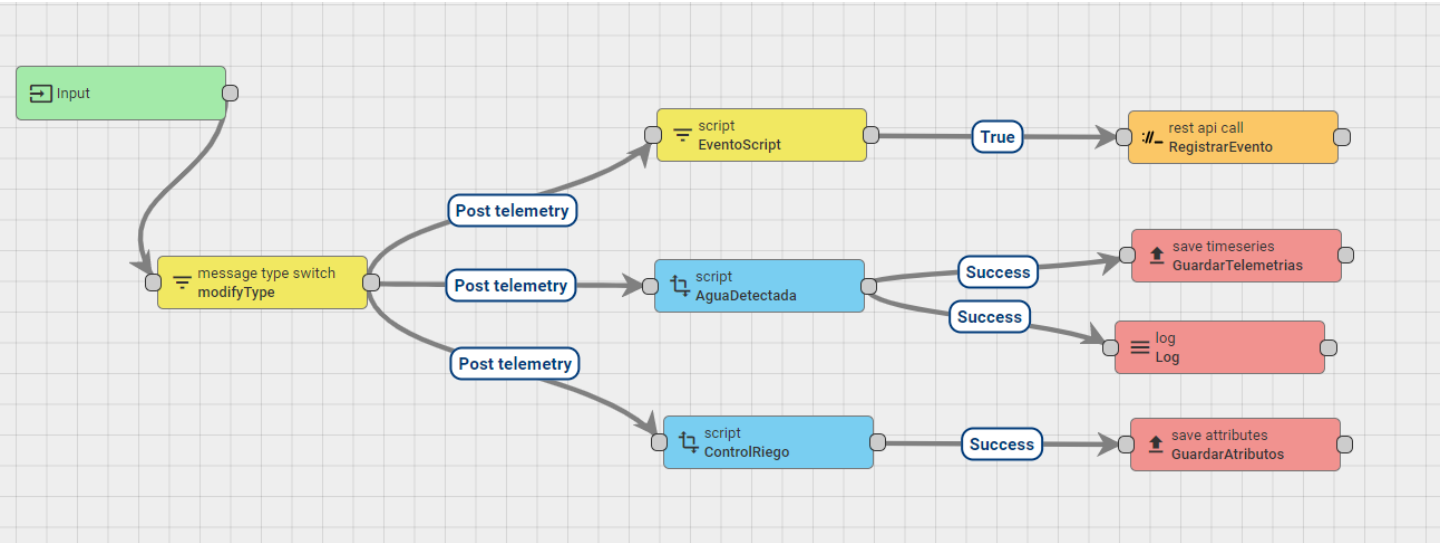


Ilustración 18: Detección de agua. (Elaboración propia)

En primer lugar, tenemos un bloque que será común a la mayoría de reglas que se describen en esta sección. Este bloque está formado por dos nodos, el nodo 'input' y el nodo 'modifyType' del tipo '*message type switch*'. El nodo input recibe el mensaje, de la regla pincipal y lo envía al nodo modifyType donde se cambiará el tipo de mensaje por el deseado.Esta plataforma IoT, trabaja con múltiples tipos de mensaje por lo que es importante tener en cuenta cual es la función que vamos a realizar y a que nodos vamos a enviar dicho mensaje. Esto se debe a que algunos de los nodos que otorga thingsboard deben recibir un tipo concreto, como puede ser el nodo 'saveTelemetry' que necesita recibir un mensaje del tipo Post Telemetry o el nodo 'saveAttribute' que necesita recibir el tipo Post Attributes.

Una vez se ha modificado el tipo de mensaje entrante por el deseado, se envía dicho mensaje al siguiente nodo, en este caso concreto al nodo '*aguaDetectada*' de tipo script. Este tipo de nodos permite implementar una función en JavaScript para realizar la lógica deseada. Esta función reflejada en la ilustración 19 comprueba si el sensor de agua envía un valor positivo, lo que significaría que está saliendo agua por debajo de la maceta, en cuyo caso se enviará un mensaje al nodo 'guardarTelemetrias' del tipo 'saveTimeseries' con los valores que se desean modificar. Este nodo 'guardarTelemetrias' recibirá el mensaje y persisitirá en la base de datos de la plataforma IoTlos valores enviados modificando asi el valor del rele y de los leds en la pantalla de las telemetrías del dispositivo.

```
1  if(msg.agua == 1){
2  var msg = {
3     rele:0,
4     leds:0,
5  };
6  }
7  var msgType =  "POST_TELEMETRY_REQUEST";
8  return {msg: msg, msgType: msgType};
9
```

}

Ilustración 19: Script 'Agua Detectada'. (Elaboración propia)

Por otro lado, cuando el mensaje es modificado en el bloque incial, también se envía a otro script que realiza una comparación análoga al anterior, no obstante, el mensaje saliente es diferente y se envía a un nodo del tipo 'save attributes'. Puesto que el mensaje se envía a un nodo del tipo 'save attributes' es importante definir el tipo de mensaje como 'POST_ATTRIBUTES_REQUEST' para que dicho nodo sea capaz de comprender el mensaje y modificar el valor del atributo 'riego', estableciendo su valor con 'stop', tal y como se refleja en la ilustración 20.

```
if(msg.agua == 1){
var msg = {
riego:"stop",
};
}else var msg = {};
var msgType =  "POST_ATTRIBUTES_REQUEST";
return {msg: msg, msgType: msgType};
```

Ilustración 20: Script 'controlRiego'. (Elaboración propia)

Otra función importante que realiza esta regla y la cual será común para muchas de ellas es, registrar un evento. Como se ha comentado anteriormente, el servicio REST implementado permite llevar un histórico de todas las acciones que se realizan sobre la planta. En este caso se ha implementado otro flujo para el mensaje entrante de la regla 1 en el cual se registra el evento de 'cerrar riego', junto con la hora que se realizó. Para ello, se ha utilizado un nodo de tipo 'script', sin embargo, este nodo no devolverá un mensaje sino que comprobará una condición y devolverá verdadero o falso. En la ilustración 21 se puede ver que la comprobación realizada es la misma que en los script anteriores pero en vez de devolver un mensaje, si la condición es verdadera, ejecutará el nodo de tipo 'rest api call' de la ilustración 22.

```
return msg.agua == 1;
```

Ilustración 21: Script Evento. (Elaboración propia)

Aunque no sea parte de la lógica encargada de controlar el estado de la maceta es una función muy útil por si en algún momento el usuario quiere ver las acciones que se realizan sobre la planta, para replicarla en otra o para evitar seguir el mismo proceso en caso de ser erróneo.

El nodo ‘registrarEvento’ realiza una llamada al servicio API REST desarrollado para registrar el evento de 'cerrar el riego’. Como se observa en la ilustración 22 la operación que se ejecuta es la de ‘Registrar Evento’ descrita en la sección de ‘Servicio Web’, la cual persistirá en base de datos el evento de ‘cerrar riego’ junto con la fecha que se realizó esa acción.

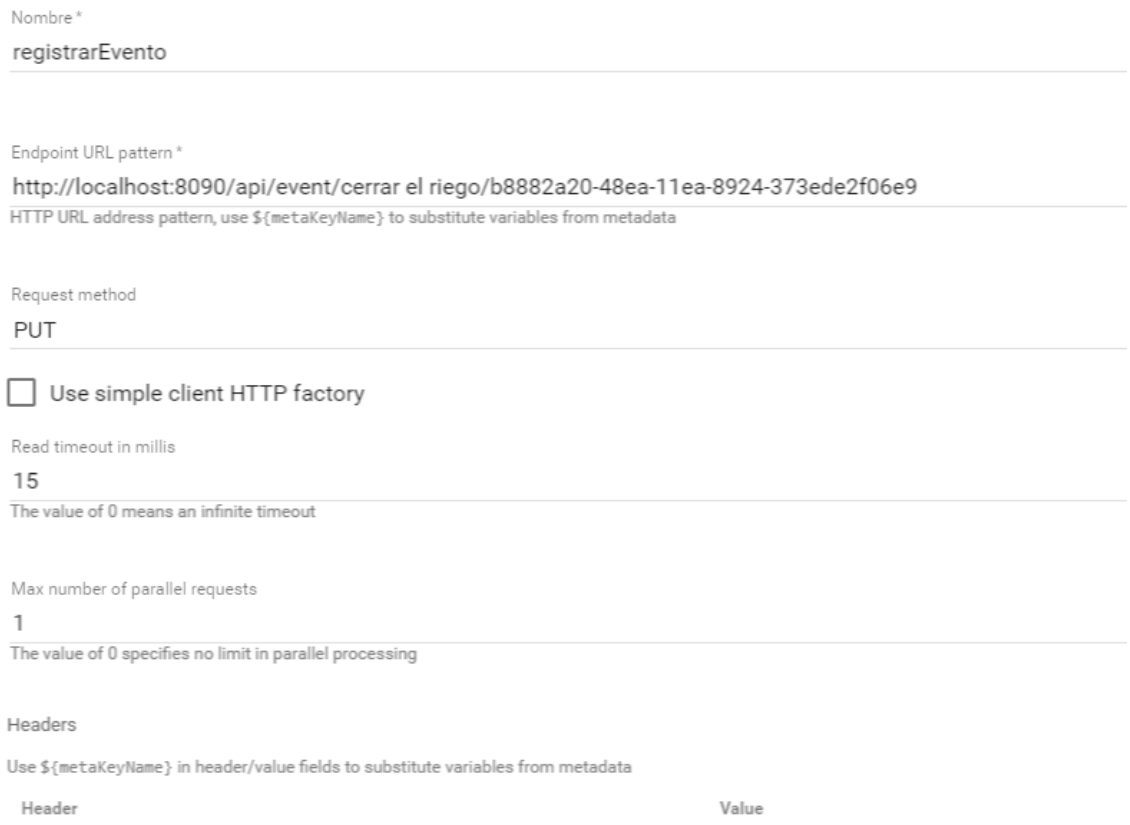


Ilustración 22: Ret Api Call RegistrarEvento. (Elaboración propia)

Regla 2: Nivel de humedad inferior

Esta cadena de reglas controla el nivel de humedad que tiene la parte inferior de la tierra de la maceta, si el nivel supera el valor establecido significa que la planta estaría suficientemente regada y habría que detener el riego. La regla recibirá como mensaje entrante los datos enviados por la regla principal y comprobará si el nivel de humedad de la tierra es superior a un valor establecido. El macetero A-POT se encuentra instalado en la casa del cliente, las lectura del sensor de humedad de la tierra de la parte inferior envía un valor por encima del establecido indicando que la planta esta regada. Por tanto el sistema realiza la lógica necesaria para cerrar el riego y registra un evento mediante el servicio REST. En la ilustración 23, se ven los nodos utilizados para la implementación de esta regla, estos son análogos a la regla 'Detección de agua', exceptuando las funciones desarrolladas en los scripts.

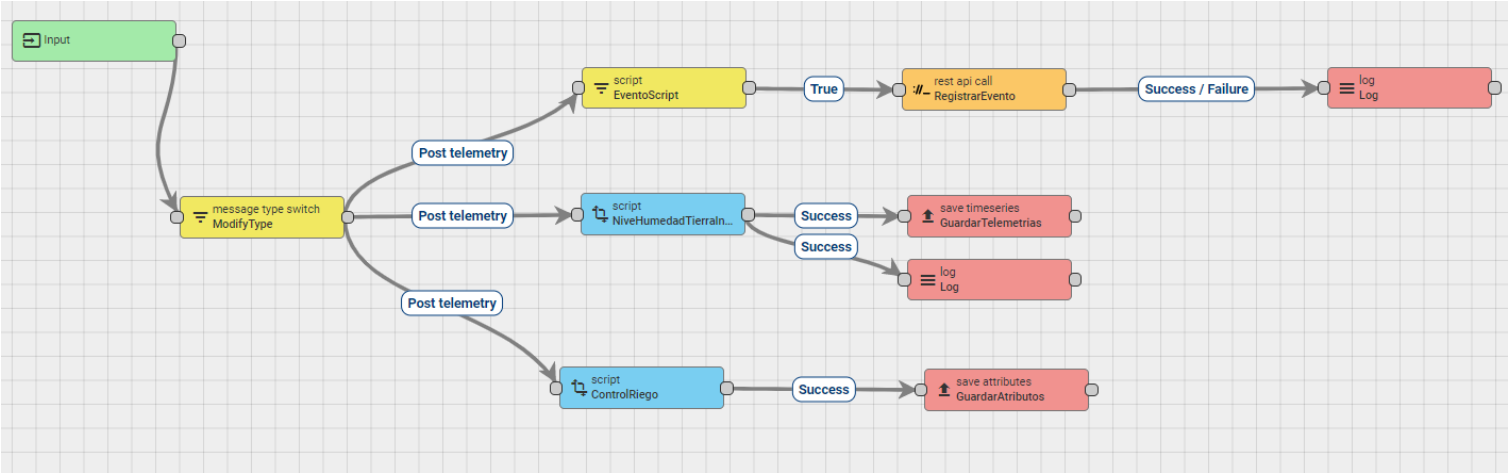


Ilustración 23: Nivel de humedad inferior. (Elaboración propia)

El bloque inicial que recibe el mensaje y modifica su tipo es común a la regla de 'Detección de Agua'. En la ilustración 24, se puede ver el script de *'NivelHumedadTierraInferior'* donde se comprueba si el sensor de humedad de la tierra situado en la parte inferior de la maceta recoge un valor superior al 50%. Si el nivel de humedad es superior al establecido se enviará un mensaje al nodo 'guardar' donde se persistirán los valores enviados en dicho mensaje. De esta forma se modifican los valores de la telemetría 'rele' y 'leds' reflejando que el dispositivo debe cerrar el riego.

```
1 if(msg.humedad_suelo >= 50){
2     var msg = {rele: 0, leds: 0,};
3 }
4 var msgType = "POST_TELEMETRY_REQUEST"
5 return {
6 msg: msg,
7 metadata: metadata,
8 msgType: msgType};
```

Ilustración 24: Script 'humedadTierraInferior'. (Elaboración propia)

Otra funcionalidad de esta regla es modificar el atributo de ‘riego’ en caso de que la humedad supere el valor, para ello tiene un segundo script ‘*ControlRiego’* donde comprueba si el nivel de humedad supera el 50%. En caso de que lo supere enviará un mensaje de tipo ‘POST_ATTRIBUTES_REQUEST’ al nodo ‘save’ para modificar el valor del atributo ‘riego’, al igual que lo hace la regla 1.

Adicionalmente esta regla realiza una tercera función para registrar el evento de ‘cerrar riego’ en el sistema. La forma de ejecutar esta función es análoga a la regla ‘Deteccion de Agua’, el mensaje pasará por un nodo de tipo script donde se comprobará si el nivel de la humedad de la tierra es superior al establecido, en caso afirmativo se ejecutará el script de ‘registrarEvento’ de la iluestracion 20.

Regla 3: Humedad Superior Insuficiente

Esta regla se encarga de comprobar si el nivel de humedad de la tierra de la parte superior baja del nivel establecido, en caso de que el nivel fuese inferior significaría que la planta necesita ser regada. Esta regla recibirá el mensaje enviado por el dispositivo y al igual que las reglas anteriores modificará el tipo de este mensaje para trabajar con los datos en los script implementados. Además de comprobar el nivel de humedad, esta regla también se encarga de llevar el contador de tiempo para saber cuánto tiempo se está regando la planta. El dispositivo A-POT se encuentra instalado en la casa del cliente, la lectura del sensor de humedad de la tierra de la parte superior informa a la plataforma con un valor por debajo del establecido indicando que la planta necesita regarse, por tanto el sistema realiza la lógica necesaria para abrir el riego. Además de abrir el riego la plataforma IoT inicia un contador para controlar el tiempo que la plata se está regando. Para implementar estas funcionalidades se puede ver en la imagen 23 los nodos relacionados que se describen a continuación.

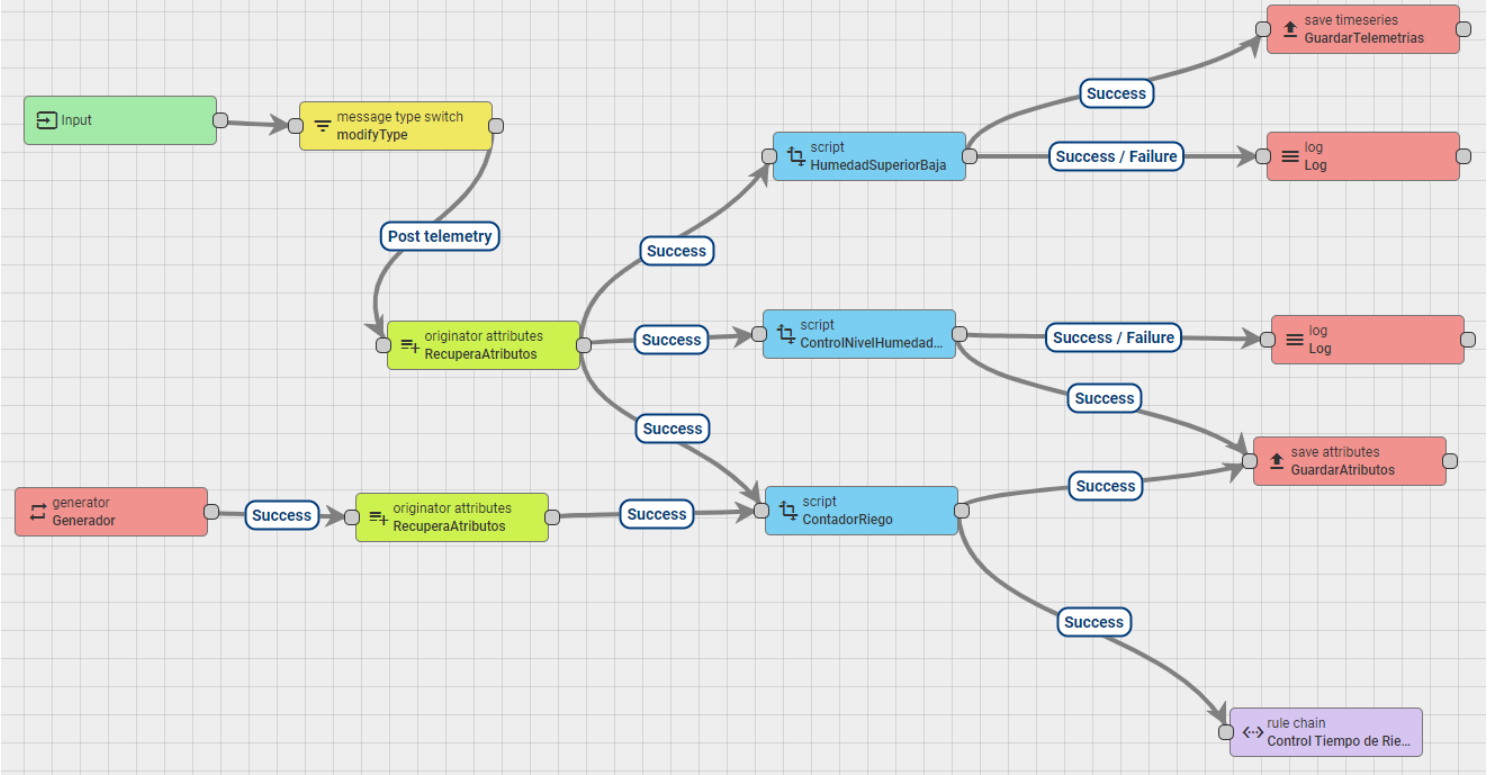


Ilustración 25: Humedad Superior Insuficiente. (Elaboración propia)

Una vez se ha modificado el tipo de mensaje entrante, lo primero que ejecuta esta regla es el nodo del tipo ‘originator attributes’ llamado ‘*RecuperarAtributos*, este nodo sirve para añadir al mensaje los valores de cualquier atributo que esté asociado al dispositivo. Para este caso en concreto, es necesario recoger los valores de los atributos ‘T_RIEGO_ACTUAL’ y ‘riego’, esto valores recogidos se añaden al mensaje, sin embargo, lo hacen como metadatos de este y con un prefijo, variable, dependiendo del tipo de atributo que sea. Como los atributos que se quieren recoger son compartidos, el nodo añade el prefijo ‘shared’ al atributo, por lo que para poder trabajar con él en el script necesitaremos referirnos al parámetro como ‘*metada.shared_riego’. E*n la ilustración 26 se puede ver un ejemplo de cómo se recogen los atributos con el nodo de tipo ‘originator attributes’.

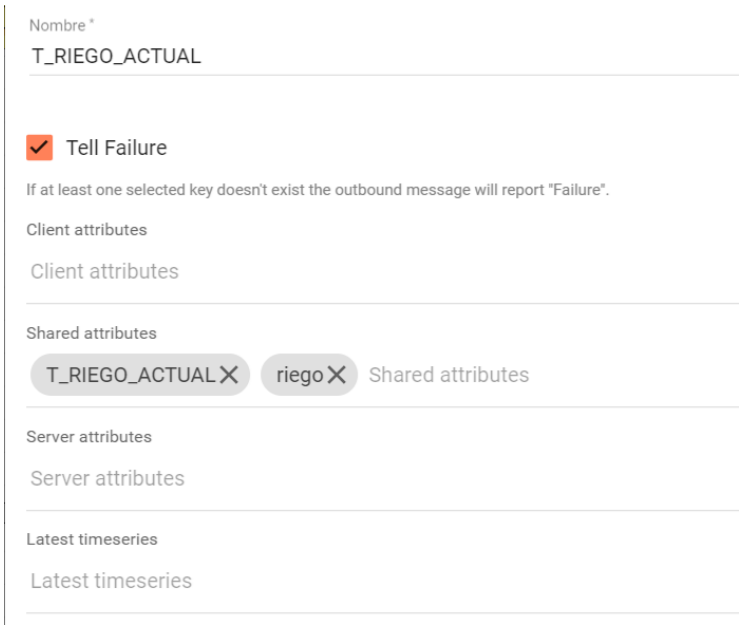


Ilustración 26: Originador de Atributos. (Elaboración propia)

El nodo ‘HumedadSuperiorBaja’ recibirá el mensaje entrante con los dos atributos añadidos anteriormente y comprobará si el nivel de humedad de la tierra de la parte superior es inferior al establecido. Adicionalmente se comprueban 2 variables más, primero si el valor del atributo riego es igual a ‘stop’ por lo que el riego se encuentra detenido actualmente y segundo, si el sensor detector de agua situado debajo de la maceta no ha detectado agua. De esta forma nos aseguramos que la planta necesita ser regada, ya que se puede dar el caso de que la planta haya recibido mucho sol y se haya secado la parte superior de la tierra, pero sin embargo, la parte baja de la

maceta siga con agua suficiente. Una vez se realizan todas estas comprobaciones podemos asegurar que la planta necesita agua y en este caso enviaremos un mensaje al nodo ‘guardar’ para modificar las telemetrías relé’ y ‘leds’ que posteriormente el dispositivo recogerá actuará en consecuencia.

```
if(msg.humedad_suelo <= 30 && metadata.shared_riego === "stop" && msg.agua === 0){
    var msg = { rele: 1, leds: "azul" };
}
var msgType = "POST_TELEMETRY_REQUEST"
return {
msg: msg,
metadata: metadata,
msgType: msgType};
```

Ilustración 27: Script Humedad Superior Baja. (Elaboración propia)

Por otro lado, tenemos el nodo ‘controlRiego’, esta función realiza una comprobación análoga al script anterior, sin embargo, su función es completamente diferente. Este script genera un mensaje para enviar a un nodo del tipo ‘save attributes’ y modificar los atributos ‘T_RIEGO_ACTUAL’ y ‘riego’ de esta forma se indica que se ha abierto el riego y se ha iniciado el contador para controlar el tiempo de riego.

```
if (msg.humedad_suelo <= 30 && metadata.shared_riego === "stop" && msg
    .agua === 0) {
    var msg = {
        T_RIEGO_ACTUAL: 0,
        riego: "start"
    };
} else var msg = {}
var msgType = "POST_ATTRIBUTES_REQUEST"
return {
    msg: msg,
    metadata: metadata,
    msgType: msgType
};
```

Ilustración 28: Script ‘controlRiego’. (Elaboración propia)

Por último esta regla se encarga de llevar el contador del riego ya que es la misma que lo inicia. Para realizar esta funcionalidad se ha añadido un nodo '*generator*' el cual es capaz de generar un mensaje cualquiera cada X segundos, sin embargo, para este caso no genera ningún mensaje si no que sirve para cronometrar el tiempo de riego. Para ello cada segundo envía un mensaje vacío al que se añadirán los atributos de 'T_RIEGO_ACTUAL' y 'riego, tal y como se refleja en ilustración 28. Este mensaje con los atributos añadidos se envían al script 'contador' que se encargará de aumentar en una unidad el atributo 'T_RIEGO_ACTUAL', como se observa en la ilustración 29 y enviar el mensaje al nodo 'guardar' para modificar dicho atributo. Además dicho mensaje también es enviado a la regla 4 'Control Tiempo de riego', la cual se encarga de realizar la lógica sobre el tiempo de riego de la maceta.

Un detalle importante de este script es la forma en la que se trabaja con el valor del atributo 'T_RIEGO_ACTUAL', puesto que el nodo 'recojeAtributos' recupera el valor en forma de string y es necesario parsearlo para que la función lo tome como un entero y poder realizar la función deseada.

```
1  var riego = metadata.shared_riego;
2  var actual = parseInt(metadata.shared_T_RIEGO_ACTUAL)
3  if(riego === "start"){
4  var msg = {
5     T_RIEGO_ACTUAL:actual +1,
6      };
7  }
8  var msgType = "POST_ATTRIBUTES_REQUEST";
9  return {msg: msg, msgType: msgType};
```

Ilustración 29: Script 'contador'. (Elaboración propia)

Estas tres reglas descritas son dependientes de los valores recogidos por lo sensores por lo que se ejecutan cuando la placa envíe la información de los sensores a la plataforma.

A continuación, se detallan otras reglas independientes de estos sensores y que se encargarán de controlar ciertos aspectos del sistema.

Regla 4: Control Tiempo de Riego

Esta regla se encarga de llevar el control sobre el atributo 'T_RIEGO_MACETA_X', para que éste siempre tenga el menor tiempo que se debe regar una maceta. En una primera instancia este atributo es parametrizable por el usuario y establecerá el tiempo que desea regar su planta, no obstante, si en algún momento la planta necesita menos tiempo para regarse el valor del atributo 'T_RIEGO_MACETA_X' se actualizará con dicho tiempo de riego. Esta funcionalidad permite llevar controlado el tiempo de riego del dispositivo evitando el desperdicio de agua.

A diferencia de las reglas anteriores, esta regla no recibe el mensaje directamente desde el dispositivo, sino que es ejecutada desde otra regla 'Humedad Superior Insuficiente'. Esto es debido a que esta regla solo se debe ejecutar una vez que el riego de la planta ha finalizado, de esta forma se podrá comprobar si el tiempo riego es menor que el establecido.

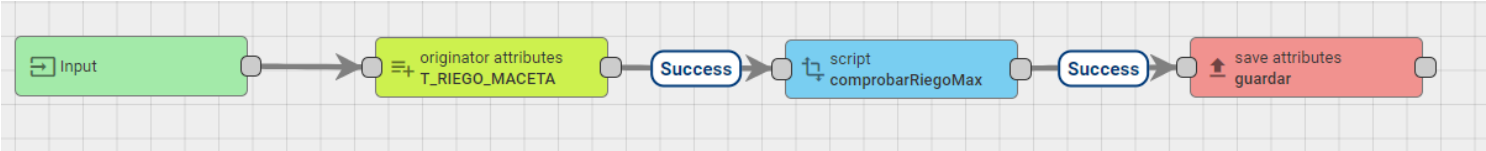


Ilustración 30: Control Tiempo de Riego. (Elaboración propia)

Para realizar la funcionalidad de esta regla en primer lugar se debe recoger los atributos 'T_RIEGO_ACTUAL', 'T_RIEGO_MACETA' y 'riego' para ello en la ilustración 30 se aprecia como el nodo del tipo 'originator attirbutes' recoge el valor de estos atributos. Una vez estos parámetros se añaden al mensaje éste se envía a un nodo de tipo script donde se ejecuta la función reflejada en la ilustración 31.

La función de este script es comprobar si el valor de 'T_RIEGO_ACTUAL' es menor a 'T_RIEGO_MACETA', y si además el riego está detenido, en caso afirmativo significaría que la planta ha tardado menos en regarse de lo que el cliente ha establecido y se debe actualizar el valor de este parámetro. Para actualizar este valor el nodo 'comprobarRiegoMax' enviará un mensaje al nodo 'guardar' para modificar los valores de los atributos.

```
if(parseInt(metadata.shared_T_RIEGO_ACTUAL) <= parseInt(metadata.shared_T_RIEGO_MACETA) && metadata
    .shared_riego =="stop"){
    var msg = {
       T_RIEGO_MACETA:metadata.shared_T_RIEGO_ACTUAL,
    };
}else{var msg = {};}
var msgType = "POST_ATTRIBUTES_REQUEST";
return {msg: msg, msgType: msgType};
```

Ilustración 31: Script 'comprobarRiegoMax'. (Elaboración propia)

En la ilustración 31 se observa cómo ha sido necesario parsear los valores recuperados para los atributos T_RIEGO_ACTUAL', 'T_RIEGO_MACETA', esto se debe al mismo factor que la regla anterior, los nodos del tipo 'originator attributes' añaden los valores de los atributos como si fueran de tipo '*strings*'.

Regla 5: Control Tiempo Conexión

La función de esta regla es controlar que el dispositivo IoT no pierde la conexión con la plataforma IoT durante un período de tiempo X. Para ello se define el atributo 'T_MUESTREO' que representa el tiempo que debe pasar entre mensaje y mensaje recibido desde el dispositivo. Adicionalmente se define el atributo 'T_DESCONEXION' que actúa como contador para cronometrar el tiempo que el dispositivo esta desconectado de la plataforma.

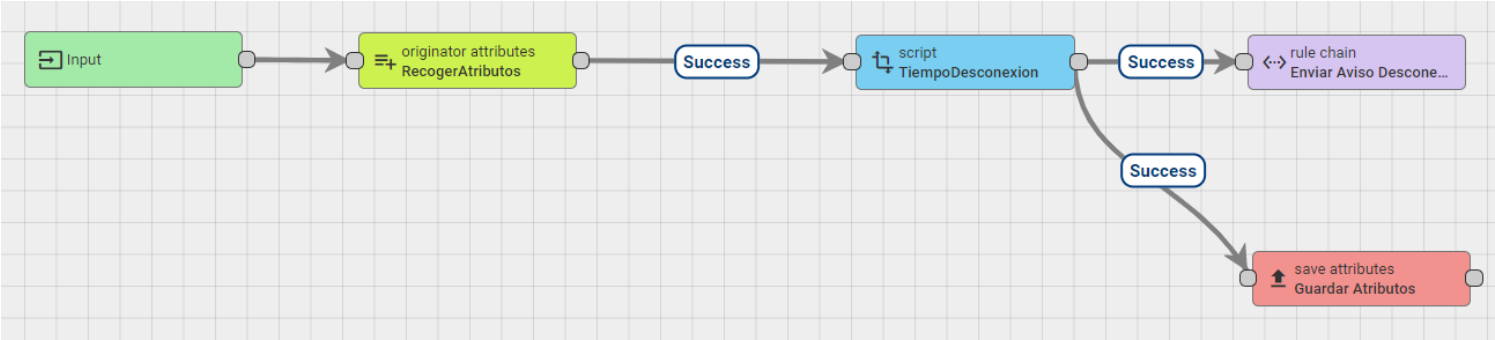


Ilustración 32: Control Tiempo Conexión. (Elaboración propia)

Para realizar esta funcionalidad la regla recibe un mensaje cada segundo, tal y como se describe en la regla principal. Este generador reflejado en la ilustración 33 envía un mensaje cada segundo, con el contenido vacío.

Ilustración 33: Generador. (Elaboración propia)

A continuación, el mensaje recibido por el generador se envía a un nodo del tipo 'originator attributes' donde se le añadirán los atributos 'T_MUESTREO' y 'T_DESCONEXION'.

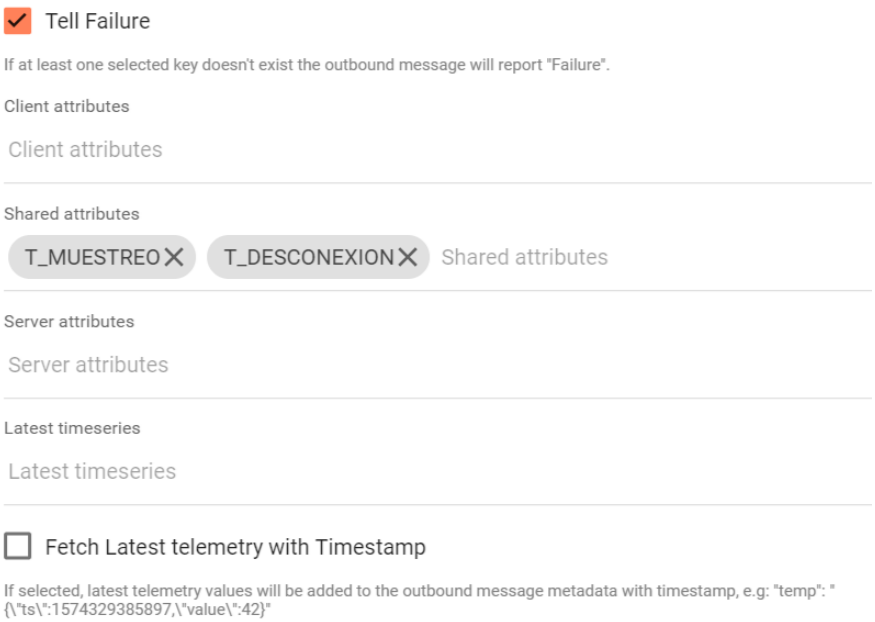


Ilustración 34: Originador Atributos. (Elaboración propia)

En la ilustración 35 se observa cómo se aumenta en 1 el valor del atributo ‘T_DESCONEXION’, cada que recibe un mensaje.

```
var desActual = parseInt(metadata.shared_T_DESCONEXION);
var msg = {
    T_DESCONEXION: desActual +1,
}

var msgType = "POST_ATTRIBUTES_REQUEST"
return {msg: msg, metadata: metadata, msgType: msgType};
```

Ilustración 35: Script 'tiempoDesconexion'. (Elaboración propia)

Regla 6: Enviar Aviso Desconexión

La funcionalidad de esta regla es avisar al cliente en caso de que la planta permanezca desconectada más de un tiempo parametrizable por el propio cliente, además debe indicar que el dispositivo tiene que activar el riego manual. Para realizar estas funciones se puede observar en la figura 23 cuales han sido los nodos necesarios. Esta regla recibe los mensajes por parte de la regla 5 'Control Tiempo de Conexión', cada vez que recibe un mensaje comprueba si el valor del atributo 'T_DESCONEXION' es superior al valor del atributo 'T_DESCONEXION_MAX', en cuyo caso envía un mensaje al bedel de la escuela.

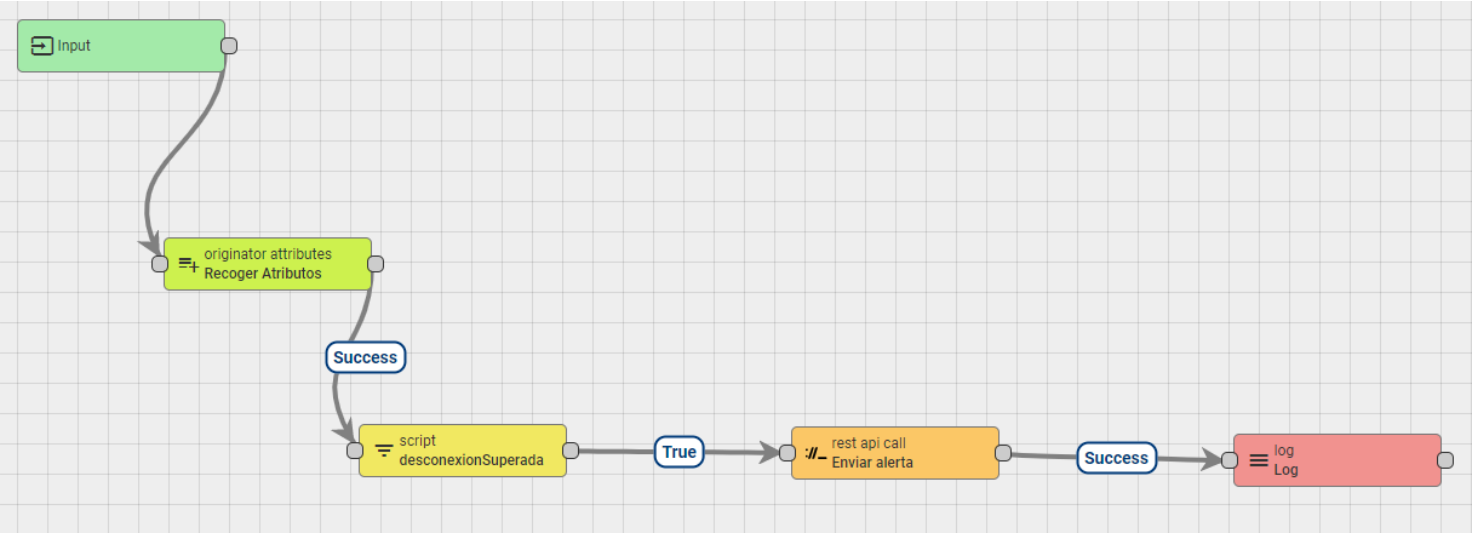


Ilustración 36: Enviar Aviso. (Elaboración propia)

Para realizar su función lo primero que necesita esta regla es recoger los valores de los atributos 'T_DESCONEXION' y 'T_DESCONEXION_MAX' para ello cuenta con un nodo del tipo 'originator attributes' que se puede ver en la ilustración 37.

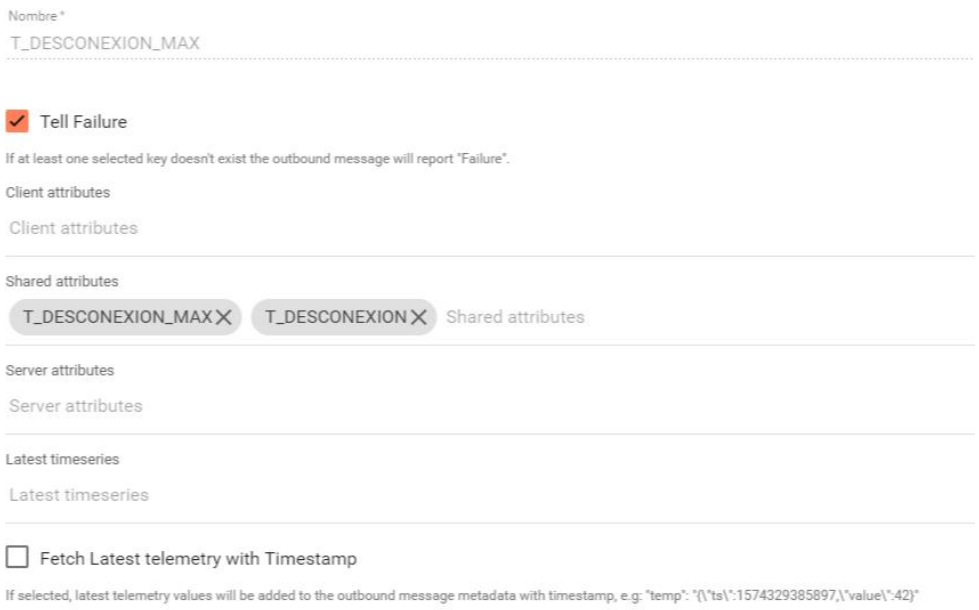


Ilustración 37: Originador Atributos. (Elaboración propia)

A continuación, el mensaje saliente de este nodo, el cual solo contiene los atributos recogidos, se envía a un nodo de tipo script que se puede ver en la ilustración 38. Este nodo '*desconexionSuperada* se encarga de comprobar si el tiempo de desconexión es mayor que el establecido por el usuario, en caso afirmativo él se ejecutara el nodo 'Enviar Alerta'.

```
1  var descActual = parseInt(metadata.shared_T_DESCONEXION) ;
2  var descMax = parseInt(metadata.shared_T_DESCONEXION_MAX);
3  return descActual === descMax;
```

Ilustración 38: Script *desconexionSuperada*. (Elaboración propia)

En la ilustración 39 se observa la llamada que se realiza al API de telegram, se puede ver que es una operación de tipo POST en la que se indica el identificador del chat al que queremos enviar el mensaje y otro parámetro llamado '*text*' donde se envía el contenido del mensaje. En este caso el mensaje que se envía es 'Conexión Perdida', para indicarle al usuario que se ha superado el tiempo máximo de desconexión.

Enviar alerta

Endpoint URL pattern *

https://api.telegram.org/bot1127642305:AAHC379b4BnkVODV3Rc2mi0R3mHcKAPEW-Q/sendMessage?chat_id=1128356456&text=Conexion Perdida

HTTP URL address pattern, use ${metaKeyName} to substitute variables from metadata

Request method

POST

☐ Use simple client HTTP factory

Ilustración 39: Petición API. (Elaboración propia)

Como se observa en las imágenes de cada una de las reglas definidas se han añadido una serie de logs que permitirán evaluar todas estas reglas, además de llevar un mejor control, por parte del desarrollador de las reglas, de las acciones que se realizan.

2.2. Base de datos

Como se ha comentado anteriormente la base de datos utilizada en el proyecto es Postgres. Hay dos elementos que harán uso de esta base datos:

- La plataforma IoT, puesto que no es de desarrollo propio el modelado de datos es inmutable, no obstante este modelado si puede consultarse accediendo a la propia base de datos. En esta base de datos se persisten todos los componentes que se crean en la plataforma cliente, dispositivo, reglas, telemetrías, etc
- Servicio API REST desarrollado, el cual permite persistir en esta base de datos toda la información recogida por el dispositivo IoT, además de permitir el registro de los clientes en el sistema.

Por otro lado, tenemos la base de datos implementada para el servicio API REST, como se describe en la sección dedicada a estos servicios, un servicio API REST está orientado a recursos. Estos recursos y la relación entre ellos modelan la base de datos, los recursos de este servicio web son:

- **Client**: Es el usuario del sistema. Esta entidad representa el usuario que va hacer uso del sistema. Los datos que lo componen son el identificador de la plataforma thingsboard y una descripción. Estos datos son mínimos, no obstante al solicitar el id de la plataforma IoT podemos obtener un gran número de datos de este usuario extrayéndolos de la plataforma IoT. Es importante destacar que para dar de alto un usuario en el sistema es necesario haberlo dado de alta en la plataforma IoT previamente. Para el caso concreto del macetero A-POT utilizado para la evaluación de este proyecto, se creó como cliente el estudiante que se hace cargo del dispositivo

- **Sensor**: Dispositivos con los que recogemos la información y que posteriormente enviamos al thingsboard. Para añadir un sensor al sistema es necesario que el nombre de este sea idéntico al que tiene en ThingsBoard, es decir, el nombre con el que se envíen los datos a la plataforma debe ser igual al nombre de los sensores en el servicio. En el caso del dispositivo implementado para este proyecto, se han dado de alta los sensores que recogen la información del macetero, "agua", "humedad_ambiental", "luz_ambiental", "Co2", "sensor_peso" y "temperatura_ambiental"

- **SensorType**: Se trata de un recurso precargado, catálogo, que se debe asignar a un sensor a la hora de crearlo en el sistema. Este catálogo puede ser modificado por el administrador del sistema. Inicialmente este catálogo se ha cargado con los valores necesarios para el macetero A-POT "Detector de Agua", "Sensor Peso", "Humedad Superficie", "Temperatura Superficie", "Humedad Ambiental", "Luminosidad", "Temperatura Ambiental", "Co2".

- **Sensor_Client**: Permite que el servicio tenga coherencia relacionando el sensor con el cliente correspondiente, esta relación la realiza el sistema de forma automática. Al registrar un nuevo cliente en el sistema y asignarle los sensores necesarios, el servicio REST se encarga de crear esta relación en base de datos.

- **Event**: Este recurso permite al sistema lleva un control de todas los eventos que se producen sobre el dispositivo. En el caso del macetero A-POT, cuando la plataforma IoT realiza alguna acción sobre el macetero envía una petición al servicio REST para que este persista en base de datos dicha acción.

- **Action**: Este recurso se asemeja bastante al ‘SensorType’ ya que se trata de un catálogo precargado. Dicho catálogo proporciona todas las acciones que se pueden realizar sobre el sistema. Inicialmente este catálogo se ha cargado con los valores necesarios para el dispositivo A-POT "abrir el riego", "cerrar el riego", "mover la tierra", "echar alimento" etc.
-

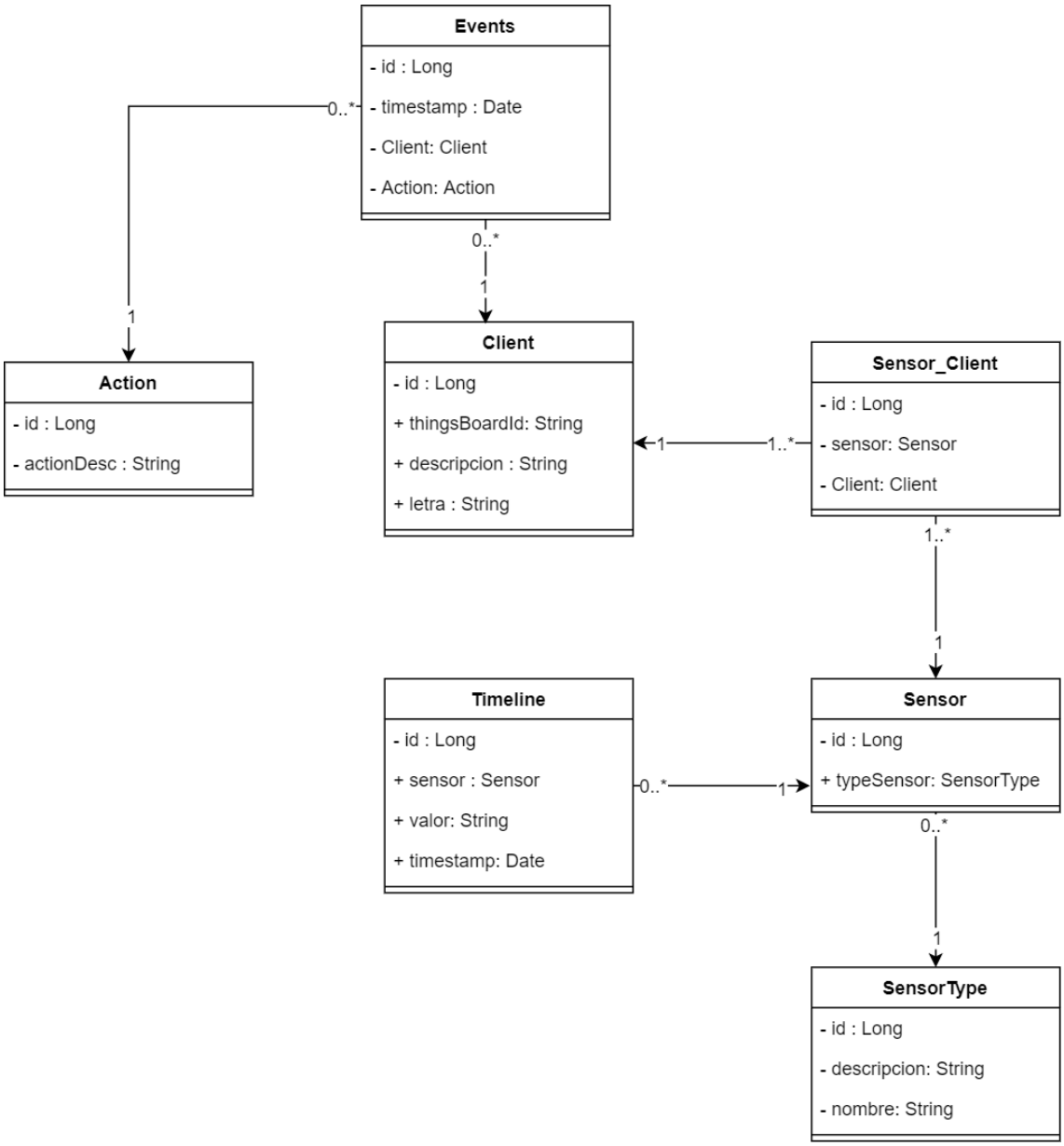


Ilustración 40: Diagrama de Clases. (Elaboración propia)

2.4. Instalación Realizada

Para el desarrollo de esta solución se ha requerido el uso de múltiples herramientas, todas ellas de código abierto y con una gran comunidad que aporta la mayoría de la información necesaria, para poder resolver cualquier problema.

En primer lugar, para varias herramientas de las que se describen a continuación es importante tener la instalación de OPENJDK 8. Para conseguir este JDK visita la página oficial de AdoptOpenJdk donde podrás descargarlo. A continuación, solo tienes que arrancar el archivo descargado y seguir las

instrucciones para la instalación. Asegúrate que se ha creado la variable de entorno JAVA_HOME. Para comprobar que la instalación se ha realizado correctamente abre una consola de comandos y ejecuta '*java -v*'.

Para el desarrollo del servicio API REST se ha utilizado IntelliJ IDEA un 'Integrated Development Enviroment'(IDE) para 'Java Virtual Machine' (JVM), diseñado para maximizar la productividad. IntelliJ se encarga de realizar las tareas repetitivas para permitirte centrarte en la lógica del código, haciendo la tarea de desarrollo más atractiva. Este IDE pude trabajar con los lenguajes de programación Java, Kotlin,Sacla y Groovy de forma nativa. Aunque esta herramienta se haya diseñado especialmente para trabajar con JVM, permite la instalación de plugins para poder trabajar con números framework, herramientas y tecnologías. Este IDE es multiplataforma lo que aporta una gran versatilidad y permite a cualquier persona utilizarlo ya sea con Linux, Windows o macOS.

IntelliJ IDEA tiene diferentes versiones Ultimate, Community y Edu, tanto la versión Community y Edu son gratuitas y a diferencia de estas la edición Utimate, es de pago, y provee de soporte al usuario.

Para la preparación de este IDE lo único necesario es realizar la descarga en su página oficial e instalarlo. No es necesario configurar ningún otro parámetro de configuración ni variable de entorno.

Para el desarrollo de las interfaces del sistema se ha utilizado la herramienta Visual Studio Code. Visual Studio Code es un editor de código fuente ligera pero potente, creado por Microsoft, que se ejecuta en su escritorio y está disponible para Windows, macOS y Linux. Esta herramienta está especialmente diseñada para trabajar con Angular, viene con soporte incorporado para JavaScript, TypeScript y Node.js y tiene un rico ecosistema de extensiones para otros lenguajes (como C ++, C #, Java, Python, PHP, Go) y tiempos de ejecución (como .NET y Unity).

Para la instalación de este IDE, al igual que IntelliJ lo único necesario es descargar el instalador y ejecutarlo, la instalación no llevará mucho ya que se trata de una herramienta muy ligera.

Para la creación del proyecto del servicio API REST se utiliza Spring Initializr, esta herramienta realmente no requiere de ninguna instalación ya que es un una aplicación web, sin embargo, es de gran utilidad a la hora de crear proyectos de SpringBoot. Esta herramienta permite crear un proyecto de SpringBoot de una forma

completamente automatizada, lo único necesario es establecer la configuración del proyecto y esta herramienta te lo crea para empezar a trabajar con él de inmediato.

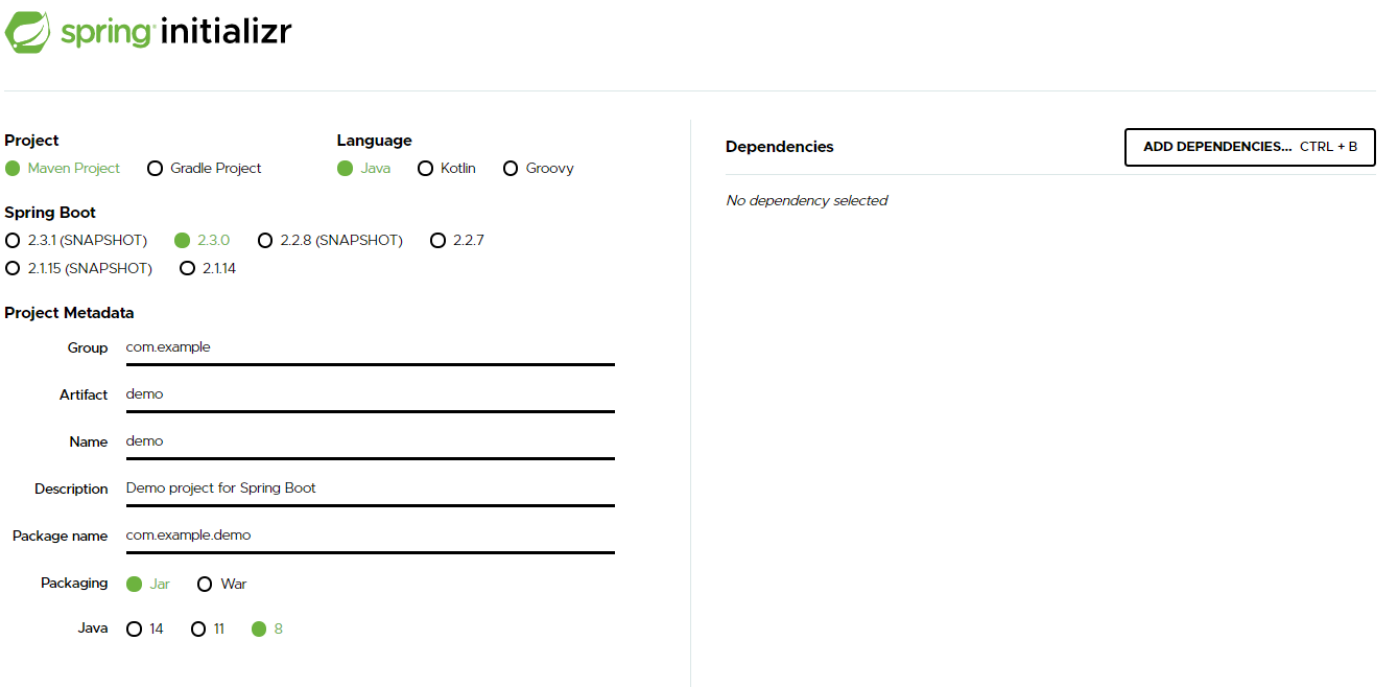


Ilustración 41: Formulario Spring Initializr

En la imagen 39 se observa la interfaz donde se puede establecer la configuración para el proyecto, como se puede apreciar las opciones de configuración son bastantes completas y además cuenta con una sección para añadir todas las dependencias que el usuario desee en la creación del proyecto.

Para trabajar con la plataforma ThingsBoard descrita anteriormente, lo primero que tenemos que hacer es la instalación. Puesto que el SO utilizado en este proyecto es Linux, solamente ha sido necesario ejecutar estos dos comandos, de las ilustraciones 42 y 43.

```
wget https://github.com/thingsboard/thingsboard/releases/download/v2.4.3/thingsboard-2.4.3.deb
```

Ilustración 42: Comando para descargar ThingsBoard

```
sudo dpkg -i thingsboard-2.4.3.deb
```

Ilustración 43: Comando para instalar ThingsBoard

De esta forma tendremos ThingsBoard listo para levantarlo y empezar a trabajar. Ahora bien, se ha incluido una base de datos Postgres para lo que ha sido necesario, realizar la instalación de Postgres y establecer una configuración para que la plataforma ThingsBoard utilice esta base de datos. Para instalar Postgres, al igual que con ThingsBoard, solo es necesario ejecutar el comando de la ilustración 44.

```
sudo apt-get install postgresql
```

Ilustración 44: Comando para instalar Postgres

De esta forma ya tendremos la base de datos lista para usar. No obstante, es necesario añadir cierta configuración referente a la base de datos en ThingsBoard. En la ilustración 45 se observan las modificaciones del archivo 'thingsboard.conf' añadiendo los siguientes parámetros, de esta forma se configura la base de datos, en la plataforma IoT.

```
export DATABASE_ENTITIES_TYPE=sql
export DATABASE_TS_TYPE=sql
export SPRING_JPA_DATABASE_PLATFORM=org.hibernate.dialect.PostgreSQLDialect
export SPRING_DRIVER_CLASS_NAME=org.postgresql.Driver
export SPRING_DATASOURCE_URL=jdbc:postgresql://localhost:5432/thingsboard
export SPRING_DATASOURCE_USERNAME=postgres
export SPRING_DATASOURCE_PASSWORD=postgrespw
```

Ilustración 45: Configuración Base de Datos

Por otro lado, se ha modificado la confirmación del access token para que este tenga un tiempo de expiración mayor. Para ello se ha modificado el parámetro '*export JWT_TOKEN_EXPIRATION_TIME*' del archivo thingsboard.yaml fijando el tiempo de expiración en un año, de esta forma se evita tener que estar generando tokens cada poco tiempo.

2.3. Casos de uso

Este sistema está pensado para alojar diferentes dispositivos IoT y almacenar sus valores de forma automática, sin embargo, hay ciertas acciones que requieren la interacción de personas. En este proyecto se comprenden dos tipos de personas el administrador del sistema y el usuario.

El **administrador** se encargará de interactuar con la base de datos del servicio para la creación, eliminación o modificación de usuarios. Si fuese necesario el administrador podría acceder directamente a la base de datos para realizar acciones sobre esta.

Por otro lado, el **usuario** también podrá crear un cliente nuevo en los sistemas y además se encarga de las creaciones necesarias en la plataforma IoT. Estas acciones se ven en la ilustración 46, el usuario deberá en primer lugar crear un cliente en la plataforma IoT, en segundo lugar deberá crear un nuevo dispositivo en la plataforma IoT al que enviará los datos recogidos por los sensores del dispositivo físico. Una vez se ha creado el cliente y el dispositivo está recibiendo datos de los sensores, el usuario crea un nuevo cliente en el sistema y le añade todos los sensores que tenga el dispositivo creado. Por último, el usuario podrá definir cualquier número de reglas en Thingsboard para que realice la lógica necesaria. En la sección 'Plataforma IoT' se describen cuáles han sido las reglas implementadas para el dispositivo concreto usado en este proyecto.

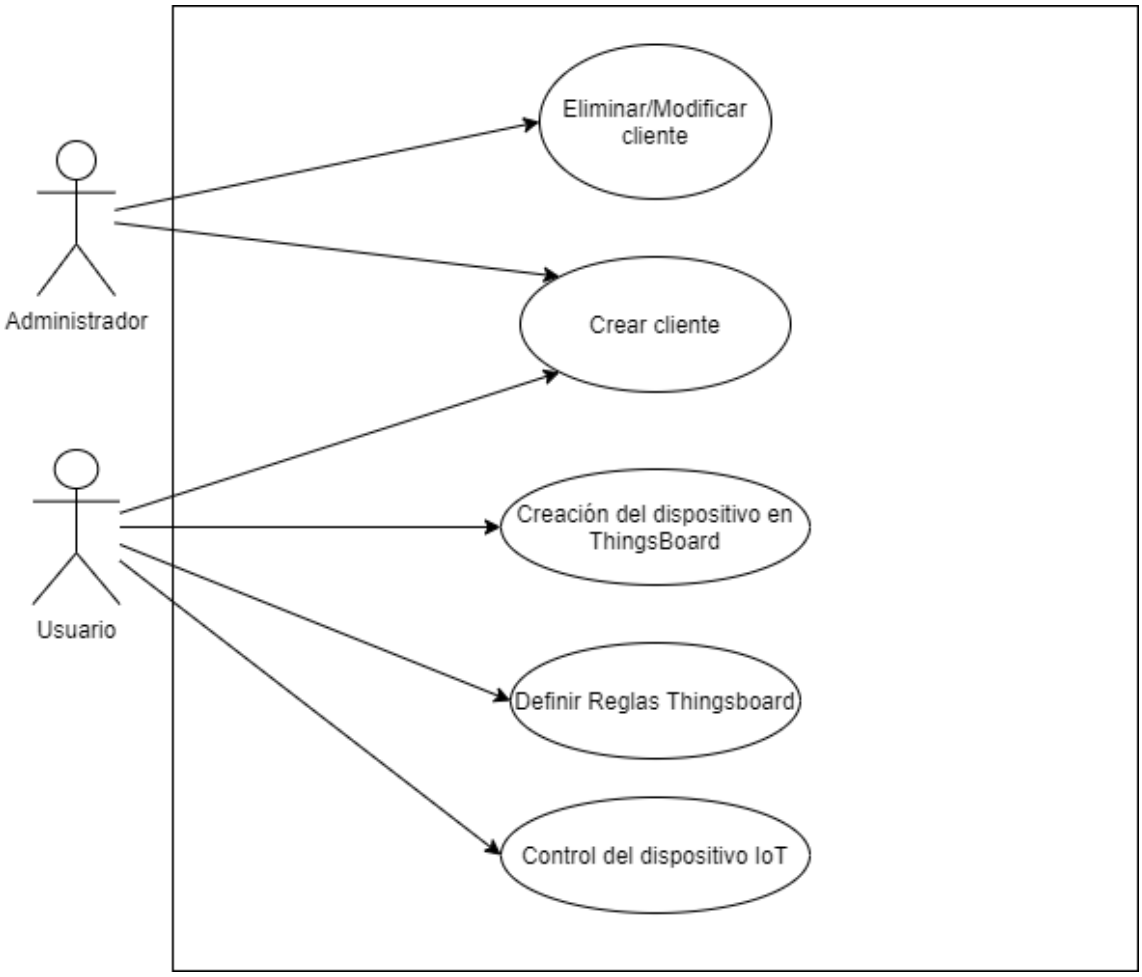


Ilustración 46: Diagrama de Casos de uso. (Elaboración propia)

3. Evaluación de la plataforma

En las secciones anteriores se han explicado los elementos que componen esta solución final. Estos elementos son; una interfaz gráfica para dar de alta los nuevos clientes en la plataforma, donde además se le asignaran los sensores de los que va a recibir información. Una segunda interfaz gráfica, la interfaz de cliente, donde el usuario podrá ver el estado de su planta y los datos tomados por los sensores. Un servicio API REST que permite persistir en una base de datos propia los datos que son enviados a ThingsBoard, además de implementar las funcionalidades para las dos interfaces gráficas. Por último la plataforma IoT, ThingsBoard, en la cual se ha implementado una serie de cadena de reglas, para realizar la lógica necesaria del cuidado de la planta.

En esta sección se evalúa el sistema mediante un ejemplo real, de un dispositivo IoT que envía información a la plataforma Thingsboard.

3.1. Evaluación del sistema con un único cliente IoT

Para esta evaluación el dispositivo IoT utilizado es una maceta 'inteligente', es decir, un macetero capaz de leer los valores recogidos por los sensores de la planta y enviar la información a la plataforma IoT, para posteriormente recuperar información de la misma.

El propósito de este sistema es recoger la información de las variables que pueden afectar al desarrollo de la planta como el sol, el agua, la contaminación, la temperatura, etcétera. Una vez se recoge toda la información se envía a la plataforma ThingsBoard donde se realizará la lógica necesaria para el cuidado de la planta y modificará los valores necesarios para que posteriormente el dispositivo sea capaz de actuar en consecuencia a la lógica realizada. Seguidamente el dispositivo leerá la información de estos actuadores, realizará las acciones necesarias y por último pasará a un estado de reposo, para volver a iniciar el ciclo una vez acabe este estado.

Para entender mejor como sería el funcionamiento de este dispositivo se ha realizado un diagrama de flujo con la lógica que sigue el micro controlador.

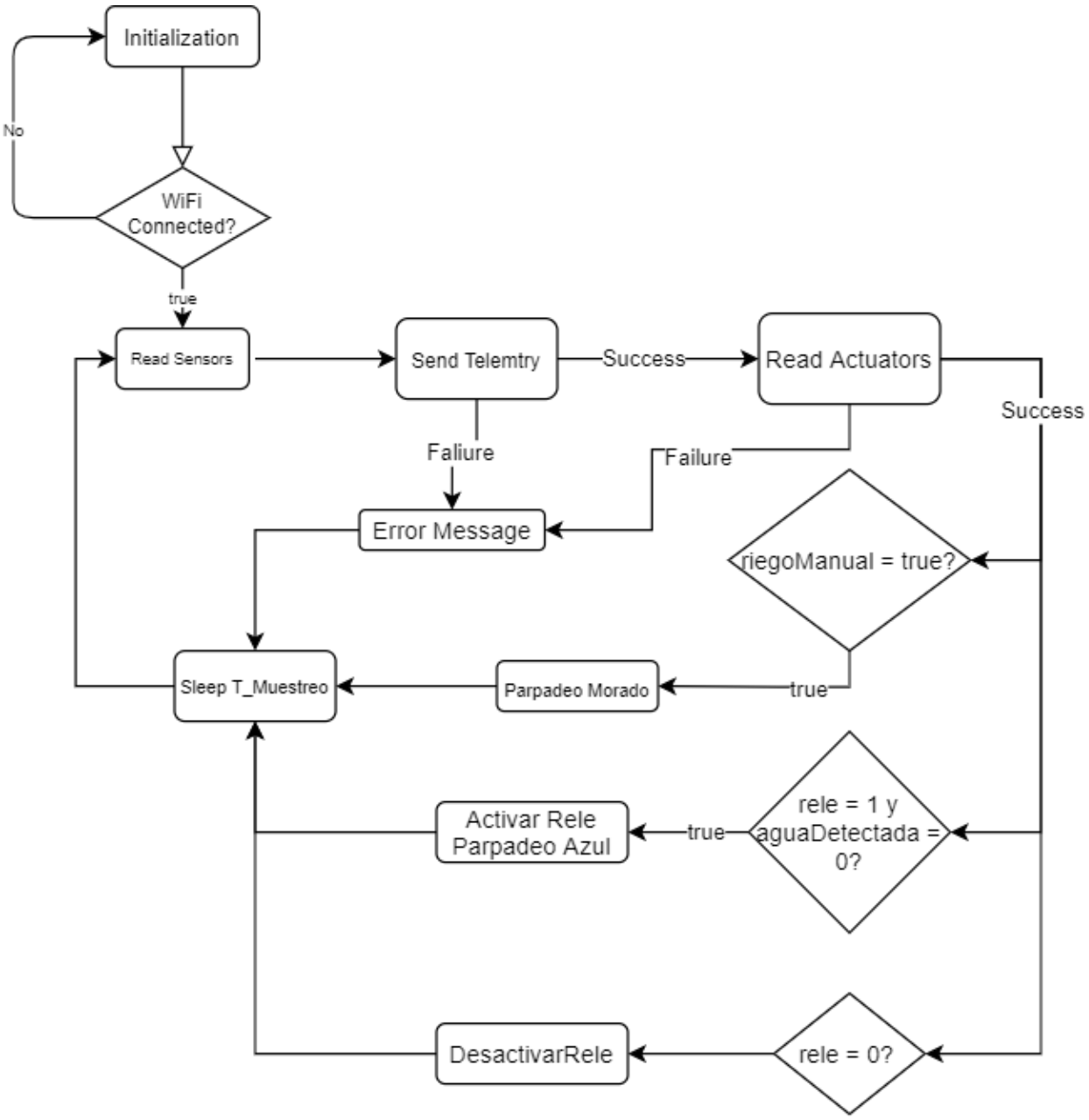


Ilustración 47: Diagrama de flujo ESP32. (Elaboración propia)

Herramientas Software

Antes de entrar en los detalles del funcionamiento de este dispositivo, se va introducir la tecnología utilizada para la implementación del software. Arduino, es una plataforma electrónica de código abierto, basada en hardware y software fácil de usar (*Arduino - Introduction*, n.d.). Esta plataforma permite la creación de dispositivos de muchos tipos facilitando la labor de trabajar con el micro-controlador. Arduino presenta una serie de características que lo convierten en una buena herramienta para el desarrollo del software de este dispositivo:

- **Entorno de programación:** Arduino cuenta con su propio Integrated Development Environment, o Arduino Software (IDE), fácil de usar para principiantes y lo suficientemente potente para que los usuarios expertos puedan sacarle rendimiento. Esta herramienta cuenta con un editor de texto para escribir código, un área de mensajes, una consola de texto, una barra de herramientas con botones para funciones comunes y una serie de menús.

- **Multiplataforma**: Arduino Software (IDE) puede ejecutarse en Windows, Macintosh OS X, y Linux.

- **Software de código abierto y extensible:** El software de Arduino se publica como herramientas de código abierto, disponibles para su extensión por programadores experimentados. Este software es ampliable mediante el uso de librerías C++.

Vistas algunas de las características que hacen al IDE de Arduino una buena herramienta para la implementación de software en un micro-controlador, se va a entrar en el detalle de cada una de las operaciones que realiza este macetero. En la ilustración 47 se representan cada una de las funciones que realiza el micro-controlador.

Al iniciar un proyecto con Arduino por defecto aparecen las funciones de '*setUp*()' y '*loop()*'. La función '*setUp()*', es la primera función que se ejecuta y se ejecutará una única vez, posteriormente la función '*loop()*' se ejecuta una y otra vez hasta que el dispositivo se apague.

```
void setup() {
  Serial.begin(115200);
  pinMode(humedadPinINT, INPUT);
  pinMode(humedadPinEXT, INPUT);
  pinMode(pesoPin, INPUT);
  pinMode(aguaPin, INPUT);
  pinMode(relePin, OUTPUT);

  pinMode(ledRojo, OUTPUT);
  pinMode(ledVerde, OUTPUT);
  pinMode(ledAzul, OUTPUT);


  Wire.begin(21, 22);
  als.begin();

  if (sht.init()) {
    Serial.print("init(): success\n");
  } else {
    Serial.print("init(): failed\n");
  }

  sht.setAccuracy(SHTSensor::SHT_ACCURACY_MEDIUM);

  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting to WiFi..");
    led_wifi_connecting();
  }

  Serial.println("Connected to the WiFi network");
  led_wifi_ok();
}
```

Ilustración 48: Función '*setUp()*'. (Elaboración propia)[10]

En este caso particular, la función '*setUp()*' se ha utilizado para definir cada uno de los pines donde están conectados los sensores, para inicializar la librerías necesarias y para establecer la conexión de red, en la ilustración 48 se puede observar el código implementado para la función '*setUp*()'.

Para poder llevar a cabo el establecimiento de la conexión lo primero que se debe hacer es incluir la librería 'Wifi.h' al proyecto. Arduino software permite añadir librerías

[10] Función setup. https://github.com/FJavier95/A-POT/blob/b0e67926ef60a893ccee24f8bcfe640939fd3bc2/src-arduino/esp32-SBC/TFG_v1/TFG_v1.ino#L35

de una manera intuitiva, en la ilustración 49 se refleja cómo se puede añadir una librería en un proyecto Arduino.

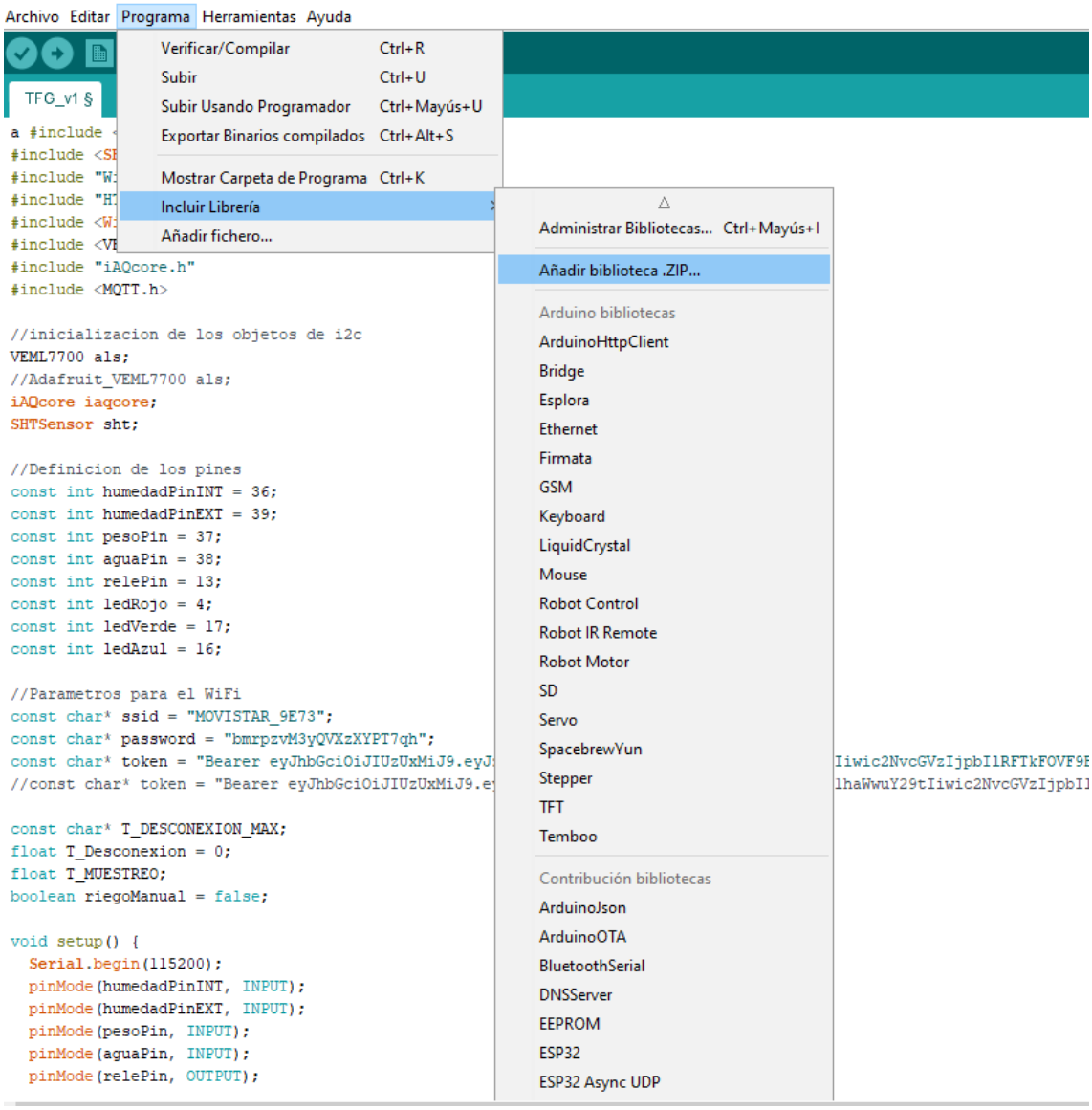


Ilustración 49: Adición librería

Una vez se ha añadido la librería debe aparecer en el código de forma automática la expresión '*#include*' con el nombre de la librería, de esta forma ya se pueden explotar las funcionalidades de dicha librería. En este caso '*Wifi.h'* cuenta con la función '*begin(SSID, Password)*', la cual permite establecer una conexión de red pasándole el identificador de la red WIFI junto con la contraseña. Si en la ejecucion de esta funcion no se ha producido ningun error, devolverá el estado 'WL_CONNECTED', lo que significará que la conexión se ha establecido correctamente.

Cuando se termina de ejecutar la funcion de '*setUp()*', se empieza a ejecutar la funcion '*loop()*' de la ilustración 50.

```
void loop() {

  // allocate the memory for the document
  DynamicJsonBuffer jsonBuffer;
  // create an empty array
  JsonObject& valores = jsonBuffer.createObject();

  int rele, peso, humedad_int, humedad_ext, co2, agua;
  float luminosidad, temperatura, humedad;
  // rele = comprobarSistema();
  //Se recogen los valores
  peso = calcularPeso();
  humedad_int = calcularHumedadSueloINT();
  humedad = read_humidity();
  co2 = read_co2();
  luminosidad = read_lux();
  temperatura = read_temperature();
  agua = comprobarAgua();

  valores["humedad_ambiental"] = humedad;
  valores["luz_ambiental"] = luminosidad;
  valores["humedad_suelo"] = humedad_int;
  valores["sensor_peso"] = peso;
  valores["agua"] = agua;
  valores["Co2"] = co2;
  valores["temperatura_ambiental"] = temperatura;

  valores.prettyPrintTo(Serial);
  sendData(valores);
  delay(10000);
  readActuator();
  delay(50000);
}
```

Ilustración 50: Función '*loop()*. (Elaboración propia)[11]

El cometido de esta función será recuperar los valores leídos por los sensores y enviárselos a la plataforma IoT, para que esta realice la lógica necesaria y posteriormente recoger las telemetrías de dicha plataforma. Dentro de esta función se realizan llamadas a otros métodos, los cuales se explican a lo largo de esta subsección.

Una vez que se ha establecido la conexión con internet en fase de '*setUp*' y el dispositivo ya está listo para enviar datos a la plataforma IoT, lo primero que se realiza

[11] https://github.com/FJavier95/A-POT/blob/b0e67926ef60a893ccee24f8bcfe640939fd3bc2/src-arduino/esp32-SBC/TFG_v1/TFG_v1.ino#L249

es la recuperación de los datos tomados por sensores, tal y como se refleja en la ilustración 47. Para ello es necesario diferenciar dos tipos de sensores, aquellos que trabajan con librerías externas y los que utilizan funciones propias de Arduino.

Los sensores que necesitan el uso de librerías externas son aquello que tiene un protocolo de comunicación I2C, mientras que los otros son de tipo analógico y Arduino ya incorpora funciones para este tipo de comunicación. Los sensores de tipo I2C requieren el uso de librerías debido al proceso que se realiza para establecer la conexión, en dichas librerías este proceso ya viene implementado facilitando la tarea de comunicación entre sensor y micro-controlador.

En este caso concreto los sensores de tipo I2C, recogen datos de cuatro variables humedad ambiental, temperatura ambiental, luminosidad ambiental y calidad del aire. La humedad y la temperatura ambiental son recogidas mediante el uso de la librería '*SHTSensor*'.

```
//Adicion de la libreria al proyecto
#include <SHTSensor.h>

//Inicializacion de libreria, para poder trabjar con ella
SHTSensor sht;

//Lectura de la temperatura
float read_temperature() {
  float temperature;
  if (sht.readSample()) {
    temperature = sht.getTemperature();
  } else {
    temperature = 0;
  }
  return temperature;
}

//Lectura de la humedad
float read_humidity() {
  float humidity;
  if (sht.readSample()) {
    humidity = sht.getHumidity();
  } else {
    humidity = 0;
  }
  return humidity;
}
```

Ilustración 51: Lectura Temperatura y Humedad Ambiental. (Elaboración propia)[12]

Como se puede ver en la ilustración 51 esta librería provee dos funciones una para recoger la humedad ambiental '*getHumidity()*' y otra función diferente para

[12] https://github.com/FJavier95/A-POT/blob/b0e67926ef60a893ccee24f8bcfe640939fd3bc2/src-arduino/esp32-SBC/TFG_v1/TFG_v1.ino#L145

recoger la temperatura '*getTemperature()*'. Además de estas funciones para recoger los valores, la librería cuenta con otra función que permite saber si hay conexión con el sensor o no. Esta función, 'readSample*()*', devolverá un valor '*true*' en caso de que el sensor esté conectado, y entonces se ejecutara la lectura de los valores.

Otro de los parámetros que miden estos sensores de tipo I2C es la luminosidad, recogida mediante el uso de la librería '*VEML7700*'. En la ilustración 52 se aprecia cómo esta librería provee la función '*getALSLux()*', la cual permite recoger el nivel de luminosidad. Esta función recibe como parámetro la variable donde se debe guardar el valor leído.

```
//Adicion de la libreria al proyecto
#include <VEML7700.h>

//Inicializacion de libreria, para poder trabjar con ella
VEML7700 als;

//Lectura de la luminosidad
float read_lux() {
  float luxes;
  als.getALSLux(luxes);
  return luxes;
}
```

Ilustración 52: Lectura Luminosidad. (Elaboración propia)[13]

Por último, en este grupo de sensores también se realiza la lectura de la calidad del aire y para ello se hace uso de la librería '*iAQcore*'. Esta librería cuenta con una función '*read()*', a la cual se tienen que pasar 4 parámetros diferentes reflejados en la ilustración 53. Como se puede apreciar en la ilustración 53, cualquier valor por encima de 2000 ppm (Particular Por Millon), significa que el sensor no ha podido tomar valores correctamente y por tanto se establece el valor 449, como valor erróneo ya que este sensor mide valores entre 450 y 2000 ppm.

[13] https://github.com/FJavier95/A-POT/blob/d6d9cf88ab7b736e49d4421355cd5aacf62b330a/src-arduino/esp32-SBC/TFG_v1/TFG_v1.ino#L155

```
int read_co2() {
  uint16_t eco2;
  uint16_t stat;
  uint32_t resist;
  uint16_t etvoc;
  iaqcore.read(&eco2, &stat, &resist, &etvoc);
  Serial.print(iaqcore.getCO2PredictionPPM());
   Serial.println(" ppm");
  if (eco2 > 2000) {
    eco2 = 449;
  }
  return eco2;
}
```

Ilustración 53: Lectura de la Calidad del Aire. (Elaboración propia)[14]

Como se ha descrito anteriormente, además de este grupo de sensores de tipo I2C, el dispositivo IoT cuenta con otros sensores de tipo analógico. Estos sensores analógicos se van a encarga de medir las variables de humedad de la tierra y el peso además de contar con otro sensor que es capaz de detectar agua. En la ilustración 54 se pueden ver las funciones implementadas para la recuperación de los datos de estos sensores.

[14] https://github.com/FJavier95/A-POT/blob/d6d9cf88ab7b736e49d4421355cd5aacf62b330a/src-arduino/esp32-SBC/TFG_v1/TFG_v1.ino#L161

```
int calcularPeso() {
  float valor_peso = 0;
  valor_peso = analogRead(pesoPin);
  return valor_peso;
}

int calcularHumedadSueloINT() {
int valor_humedad = 0;
  valor_humedad = analogRead(humedadPinINT);
  Serial.println(valor_humedad);
   int valor = 4095 - valor_humedad;
  valor_humedad = (valor * 100)/3071;
  return valor_humedad;
}

int comprobarAgua() {
  int valor_agua = 0;
  valor_agua = analogRead(aguaPin);
  if (valor_agua > 3500){ valor_agua = 0;}else{ valor_agua = 1;}
  return valor_agua;
}
```

Ilustración 54: Lectura de sensores analógicos. (Elaboración propia)[15]

Como se aprecia en la ilustración 54, la forma de recuperar los valores de estos sensores es, mediante el uso de la función '*analogRead()*'. Dicha función espera como parámetro el nuero del pin en el cual tiene que realizar la lectura. En primer lugar la función que recoge el valor del peso, se declara la variable donde queremos almacenar ese valor y la igualamos a la función '*analogRead*()' pasándole el pin del sensor de peso como parámetro .

A continuación de la función de peso, en la ilustración 54, se encuentra la función para recoger el nivel de humedad de la tierra. El funcionamiento es análogo a la función de peso, sin embargo, en este caso se ha ajustado el valor para que este sea devuelto en tanto por ciento. Por último dentro de este grupo de sensores analógicos está el detector de agua, puesto que de este sensor solo nos interesa saber si detecta agua o no, se ha ajustado el valor que devuelve para que sea de '0' o de '1', respectivamente.

Siguiendo el flujo descrito en la ilustración 47, una vez se han recogido todos los valores de los sensores, se realiza el envío de los datos recopilados a la plataforma

[15] https://github.com/FJavier95/A-POT/blob/c6fc72011a2952780b96e0b2ef6c4bd924d44669/src-arduino/esp32-SBC/TFG_v1/TFG_v1.ino#L105

ThingsBoard. Para ello es necesario utilizar la API REST proporcionada por ThingsBoard, realizando una petición POST en la cual se envíe como *body*, un JSON con el nombre que se le quiera dar al sensor y el valor del mismo, pudiendo ser múltiples duplas de *<nombre, valor>*. Para realizar este envío es necesario 'atacar' al endpoint '*http://138.4.92.46:8080/api/vi/{device_token}/telemetry*', el cual necesita el token del dispositivo como parámetro.

En la ilustración 55 se observa la función desarrollada para él envió de estos datos, mediante una petición http. Dicha función espera recibir como parámetro un JSON con los valores que se desean enviar a la plataforma. Para realizar esta petición a la API de ThingsBoard es necesario incluir en el proyecto la librería '*HTTPClient',* de la misma manera que se observa en la ilustración 49. Una vez se ha importado la librería ya se puede hacer uso de ella, la primera función que se utiliza es la función '*begin()*' la cual espera recibir como parámetro el endpoint al que deberá realizar la petición. Como se puede apreciar en la ilustración 55 el endpoint ya incluye el token asociado al dispositivo y por el cual, ThingsBoard sabrá a que dispositivo asociar los valores. A continuación se incluyen en la petición las cabeceras necesarias, en este caso concreto se indica el tipo de interfaz para la comunicación (JSON) y se incluye el token de la aplicación. Es importante destacar que este segundo token no tiene nada que ver con el token del dispositivo, este token se genera para la plataforma en general y cualquier petición que se quiera realizar debe incluir este token.

Por último, se hace uso de la función 'POST()', esta función espera recibir como parámetro el *body* de la petición para enviar los datos necesarios. Esta función será la encargada de realizar la petición a la API y recibir el código de respuesta, si este código es un 200 significa que la petición se ha realizado correctamente y por tanto se mostrará un mensaje que indique que los datos han sido enviados. En el caso contrario, en el que la petición fallase, se mostraría un mensaje de error.

```
int sendData(JsonObject& valores) {
  if (WiFi.status() == WL_CONNECTED) {
    HTTPClient http;
      char json_str[1000];
    valores.printTo(json_str, sizeof(json_str));
    http.begin("http://138.4.92.46:8080/api/v1/66WxSzusOoPemMXixHHV/telemetry");
    http.addHeader("Content-Type", "application/json");
    http.addHeader("X-Authorization", token);
    int httpResponseCode = http.POST(json_str); //Send the actual POST request


    if (httpResponseCode > 0) {
      String response = http.getString();
      Serial.println(httpResponseCode);
      Serial.println(response);
    } else {
      Serial.print("Error on sending POST Request: ");
      Serial.println(httpResponseCode);

    }
    http.end();
    return httpResponseCode;
  } else {
    Serial.println("Error in WiFi connection");
    return 0;
  }
}
```

Ilustración 55: Envío de los datos. (Elaboración propia)

Una vez los valores son enviados a la plataforma siguiendo la ilustración 47, la siguiente función que realiza el micro-controlador, es leer los valores de actuadores para ver si la plataforma IoT los ha modificado y así actuar en consecuencia.

Para recoger los valores de ThingsBoard se ha implementado otra función, la cual se puede observar en la ilustración 56. Al igual que para enviar los datos, también es necesario utilizar la API REST de la plataforma, por lo que la librería importada anteriormente servirá de forma análoga en este caso. No obstante, el endpoint al que se debe 'atacar' en esta petición es, '*http://138.4.92.46:8080/api/plugins/telemetry/DEVICE/{device_id}/values/timeseries'*. A diferencia de la función para enviar los datos, en esta petición se debe pasar el identificador del dispositivo en lugar del token. Además de ser un endpoint diferente, el tipo de petición que se realiza también es diferente, en este caso como queremos recoger los datos de la plataforma se debe realizar una petición de tipo 'GET'. Para ello la librería '*HTTPClient'* también provee una función para realizar esta petición, sin embargo esta función no espera recibir ningún parámetro.

```
void readActuator() {
  HTTPClient http;
  DynamicJsonBuffer jsonBuffer;

  http.begin("http://138.4.92.46:8080/api/plugins/telemetry/DEVICE/75a6f2f0-c13e-11ea-bd36-25db62324136/values/timeseries");
  http.addHeader("Content-Type", "application/json");
  http.addHeader("X-Authorization", token);

  StaticJsonBuffer<300> JSONBuffer; //Memory pool
  int httpCode = http.GET();

  if (httpCode > 0) {
    String payload = http.getString();
    JsonObject& root = jsonBuffer.parseObject(payload);
    String rele = root["rele"][0]["value"];
    String leds = root["leds"][0]["value"];
    String riegoManual = root["riegoManual"][0]["value"];
    agua = comprobarAgua();
    if(riegoManual == "true"){
      led_parpadeo_mezcla(4,16);
    }
     if(rele == "1" && agua == "0"){
      digitalWrite(relePin, HIGH);
      led_parpadeo(ledAzul);
    }else if(rele == "0"){
      digitalWrite(relePin, LOW);
    }
  } else {
      Serial.print("Error on sending POST Request: ");
      Serial.println(httpResponseCode);
      led_parpadeo_mezcla(4,16);
   }
  http.end();
}
```

Ilustración 56: Lectura de Actuadores. (Elaboración propia)

Como se aprecia en la ilustración 56, una vez realizada la petición para obtener las telemetrías es necesario extraer aquellas que nos interesan, en este caso la plataforma Thingsboard modifica el valor del relé en función de si es necesario o no activar el riego. Para poder extraer el valor la librería '*HTTPClient'* proporciona una función llamada '*getString()*', con esta función se recoge el contenido de la respuesta que otorga la API en formato JSON. Una vez se ha recuperado el contenido de la respuesta en formato JSON se extraen el valor del relé y el valor de los leds, si el valor del relé es igual a '1', debemos enviar una señal al relé para que se active y deje correr el agua.

Hardware

El hardware es el conjunto de dispositivos electrónicos con los que se forma un dispositivo, en este caso más concreto una maceta IoT.

Esta parte hardware del sistema está compuesta por una placa ESP32 SparkFun Thing, la cual cuenta con un micro-controlador que permite ejecutar el código descrito en la sub sección de 'Software' y además tiene una tarjeta de red que permite la conexión a internet. En la ilustración 57 se puede ver las especificaciones de dicha placa y donde se encuentran cada uno de los pines. A esta placa se conectan el resto de elemento que requiere el sistema y cada uno de los cuales se explican a continuación.

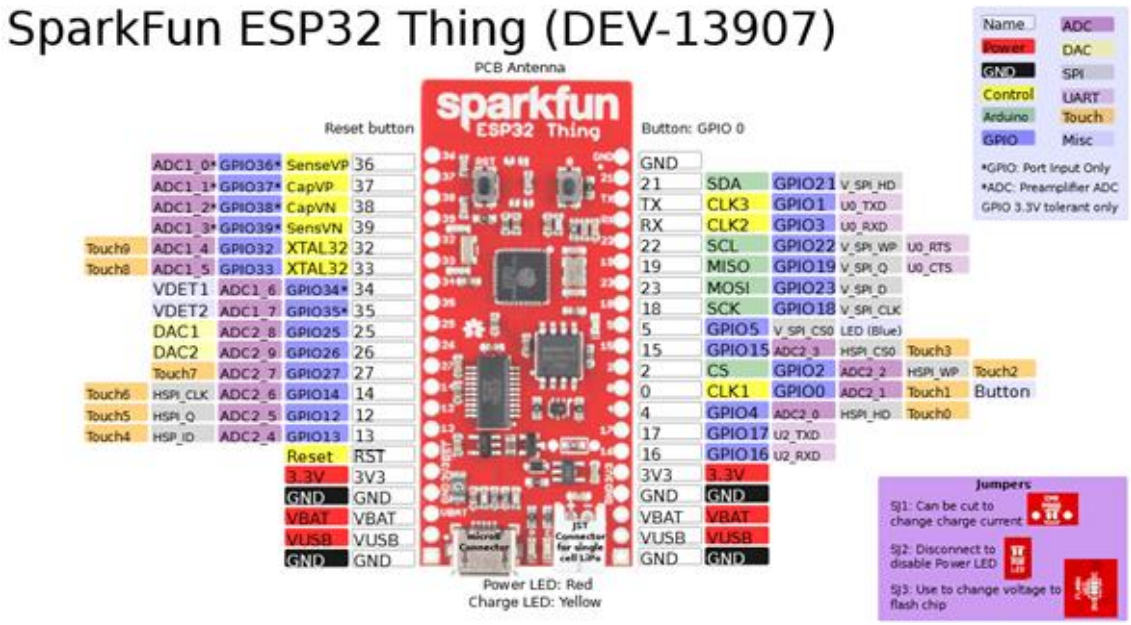


Ilustración 57: PinOut SparkFun32

Puesto que el sistema cuenta con una cantidad considerable de sensores se van a clasificar según los protocolos de comunicación, I2C o analógicos.

En primer lugar, se va a hablar de los sensores de tipo I2C. Esto tipos de sensores únicamente requieren el uso de dos pines, el SDA(Serial Data) y el SCL(Serial Clock). La ventaja de estos sensores es que no necesitan un pin cada uno para enviar los datos sino que conectando todos a los mismos pines son capaces de funcionar correctamente. Por tanto las conexiones que se realizan para estos sensores

son, el SDA al pin 21 y el SCL al pin 22, con esta conexión y con las funciones que se han descrito anteriormente en la sub sección de 'Software', se obtienen los valores tomados por los sensores.

A continuación se detallan cada uno de los sensores de tipo I2C utilizados para este proyecto.

- **Sensor de temperatura + humedad ambiental(SHT21):**

El SHT21 es un sensor estándar que mide temperatura y humedad relativa, con una precisión del ±0.3ºC y ±2HR respectivamente. Es importante añadir que el rango del funcionamiento esta entre -40+125ºC, una vez superado estos límites el sensor podría no funcionar. La salida proporcionada es de tipo digital y con una interfaz I2C.

En los sensores de tipo I2C se utiliza el pin SDA para transferir los datos del sensor al micro controlador de forma bidireccional, es decir, este pin también permite enviarle un comando al sensor desde el MCU para especificarlo cuando tiene que leer. Sin embargo, para que el MCU pueda enviar al SHT21 es necesario que el SerialClock esté a nivel alto. Este SerialClock(SCL) permite una comunicación síncrona entre micro controlador y sensor.

Pin	Name	Comment
1	SDA	Serial Data, bidirectional
2	VSS	Ground
5	VDD	Supply Voltage
6	SCL	Serial Clock, bidirectional
3,4	NC	Not Connected

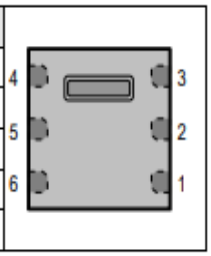


Ilustración 58: Sensor SHT

- **Sensor de Calidad del Aire(IAQ-CORE):**

El IAQ-CORE es capaz de medir los niveles de COV (compuestos orgánicos volátiles), lo que proporciona predicciones de CO2 y de TVOC (total de compuestos orgánicos volátiles) equivalentes. Para este elemento el rango de funcionamiento está entre -0+50ºC. Al igual que el componente anterior proporciona la salida por una interfaz 12C.

Como se ha explicado anteriormente en la descripción de la ESP32 Sparkfun Thing, todos los sensores con tipo de interfaz I2C se conectan de igual forma, por lo que la conexiones de este sensor son iguales que el SHT21.

Pin Number	Pin Name	Comment
1	NC	Not connected
2	SCL	I²C serial clock
3	GND	Ground
4	SDA	I²C serial data
5	NC	Not connected
6	VCC	+3.3V

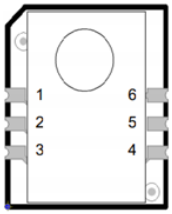


Ilustración 59: Sensor IAQ-Core

- **Sensor de Luminosidad(VEML7700-TT):**

El VEML7700-TT proporciona una gran precisión gracias a su fotodiodo sensible, lo que complementa con un ADC (conversor analógico digital) de 16 bits. Para este sensor aparte de tener en cuenta los límites de temperatura(-25+85ºC), es importante no sobrepasar la alimentación máxima que es de 3,6V, de superarla el sensor podría quemarse.

Las conexiones son análogas a los sensores descritos anteriormente.

Pinning
1: SCL
2: V_{DD}
3: GND
4: SDA

Ilustración 60: Sensor VEML7700

En segundo lugar, se encuentran los sensores de tipo analógico, cada uno de estos sensores es necesario conectarlo a un pin diferentes. Las conexiones de estos sensores se han realizado en siguientes pines; GPIO36 (pin36) para conectar el sensor de humedad que está colocado en el interior de la tierra; GPIO37 (pin 37) para el sensor de peso y por último el detector de agua en el pin GPIO38(pin38).

- **Sensor de peso(FC22):**

El FC22 es un sensor analógico que mide la fuerza ejercida sobre el botón plateado que tiene en la superficie. La masa máxima soportado por este sensor es de 4,54Kg y su rango de funcionamiento esta entre -40+80ºC.

Puesto que este sensor venía conjuntamente con el sensor de agua, el único pin al que ha hecho falta conectarlo es el GPIO37.

Ilustración 61: Sensor FC22

- **Sensor de agua(ELB145D2P):**

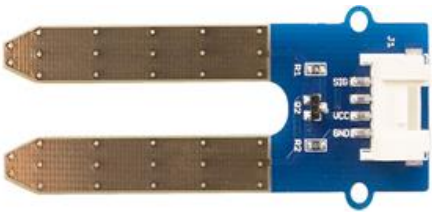
El ELB145D2P es un sensor de tipo analógico capaz de detectar si hay agua o no gracias a sus barras conductoras. Los sensores con este tipo transmisión solo necesitan de un canal, un pin, por el que enviar los datos que recogen, además de la alimentación y la tierra. El rango de correcto funcionamiento de este sensor se encuentra entre 10+30ºC

Pin number	Pin name	Comment
1	GND	Ground
2	VCC	3.3V
3	NC	Not connected
4	SIG	Data

Ilustración 62: Sensor ELB14D2P

- **Sensor de Humedad de la tierra(Kit Grove):**

Puesto que se trata de un sensor de tipo analógico el funcionamiento es análogo al anterior. No obstante no se ha de olvidar que este tipo de sensores no pueden conectarse todos al mismo pin, es decir, su canal de 'Data' ha de estar conectado en solitario al pin 36.



Pin number	Pin name	Comment
1	GND	Ground
2	VCC	3.3V
3	NC	Not connected
4	SIG	Data

Ilustración 63: Kit Grove

Por último, se encuentran los actuadores, estos elementos son imprescindibles ya que nos van a permitir realizar acciones sobre la planta, como activar/desactivar el riego.

- **Leds de alta intensidad:**

El sistema cuenta con 3 leds RGB de alta intensidad, montados sobre una pequeña placa, esto permite recibir feedback del estado de la planta, como por ejemplo si le falta agua o si tiene mucha agua. Para conectar estos leds se ha elegido el pin GPIO4 para el color rojo, el GPIO17 para el color verde y el GPIO16 para el azul. Esto permite que simplemente con enviar 3.3V a esos pines el led luzca del color correspondiente.

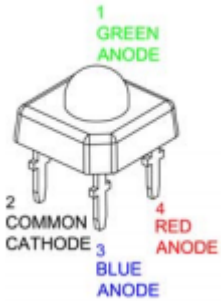


Ilustración 64: Led

- **Relé**

Además de estos leds, el sistema cuenta con un relé para poder conmutar la válvula que deja pasar el agua. De esta manera se ha podido implementar una lógica, que permite regular el riego según los valores tomados del thingsboard.

Ilustración 65: Relé

3.1.1. Puesta en práctica

En la introducción de esta sección se ha descrito como está formado y cuál es el comportamiento del dispositivo que se va a utilizar para la evaluación del sistema. En esta sub sección se evalúa la lógica que seguirán los datos enviados por el dispositivo a la plataforma IoT, para ello se comprueba que cada una de las cadenas de reglas sigue el comportamiento esperado.

La regla que recibe los datos entrantes del dispositivo es la 'Regla Principal', reflejada en la ilustración 13. Esta regla en primer lugar modifica el tipo del mensaje entrante, por el necesario para cada uno de los nodos que se ejecutan a continuación. Para cada una de las funciones que debe implementar esta regla se han añadido logs para verificar que los nodos ejecutan su tarea y devuelven el resultado esperada. Para el resto de cadenas de reglas que se ejecutan a través de esta 'Regla Principal', implementan sus propios logs que permitirán evaluar su funcionamiento.

En la ilustración 66 se puede ver una parte de la 'Regla Principal', con los dos script de la parte superior. El script 'T_DESCONEXION = 0' se encargara de enviar un mensaje a un nodo de tipo 'save attributes' para que se establezca el atributo de 'T_DESCONEXION' a 0 ya que se ha recibido un mensaje entrante del dispositivo. Análogo a este comportamiento el segundo script establecerá el valor de la telemetría 'riegoManual' en false, ya que la plataforma recibe datos desde el dispositivo.

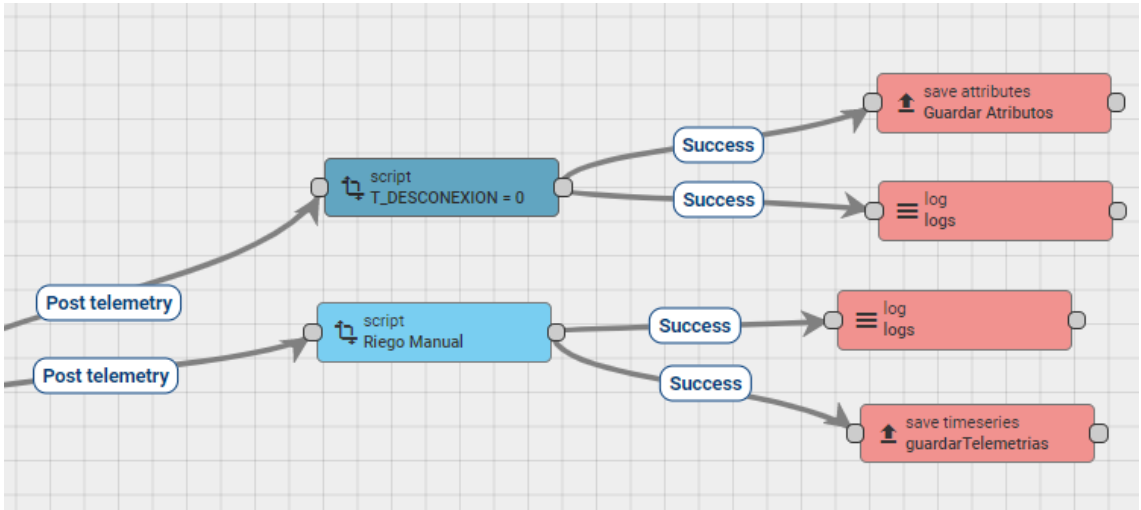


Ilustración 66: Parte Regla Principal

Para verificar que estos nodos se ejecutan correctamente en la ilustración 67 se observan los logs que muestran los mensajes resultantes de los mismos.

```
RIEGO MANUAL =>{"riegoManual":false}
METADATA =>{"deviceType":"ESP32","deviceName":"Maceta IoT","ts":"1600614612106"}
2020-09-20 17:10:12,355 [ForkJoinPool-3-worker-9] INFO  o.t.rule.engine.action.TbLogNode -

DETECCION DE AGUA (MENSAJE ENTRANTE)
MENSAJE =>{"humedad_ambiental":52.16,"luz_ambiental":0,"humedad_suelo":16,"sensor_peso":1131,
METADATA =>{"deviceType":"ESP32","deviceName":"Maceta IoT","ts":"1600614612106"}
2020-09-20 17:10:12,355 [ForkJoinPool-3-worker-30] INFO  o.t.rule.engine.action.TbLogNode -

NIVEL DE HUMEDAD DE LA TIERRA INFERIOR (MENSAJE ENTRANTE)
MENSAJE =>{"humedad_ambiental":52.16,"luz_ambiental":0,"humedad_suelo":16,"sensor_peso":1131,
METADATA =>{"deviceType":"ESP32","deviceName":"Maceta IoT","ts":"1600614612106"}
2020-09-20 17:10:12,356 [ForkJoinPool-3-worker-16] INFO  o.t.rule.engine.action.TbLogNode -

T_DESCONEXION =>{"T_DESCONEXION":0}
METADATA =>{"deviceType":"ESP32","deviceName":"Maceta IoT","ts":"1600614612106"}
2020-09-20 17:10:12,369 [ForkJoinPool-3-worker-44] INFO  o.t.rule.engine.action.TbLogNode -
```

Ilustración 67: Logs Mensaje Recibido

Otra de las funciones que realiza dicha regla, es la de realiza una petición al servicio REST para persistir los datos enviado por el dispositivo. Para ello cuenta con un nodo del tipo ‘rest api call’, representado en la ilustración 15, donde se enviaran el identificador del dispositivo y el identificador del cliente como cabeceras de dicha petición. Para la evaluación de este nodo se añaden otro nodo de tipo ‘log’, que mostrará por consola la respuesta del servicio REST a la petición enviada. El resultado que se espera recibir desde el servicio REST es un ‘*HttpResponse*’ con un código 200, lo que significaría que la operación de persistir en base de datos se ha ejecutado correctamente. En la ilustración 68 se observa como el log mostrado por consola verifica que el funcionamiento es correcto.

```
PERSISTIR DATOS
MENSAJE =>[{"id":57449,"sensor":{"id":6,"nombre":"sensor_peso","tipoSensorId":"Sensor Peso"},"valor":"593","fecha":"2020-10-25T13:26:50.617+0000"},{"id":57450,"sen
sor":{"id":3,"nombre":"humedad_ambiental","tipoSensorId":"Humedad Ambiental"},"valor":"0.0","fecha":"2020-10-25T13:26:50.624+0000"},{"id":57451,"sensor":{"id":5,"n
ombre":"Co2","tipoSensorId":"Co2"},"valor":"449.0","fecha":"2020-10-25T13:26:50.630+0000"},{"id":57452,"sensor":{"id":4,"nombre":"luz_ambiental","tipoSensorId":"Lu
minosidad"},"valor":"0.0","fecha":"2020-10-25T13:26:50.634+0000"},{"id":57453,"sensor":{"id":7,"nombre":"temperatura_ambiental","tipoSensorId":"Temperatura Ambient
al"},"valor":"0.0","fecha":"2020-10-25T13:26:50.639+0000"},{"id":57454,"sensor":{"id":2,"nombre":"agua","tipoSensorId":"Detector de Agua"},"valor":"0.0","fecha":"2
020-10-25T13:26:50.643+0000"},{"id":57455,"sensor":{"id":1,"nombre":"humedad_suelo_inferior","tipoSensorId":"Humedad Superficie"},"valor":"29","fecha":"2020-10-25T
13:26:50.651+0000"}]
METADATA =>{"deviceType":"ESP32","content-length":"988","statusReason":"OK","Connection":"close","Vary":"Origin","deviceName":"Maceta IoT","Date":"Sun, 25 Oct 2020
 13:26:50 GMT","Content-Type":"application/json","ts":"1603632410530","status":"OK","statusCode":"200"}
2020-10-25 14:26:50,666 [ForkJoinPool-3-worker-11] INFO  o.t.rule.engine.action.TbLogNode -
```

Además de estos tres nodos y como hemos comentado anteriormente, desde la regla principal se ejecutan otras cuatro reglas diferentes, la cuales se evalúan a continuación.

En la regla de ‘Detección de agua’ reflejada en la ilustración 16, se espera que recibida el mensaje de la ‘regla principal’, compruebe si el valor del parámetro ‘agua’ es igual a 1 o a 0 y en caso de que el valor sea igual a 1 envíe dos mensajes, uno para modificar telemetrías y otro para modificar atributos del dispositivo. Para evaluar que la regla cumple su función se han incluido unos logs que muestran el mensaje que recibe la regla y el mensaje saliente de cada uno de los script de esta regla. Además se ha añadido un tercer nodo de tipo ‘log’ para comprobar que el nodo de tipo ‘*rest api call*’ realiza la petición al servicio REST correctamente. En la ilustración 69, refleja el caso de que la regla reciba un valor para el sensor de agua igual a cero, por tanto los script deben enviar un mensaje vacío ya que no se requiere realizar ninguna acción y además no se debe registrar ningún evento.

```
DETECCION DE AGUA (MENSAJE ENTRANTE)
MENSAJE =>{"humedad_ambiental":53.02,"luz_ambiental":0,"humedad_suelo":0,"sensor_peso":656,"agua":0,"Co2":449,"temperatura_ambiental":23.28}
METADATA =>{"deviceType":"ESP32","deviceName":"Maceta IoT","ts":"1600616369839"}
2020-09-20 17:39:30,220 [ForkJoinPool-3-worker-23] INFO  o.t.rule.engine.action.TbLogNode -

NIVEL DE HUMEDAD DE LA TIERRA INFERIOR (MENSAJE ENTRANTE)
MENSAJE =>{"humedad_ambiental":53.02,"luz_ambiental":0,"humedad_suelo":0,"sensor_peso":656,"agua":0,"Co2":449,"temperatura_ambiental":23.28}
METADATA =>{"deviceType":"ESP32","deviceName":"Maceta IoT","ts":"1600616369839"}
2020-09-20 17:39:30,236 [ForkJoinPool-3-worker-37] INFO  o.t.rule.engine.action.TbLogNode -

DETECCION DE AGUA
MENSAJE =>{}
METADATA =>{"deviceType":"ESP32","deviceName":"Maceta IoT","ts":"1600616369839"}
2020-09-20 17:39:30,236 [ForkJoinPool-3-worker-50] INFO  o.t.rule.engine.action.TbLogNode -

DETECCION DE AGUA
MENSAJE =>{}
METADATA =>{"deviceType":"ESP32","deviceName":"Maceta IoT","ts":"1600616369839"}
2020-09-20 17:39:30,236 [ForkJoinPool-3-worker-23] INFO  o.t.rule.engine.action.TbLogNode -
```

Ilustración 68: Logs agua = 0

En la ilustración 68 se observa como la regla recibe el valor del parámetro agua igual a 1 y en consecuencia envía los mensajes para modificar los atributos y las telemetrías necesarias. Además se observa que la respuesta del servicio REST, a la petición de registrar el evento, es de OK.

```
TECCION DE AGUA (MENSAJE ENTRANTE)
NSAJE =>{"humedad_ambiental":53.59,"luz_ambiental":0,"humedad_suelo":0,"sensor_peso":527,"agua":1,"Co2":449,"temperatura_ambiental":23.2}
TADATA =>{"deviceType":"ESP32","deviceName":"Maceta IoT","ts":"1600616972867"}
020-09-20 17:49:33,257 [ForkJoinPool-3-worker-29] INFO  o.t.rule.engine.action.TbLogNode -

IEGO MANUAL =>{"riegoManual":false}
TADATA =>{"deviceType":"ESP32","deviceName":"Maceta IoT","ts":"1600616972867"}
020-09-20 17:49:33,257 [ForkJoinPool-3-worker-57] INFO  o.t.rule.engine.action.TbLogNode -

DESCONEXION =>{"T_DESCONEXION":0}
TADATA =>{"deviceType":"ESP32","deviceName":"Maceta IoT","ts":"1600616972867"}
020-09-20 17:49:33,258 [ForkJoinPool-3-worker-37] INFO  o.t.rule.engine.action.TbLogNode -

VEL DE HUMEDAD DE LA TIERRA INFERIOR (MENSAJE ENTRANTE)
NSAJE =>{"humedad_ambiental":53.59,"luz_ambiental":0,"humedad_suelo":0,"sensor_peso":527,"agua":1,"Co2":449,"temperatura_ambiental":23.2}
TADATA =>{"deviceType":"ESP32","deviceName":"Maceta IoT","ts":"1600616972867"}
bernate: select actions0_.id as id1_0_, actions0_.action_desc as action_d2_0_ from action actions0_ where actions0_.action_desc in (?)
bernate: select cliente0_.id as id1_1_, cliente0_.descripcion as descripc2_1_, cliente0_.letra as letra3_1_, cliente0_.thingsboard_id as thingsbo4_1_ from client cliente0_ where
_.thingsboard_id in (?)
bernate: insert into events (action_id, cliente_id, timestamp) values (?, ?, ?)
020-09-20 17:49:34,931 [TB-Scheduling-1] INFO  o.t.server.actors.ActorSystemContext - Rule Engine JS Invoke Stats: requests [113] responses [113] failures [0]
020-09-20 17:49:34,984 [ForkJoinPool-3-worker-37] INFO  o.t.rule.engine.action.TbLogNode -

TECCION DE AGUA (EVENTO)
NSAJE =>{"id":1,"timestamp":"2020-09-20T15:49:33.368+0000","cliente":{"id":1,"thingsboardId":"593d8ac0-c13e-11ea-bd36-25db62324136","descripcion":"Javier","letra":""},"action":{"
ctionDesc":"cerrar el riego"}}
TADATA =>{"deviceType":"ESP32","content-length":"208","statusReason":"OK","Connection":"close","deviceName":"Maceta IoT","Date":"Sun, 20 Sep 2020 15:49:34 GMT","ts":"160061697286
us":"OK","statusCode":"200","Content-Type":"application/json"}
020-09-20 17:49:35,075 [ForkJoinPool-3-worker-8] INFO  o.t.rule.engine.action.TbLogNode -

VEL DE HUMEDAD DE LA TIERRA INFERIOR
NSAJE =>{}
TADATA =>{"deviceType":"ESP32","deviceName":"Maceta IoT","ts":"1600616972867"}
020-09-20 17:49:35,075 [ForkJoinPool-3-worker-15] INFO  o.t.rule.engine.action.TbLogNode -

TECCION DE AGUA
NSAJE =>{"riego":"stop"}
TADATA =>{"deviceType":"ESP32","deviceName":"Maceta IoT","ts":"1600616972867"}
020-09-20 17:49:35,075 [ForkJoinPool-3-worker-57] INFO  o.t.rule.engine.action.TbLogNode -

VEL DE HUMEDAD DE LA TIERRA INFERIOR
NSAJE =>{"humedad_ambiental":53.59,"luz_ambiental":0,"humedad_suelo":0,"sensor_peso":527,"agua":1,"Co2":449,"temperatura_ambiental":23.2}
TADATA =>{"deviceType":"ESP32","deviceName":"Maceta IoT","ts":"1600616972867"}
020-09-20 17:49:35,076 [ForkJoinPool-3-worker-30] INFO  o.t.rule.engine.action.TbLogNode -

TECCION DE AGUA
NSAJE =>{"rele":0,"leds":0}
TADATA =>{"deviceType":"ESP32","deviceName":"Maceta IoT","ts":"1600616972867"}
020-09-20 17:49:35,112 [ForkJoinPool-3-worker-15] INFO  o.t.rule.engine.action.TbLogNode -
```

Ilustración 69: Logs agua = 1

La regla 2, ‘Nivel de Humedad Inferior’, tiene una estructura muy similar a la regla de ‘Detección de Agua’, por lo que la forma de evaluar su comportamiento es idéntico que la descrita en el párrafo anterior. No obstante, para este caso se comparará el nivel de humedad de la tierra de la parte inferior con el límite establecido. En caso de que el valor recogido por el sensor sea mayor a dicho límite, esta cadena de reglas enviará dos mensajes, uno para modificar las telemetrías de ‘rele’ y ‘leds’ e igualarlas a 0 tal y como se observa en la ilustración 22; y un segundo mensaje para modificar el atributo ‘riego’ del dispositivo. En cuanto al nodo de tipo ‘rest api call’, el resultado que se desea obtener es el mismo que en la regla de ‘Detección de Agua’, una ‘*HttpReponse*’ con código 200. Para comprobar que se cumplen estas funcionalidades descritas se añaden 4 nodos de tipo ‘log’ para mostrar el mensaje entrante a la regla y los mensajes resultantes de los script y del nodo ‘rest api call’. En

la ilustración 71 se pueden observar los logs que permiten verificar el correcto funcionamiento, cuando el valor de la humedad de la tierra de la parte inferior es menor del 50%.

```
NIVEL DE HUMEDAD DE LA TIERRA INFERIOR (MENSAJE ENTRANTE)
MENSAJE =>{"humedad_ambiental":54.65,"luz_ambiental":0,"humedad_suelo_inferior":0,"humedad_suelo_superior":0,"sensor_peso":673,"agua":0,"Co2"
METADATA =>{"deviceType":"ESP32","deviceName":"Maceta IoT","ts":"1600620204027"}
2020-09-20 18:43:24,506 [ForkJoinPool-3-worker-2] INFO  o.t.rule.engine.action.TbLogNode -
NIVEL DE HUMEDAD DE LA TIERRA INFERIOR
MENSAJE =>{}
METADATA =>{"deviceType":"ESP32","deviceName":"Maceta IoT","ts":"1600620204027"}
2020-09-20 18:43:24,506 [ForkJoinPool-3-worker-1] INFO  o.t.rule.engine.action.TbLogNode -
DETECCION DE AGUA
MENSAJE =>{}
METADATA =>{"deviceType":"ESP32","deviceName":"Maceta IoT","ts":"1600620204027"}
2020-09-20 18:43:24,506 [ForkJoinPool-3-worker-37] INFO  o.t.rule.engine.action.TbLogNode -
DETECCION DE AGUA
MENSAJE =>{}
METADATA =>{"deviceType":"ESP32","deviceName":"Maceta IoT","ts":"1600620204027"}
2020-09-20 18:43:24,507 [ForkJoinPool-3-worker-59] INFO  o.t.rule.engine.action.TbLogNode -
CONTROL DEL TIEMPO DE DESCONEXION (MENSAJE ENTRANTE)
MENSAJE =>{}
METADATA =>{"shared_T_DESCONEXION":"70"}
2020-09-20 18:43:24,507 [ForkJoinPool-3-worker-8] INFO  o.t.rule.engine.action.TbLogNode -
NIVEL DE HUMEDAD DE LA TIERRA INFERIOR
MENSAJE =>{}
METADATA =>{"deviceType":"ESP32","deviceName":"Maceta IoT","ts":"1600620204027"}
2020-09-20 18:43:24,573 [ForkJoinPool-3-worker-37] INFO  o.t.rule.engine.action.TbLogNode -
```

Ilustración 70: Logs 'tierra inferior' < 50%

Para el segundo caso, cuando la humedad de la tierra de la parte inferior, es superior al 50, se aprecia en la ilustración 72 como esta regla envía dos mensajes para modificar los atributos y las telemetrías. Además de realizar una petición REST para registrar un evento.

```
NIVEL DE HUMEDAD DE LA TIERRA INFERIOR (MENSAJE ENTRANTE)
MENSAJE =>{"humedad_ambiental":55.19,"luz_ambiental":0,"humedad_suelo_inferior":57,"humedad_suelo_superior":57,"sensor_peso":528,"agua":
METADATA =>{"deviceType":"ESP32","deviceName":"Maceta IoT","ts":"1600620896802"}
2020-09-20 18:54:56,952 [ForkJoinPool-3-worker-37] INFO  o.t.rule.engine.action.TbLogNode -

DETECCION DE AGUA (MENSAJE ENTRANTE)
MENSAJE =>{"humedad_ambiental":55.19,"luz_ambiental":0,"humedad_suelo_inferior":57,"humedad_suelo_superior":57,"sensor_peso":528,"agua":
METADATA =>{"deviceType":"ESP32","deviceName":"Maceta IoT","ts":"1600620896802"}
Hibernate: select actions0_.id as id1_0_, actions0_.action_desc as action_d2_0_ from action actions0_ where actions0_.action_desc in (?)
Hibernate: select cliente0_.id as id1_1_, cliente0_.descripcion as descripc2_1_, cliente0_.letra as letra3_1_, cliente0_.thingsboard_id
e0_.thingsboard_id in (?)
Hibernate: insert into events (action_id, cliente_id, timestamp) values (?, ?, ?)
2020-09-20 18:54:57,048 [ForkJoinPool-3-worker-52] INFO  o.t.rule.engine.action.TbLogNode -

NIVEL DE HUMEDAD DE LA TIERRA INFERIOR
MENSAJE =>{"id":10,"timestamp":"2020-09-20T16:54:56.996+0000","cliente":{"id":1,"thingsboardId":"593d8ac0-c13e-11ea-bd36-25db62324136","
,"actionDesc":"cerrar el riego"}}
METADATA =>{"deviceType":"ESP32","content-length":"209","statusReason":"OK","Connection":"close","deviceName":"Maceta IoT","Date":"Sun,
atus":"OK","statusCode":"200","Content-Type":"application/json"}
2020-09-20 18:54:57,052 [ForkJoinPool-3-worker-37] INFO  o.t.rule.engine.action.TbLogNode -

NIVEL DE HUMEDAD DE LA TIERRA INFERIOR
MENSAJE =>{"rele":0,"leds":0}
METADATA =>{"deviceType":"ESP32","deviceName":"Maceta IoT","ts":"1600620896802"}
2020-09-20 18:54:57,052 [ForkJoinPool-3-worker-8] INFO  o.t.rule.engine.action.TbLogNode -

DETECCION DE AGUA
MENSAJE =>{}
METADATA =>{"deviceType":"ESP32","deviceName":"Maceta IoT","ts":"1600620896802"}
2020-09-20 18:54:57,052 [ForkJoinPool-3-worker-29] INFO  o.t.rule.engine.action.TbLogNode -

DETECCION DE AGUA
MENSAJE =>{}
METADATA =>{"deviceType":"ESP32","deviceName":"Maceta IoT","ts":"1600620896802"}
2020-09-20 18:54:57,052 [ForkJoinPool-3-worker-16] INFO  o.t.rule.engine.action.TbLogNode -

NIVEL DE HUMEDAD DE LA TIERRA INFERIOR
MENSAJE =>{"riego":"stop"}
METADATA =>{"deviceType":"ESP32","deviceName":"Maceta IoT","ts":"1600620896802"}
2020-09-20 18:54:57,244 [ForkJoinPool-3-worker-8] INFO  o.t.rule.engine.action.TbLogNode -
```

Ilustración 71: Logs 'tierra inferior' > 50%

Cuando la tercera regla recibe el mensaje desde la 'Regla Principal' se espera que compruebe que el valor recibido por el sensor de humedad de la tierra de la parte superior, no sea inferior al establecido. En caso de que este nivel de humedad sea inferior al establecido se enviarán dos mensajes, uno para modificar las telemetrías de 'rele' y 'leds' e indiciar al dispositivo IoT que es necesario abrir el riego y otro para modificar el atributo 'riego' del dispositivo. Para evaluar estas funcionalidades se han añadido una serie de nodos de tipo 'log' que permiten ver el mensaje entrante de la regla y el mensaje saliente de los scripts. En la ilustración 73 se observa la primera casuística en la cual el nivel de humedad de la parte superior es mayor que el 30%, por lo que los mensajes salientes de los scripts deben ser vacíos y la petición Rest no se debe realizar.

```
NIVEL DE HUMEDAD DE LA TIERRA SUPERIOR (MENSAJE ENTRADA)
MENSAJE =>{"humedad_ambiental":56.99,"luz_ambiental":0,"humedad_suelo_inferior":56,"humedad_suelo_superior":56,"sensor_peso":532,"agua":0,"Co2":4
METADATA =>{"deviceType":"ESP32","deviceName":"Maceta IoT","ts":"1600621318323"}
2020-09-20 19:02:03,343 [ForkJoinPool-3-worker-37] INFO  o.t.rule.engine.action.TbLogNode -
```

```
NIVEL DE HUMEDAD DE LA TIERRA SUPERIOR
MENSAJE =>{}
METADATA =>{"deviceType":"ESP32","shared_riego":"stop","shared_T_RIEGO_ACTUAL":"3360","deviceName":"Maceta IoT","ts":"1600621318323"}
2020-09-20 19:02:03,343 [ForkJoinPool-3-worker-59] INFO  o.t.rule.engine.action.TbLogNode -
```

```
NIVEL DE HUMEDAD DE LA TIERRA SUPERIOR(ACTIVAR RIEGO)
MENSAJE =>{}
METADATA =>{"deviceType":"ESP32","shared_riego":"stop","shared_T_RIEGO_ACTUAL":"3360","deviceName":"Maceta IoT","ts":"1600621318323"}
2020-09-20 19:02:03,355 [ForkJoinPool-3-worker-16] INFO  o.t.rule.engine.action.TbLogNode -
```

Ilustración 72: Logs 'tierra superior' > 30%

Para el segundo caso, donde el valor recibido es inferior al 30%, la regla debera enviar dos mensajes para modificar los atributos y las telemetrias necesarias.

```
NIVEL DE HUMEDAD DE LA TIERRA SUPERIOR (MENSAJE ENTRADA)
MENSAJE =>{"humedad_ambiental":55.7,"luz_ambiental":0,"humedad_suelo_inferior":0,"humedad_suelo_superior":0,"sensor_peso":627,"agua":0,"
METADATA =>{"deviceType":"ESP32","deviceName":"Maceta IoT","ts":"1600623245735"}
2020-09-20 19:34:10,781 [ForkJoinPool-3-worker-36] INFO  o.t.rule.engine.action.TbLogNode -
```

```
NIVEL DE HUMEDAD DE LA TIERRA SUPERIOR
MENSAJE =>{"rele":1,"leds":"azul"}
METADATA =>{"deviceType":"ESP32","shared_riego":"stop","shared_T_RIEGO_ACTUAL":"36","deviceName":"Maceta IoT","ts":"1600623245735"}
2020-09-20 19:34:10,781 [ForkJoinPool-3-worker-16] INFO  o.t.rule.engine.action.TbLogNode -
```

```
NIVEL DE HUMEDAD DE LA TIERRA SUPERIOR(ACTIVAR RIEGO)
MENSAJE =>{"T_RIEGO_ACTUAL":0,"riego":"start"}
METADATA =>{"deviceType":"ESP32","shared_riego":"stop","shared_T_RIEGO_ACTUAL":"36","deviceName":"Maceta IoT","ts":"1600623245735"}
2020-09-20 19:34:10,795 [ForkJoinPool-3-worker-36] INFO  o.t.rule.engine.action.TbLogNode -
```

Ilustración 73: Logs 'tierra superior < 30%'

Por otro lado, cuando el riego está activo, esta regla se encarga de modificar el atributo ‘T_RIEGO_ACTUAL’ para ir aumentado su valor en una unidad cada segundo. Esta funcionalidad se realiza mediante el nodo de tipo ‘*generator*’ el cual envía un mensaje vacío cada segundo. Además de modificar dicho atributo para reflejar el tiempo de riego que lleva actualmente la maceta, se envía un mensaje a la regla 4, para que esta sea capaz de realizar la lógica para establecer el menor tiempo de riego en la maceta. Para evaluar esta parte de la regla ‘Humedad Superior Insuficiente’ se va a utilizar la propia regla 4 la cual incluye nodos de tipo ‘log’ que permitirán verificar que el control del tiempo de riego funciona correctamente.

La regla de ‘Control tiempo riego’ que se puede observar en la ilustración 28, recibirá el mensaje enviado desde la regla 3 cada segundo tal y como se describe en el párrafo anterior. Posteriormente cuando el riego se detenga se espera que esta regla compruebe si el valor del atributo ‘T_RIEGO_ACTUAL’ es menor que el tiempo de riego establecido en la maceta, es caso afirmativo se modifica el atributo ‘T_RIEGO_MACETA’ del dispositivo y se establece el tiempo de riego de la maceta

con el tiempo de riego actual, como se observa en el script de la ilustración 29.Para verificar que esta regla realiza su funcionalidad correctamente se ha añadido un log que muestra el mensaje entrante de la regla y el mensaje resultante del script. En la ilustración 75 se puede observar como esta regla recibe cada segundo un mensaje con 'T_RIEGO_ACTUAL'. Puesto que en este caso el tiempo de riego supera el establecido por el usuario no se modificara.

```
CONTROL DEL TIEMPO DE RIEGO (MENSAJE ENTRANTE)
MENSAJE =>{"T_RIEGO_ACTUAL":37}
METADATA =>{"shared_riego":"start","shared_T_RIEGO_MACETA":"30","shared_T_RIEGO_ACTUAL":"36"}
2020-09-20 22:34:46,265 [ForkJoinPool-3-worker-1] INFO  o.t.rule.engine.action.TbLogNode -

CONTROL DEL TIEMPO DE RIEGO
MENSAJE =>{}
METADATA =>{"shared_riego":"start","shared_T_RIEGO_MACETA":"30","shared_T_RIEGO_ACTUAL":"36"}
2020-09-20 22:34:46,319 [ForkJoinPool-3-worker-2] INFO  o.t.rule.engine.action.TbLogNode -

CONTROL DEL TIEMPO DE DESCONEXION (MENSAJE ENTRANTE)
MENSAJE =>{}
METADATA =>{"shared_T_DESCONEXION":"14"}
2020-09-20 22:34:46,321 [ForkJoinPool-3-worker-1] INFO  o.t.rule.engine.action.TbLogNode -

CONTROL DEL TIEMPO DE DESCONEXION
MENSAJE =>{"T_DESCONEXION":15}
METADATA =>{"shared_T_DESCONEXION":"14"}
2020-09-20 22:34:46,325 [ForkJoinPool-3-worker-51] INFO  o.t.rule.engine.action.TbLogNode -

ENIVAR AVISO POR DESCONEXION (MENSAJE ENTRANTE)
MENSAJE =>{"T_DESCONEXION":15}
METADATA =>{"shared_T_DESCONEXION":"14","shared_T_DESCONEXION_MAX":"300"}
2020-09-20 22:34:47,243 [ForkJoinPool-3-worker-22] INFO  o.t.rule.engine.action.TbLogNode -

CONTROL DEL TIEMPO DE RIEGO (MENSAJE ENTRANTE)
MENSAJE =>{"T_RIEGO_ACTUAL":38}
METADATA =>{"shared_riego":"start","shared_T_RIEGO_MACETA":"30","shared_T_RIEGO_ACTUAL":"37"}
2020-09-20 22:34:47,245 [ForkJoinPool-3-worker-22] INFO  o.t.rule.engine.action.TbLogNode -

CONTROL DEL TIEMPO DE RIEGO
MENSAJE =>{}
METADATA =>{"shared_riego":"start","shared_T_RIEGO_MACETA":"30","shared_T_RIEGO_ACTUAL":"37"}
2020-09-20 22:34:47,325 [ForkJoinPool-3-worker-1] INFO  o.t.rule.engine.action.TbLogNode -
```

Ilustración 74: Logs entrada

Sin embargo en la ilustración 76 se representa la casuística en la cual el tiempo de riego ha sido menor que el establecido. En este caso se espera que el script tenga un mensaje saliente en el cual se indique el nuevo ‘T_RIEGO_MACETA’.

```
CONTROL DEL TIEMPO DE RIEGO (MENSAJE ENTRANTE)
MENSAJE =>{}
METADATA =>{"shared_riego":"stop","shared_T_RIEGO_MACETA":"50","shared_T_RIEGO_ACTUAL":"20"}
2020-09-20 22:42:44,247 [ForkJoinPool-3-worker-51] INFO  o.t.rule.engine.action.TbLogNode -

CONTROL DEL TIEMPO DE RIEGO
MENSAJE =>{"T_RIEGO_MACETA":"20"}
METADATA =>{"shared_riego":"stop","shared_T_RIEGO_MACETA":"50","shared_T_RIEGO_ACTUAL":"20"}
2020-09-20 22:42:44,302 [ForkJoinPool-3-worker-22] INFO  o.t.rule.engine.action.TbLogNode -
```

Ilustración 75: Logs T_RIEGO_ACTUAL < T_RIEGO_MACETA

La quinta regla, ‘Control Tiempo Desconexión’, recibe un mensaje cada segundo desde el nodo de tipo ‘*generator*’ que se encuentra en la ‘Regla Principal’. Esta regla se debe encargar de aumentar en una unidad el valor del atributo ‘T_DESCONEXION’ por cada uno de los mensajes que le lleguen, es decir, un mensaje cada segundo. Esto permite llevar un contador del tiempo que pasa entre mensaje y mensaje recibido desde el dispositivo. Además de modificar los atributos, se espera que envíe un mensaje a la regla ‘Enviar Aviso Desconexión’ para que se encargue de controlar si se ha superado el tiempo máximo de desconexión. Para la evaluación de esta regla se han añadido dos logs uno para mostrar el mensaje entrante de la regla y poder verificar que recibe un mensaje cada segundo y otro log para mostrar el mensaje saliente del script. En la imagen 77 se observan los logs añadidos que permiten comprobar como el atributo ‘T_DESCONEXION’ se va incrementando en una unidad cada vez que se recibe un mensaje, es decir, cada segudo. Además en esta misma ilustración se pude observar cómo se envía el mensaje a la regla ‘Enviar Aviso desconexión’, con el valor de este atibuto..

```
CONTROL DEL TIEMPO DE DESCONEXION (MENSAJE ENTRANTE)
MENSAJE =>{}
METADATA =>{"shared_T_DESCONEXION":"57"}
2020-10-04 17:36:43,319 [ForkJoinPool-3-worker-8] INFO  o.t.rule.engine.action.TbLogNode -

CONTROL DEL TIEMPO DE DESCONEXION
MENSAJE =>{"T_DESCONEXION":58}
METADATA =>{"shared_T_DESCONEXION":"57"}
2020-10-04 17:36:43,340 [ForkJoinPool-3-worker-36] INFO  o.t.rule.engine.action.TbLogNode -

ENIVAR AVISO POR DESCONEXION (MENSAJE ENTRANTE)
MENSAJE =>{"T_DESCONEXION":58}
METADATA =>{"shared_T_DESCONEXION":"57","shared_T_DESCONEXION_MAX":"1800"}
2020-10-04 17:36:44,258 [ForkJoinPool-3-worker-23] INFO  o.t.rule.engine.action.TbLogNode -

CONTROL DEL TIEMPO DE RIEGO (MENSAJE ENTRANTE)
MENSAJE =>{"T_RIEGO_ACTUAL":136}
METADATA =>{"shared_riego":"start","shared_T_RIEGO_MACETA":"20","shared_T_RIEGO_ACTUAL":"135"}
2020-10-04 17:36:44,260 [ForkJoinPool-3-worker-8] INFO  o.t.rule.engine.action.TbLogNode -

CONTROL DEL TIEMPO DE RIEGO
MENSAJE =>{}
METADATA =>{"shared_riego":"start","shared_T_RIEGO_MACETA":"20","shared_T_RIEGO_ACTUAL":"135"}
2020-10-04 17:36:44,314 [ForkJoinPool-3-worker-23] INFO  o.t.rule.engine.action.TbLogNode -

CONTROL DEL TIEMPO DE DESCONEXION (MENSAJE ENTRANTE)
MENSAJE =>{}
METADATA =>{"shared_T_DESCONEXION":"58"}
2020-10-04 17:36:44,316 [ForkJoinPool-3-worker-8] INFO  o.t.rule.engine.action.TbLogNode -

CONTROL DEL TIEMPO DE DESCONEXION
MENSAJE =>{"T_DESCONEXION":59}
METADATA =>{"shared_T_DESCONEXION":"58"}
2020-10-04 17:36:44,340 [ForkJoinPool-3-worker-23] INFO  o.t.rule.engine.action.TbLogNode -

ENIVAR AVISO POR DESCONEXION (MENSAJE ENTRANTE)
MENSAJE =>{"T_DESCONEXION":59}
METADATA =>{"shared_T_DESCONEXION":"58","shared_T_DESCONEXION_MAX":"1800"}
```

Ilustración 76: Logs 'Control Tiempo Desconexión'

Por último se encuentra la regla 6, ‘Enviar Aviso Desconexión’, que se encarga de controlar si el tiempo de desconexión ha superado el máximo establecido. Para ello recibirá el mensaje de la regla 5 con el valor del tiempo de desconexión actual y se espera que compruebe dicho valor con el valor establecido en el atributo ‘T_DESCONEXION_MAX’. En caso de que el valor sea superior, esta cadena de reglas debe enviar un mensaje al móvil del administrador del dispositivo mediante la aplicación ‘Telegram’. Al igual que las reglas anteriores para evaluar la funcionalidad de esta regla se han añadido una serie de logs, en este caso se han añadido dos uno para ver el mensaje entrante y un segundo log en el que se ve le resultado del nodo de tipo ‘rest api call’, con el envío al chat de telegram. El resultado que se espera de este nodo de tipo ‘rest api call’, es una ‘*HttpResponse*’ con código 200.En la ilustración 78 se pueden observar los logs que permiten verificar el funcionamiento de esta regla, para la realización de esta prueba se establece el ‘T_DESCONEXION_MAX’ en 70 segundos de ahí que cuando la regla recibe el atributo ‘T_DESCONEXION’, superior a 70 segundos envía el mensaje..

```
ENIVAR AVISO POR DESCONEXION (MENSAJE ENTRANTE)
MENSAJE =>{"T_DESCONEXION":71}
METADATA =>{"shared_T_DESCONEXION":"70","shared_T_DESCONEXION_MAX":"70"}
2020-10-04 17:54:29,703 [ForkJoinPool-3-worker-44] INFO  o.t.rule.engine.action.TbLogNode -

ENIVADO AVISO POR DESCONEXION
MENSAJE =>{"ok":true,"result":{"message_id":91,"from":{"id":1127642305,"is_bot":true,"first_name":"Maceta","username":"Maceta_bot"},"chat":{"id":1128356456,"first_name":"Javier","t
private"},"date":1601826869,"text":"Conexion Perdida"}}
METADATA =>{"Server":"nginx/1.16.1","Access-Control-Allow-Origin":"*","Access-Control-Allow-Methods":"GET, POST, OPTIONS","Connection":"close","shared_T_DESCONEXION_MAX":"70","Date
, 04 Oct 2020 15:54:29 GMT","Strict-Transport-Security":"max-age=31536000; includeSubDomains; preload","Access-Control-Expose-Headers":"Content-Length,Content-Type,Date,Server,Conn
","statusReason":"OK","shared_T_DESCONEXION":"70","Content-Length":"231","Content-Type":"application/json","status":"OK","statusCode":"200"}
2020-10-04 17:54:30,250 [ForkJoinPool-3-worker-2] INFO  o.t.rule.engine.action.TbLogNode -
```

Ilustración 77: Logs envío aviso desconexión

Las evaluaciones de cada una de las cadenas de reglas que se acaban de realizar permiten comprobar que el flujo de datos entrante desde el dispositivo IoT, realiza la lógica que se esperaba en la plataforma IoT.

5. Conclusiones

Desde la Universidad Politécnica de Madrid, se me propuso realizar un proyecto que fuera capaz de alojar cualquier número de sistemas IoT que pudiera llegar a tener la universidad. Aprovechando los maceteros que se iban a desarrollar en la asignatura de Sistemas Basados en Computador, la idea era crear una plataforma capaz de alojar todos los dispositivos que hicieran mis compañeros y poder mostrar el estado de cada una de las plantas.

A consecuencia de esa propuesta realice una investigación acerca de que es el mundo IoT en detalle, qué importancia tenía en nuestra sociedad y cuáles podrían ser los beneficios y desventajas de este concepto.

Es un hecho que el mundo del 'Internet de las cosas' ha adquirido una gran importancia en los últimos diez años. Hoy en día la mayoría de las casas cuentan con uno, dos o incluso más dispositivos IoT que permanentemente se intercambian información con otros dispositivos. Extrapolando esto al entorno empresarial, donde las grandes fábricas cuentan con un enorme número de dispositivos IoT que les permiten controlar y mejorar el funcionamiento de la fábrica. La posibilidad de controlar todos estos dispositivos desde una misma plataforma y poder almacenar los datos recogidos por los sensores u otros dispositivos, en una base de datos propia abre un gran abanico de opciones.

El interés que este concepto me causó, me llevó a aceptar la propuesta del profesorado. Para dar una solución a dicha propuesta tuve que adquirir conocimientos acerca de las plataformas IoT, más concreta la plataforma ThingsBoard que fue la seleccionada para este proyecto. Durante este proceso de aprendizaje descubrí el enorme potencial que realmente tiene el mudo de las IoT y la cantidad de aplicaciones que puede llegar a tener.

Puesto que mi idea inicial era desarrollar una aplicación web, para realizar este proyecto decidí implementar un servicio que se comunicara con la plataforma IoT para intercambiar información y poder alojar cualquier número de dispositivo que tenga la plataforma IoT, en un espacio de almacenamiento propio de la universidad. De esta forma con los valores tomados por cada uno de los dispositivos de la plataforma la universidad podría generarse '*datasheet*' para su posterior explotación.

Para la representación de estos dispositivos (maceteros) que se iban a realizar en la asignatura Sistemas Basados en Computador se realizó un estudio de cuáles

eran las mejores formas de representación de dicho dispositivo. Finalmente se decidió desarrollar una interfaz gráfica que permitirá ver el estado general de cada una de las plantas y además tener la opción de ver los valores recogidos de forma gráfica.

Al igual que en el caso concreto de este proyecto, en el cual se han desarrollado macetas como dispositivos IoT es fácilmente extrapolable a cualquier otro tipo de dispositivos.

Puedo concluir afirmando que el esfuerzo y el tiempo dedicado a este proyecto me han permitido adquirir una gran cantidad de conocimientos sobre el mundo IoT y cómo funciona. Cierto es que la idea inicial que yo tenía no era relacionada con el mundo del ‘Internet de las cosas’ sin embargo es un tema que me ha resultado realmente interesante y me ha permitido cumplir mi idea de desarrollar una aplicación web.

6. Impactos éticos, sociales y ambientales

En esta sección se pretende realizar un análisis sobre los diferentes impactos que puede tener el desarrollo de este proyecto en diferentes ámbitos. Se entiende como impacto, en el caso concreto de este proyecto, el cambio a nivel social, ambiental o ético debido a algún elemento o aspecto del mismo.

Este proyecto pretende permitir a las personas gestionar sus diferentes dispositivos IoT desde una misma plataforma, además de almacenar en propiedad todos los datos enviados por estos dispositivos. Hoy en día la información, los datos, es una de los factores más valiosos a nivel empresarial puesto que permite a grandes empresas saber mucha más información privada sobre las personas y con ello realizar estudios de mercado mucho más efectivos, lo que les permite aumentar sus ingresos. Sin embargo si cada persona fuese propietaria de la información de sus propios dispositivos IoT, la información privada que estos dispositivos sean capaces de recoger no irá a parar a las empresa, respetando la intimidad de cada persona.

El dispositivo utilizado para la evaluación de este proyecto permite automatizar la tarea de cuidado de una planta. Este dispositivo fue diseñado con la idea de fomentar algunos 'Objetivos de Desarrollo Sostenible' como por ejemplo el ODS 11 'Ciudades y comunidades sostenibles' o el 13 'Acción por el clima'. No obstante extrapolando este tipo de dispositivos a una gran empresa que pueda contar con un gran número de estos dispositivos para el cuidado de su cultivo, el impacto medioambiental que puede suponer es mucho mayor.

Sin embargo esta automatización en grandes cultivos de empresas podría provocar la disminución del empleo en sectores como la agricultura o la floricultura, debido a que el personal necesario para gestionar y controlar la plataforma desarrollada en este proyecto, no requiere de un gran número de personas. No obstante, a nivel empresarial, la posibilidad de automatizar los procesos de cuidado de un gran cultivo o de varios, puede suponer un gran impulso en cuanto al ahorro de costes y al aumento en la producción.

En conclusión, se puede afirmar que la plataforma desarrolla en este proyecto genera un impacto a nivel social, ambiental y ético en mayor o menos medida, dependiente del ámbito en el que se implante.

Referencias

¿Qué es una API? (n.d.). Retrieved June 22, 2020, from https://www.redhat.com/es/topics/api/what-are-application-programming-interfaces

¿Qué son las reglas de cortafuegos? | Client Security for Windows | 13.10 | F-Secure User Guides. (n.d.). Retrieved June 22, 2020, from https://help.f-secure.com/product.html?business/client-security/13.10/es-mx/concept_F980D184D74D4FF0A79562231548178C-13.10-es-mx

Angular - Introduction to the Angular Docs. (n.d.). Retrieved June 3, 2020, from https://angular.io/docs

Arduino - Introduction. (n.d.). Retrieved June 27, 2020, from https://www.arduino.cc/en/Guide/Introduction#

Asemani, M., Abdollahei, F., & Jabbari, F. (2019). Understanding IoT Platforms: Towards a comprehensive definition and main characteristic description. In *2019 5th International Conference on Web Research, ICWR 2019*. Institute of Electrical and Electronics Engineers Inc. https://doi.org/10.1109/ICWR.2019.8765259

Luis, C. X. J., Luz, R. S. M. de la, Paola, C. C. D., & Stefhan, L. G. (n.d.). *Interfaz Gráfica : - "CONCEPTOS BÁSICOS DE COMPUTACIÓN."* Retrieved June 7, 2020, from https://sites.google.com/site/computadora8991/6-sistemas-operativos/interfaz-grafica

Objetivos y metas de desarrollo sostenible – Desarrollo Sostenible. (n.d.). Retrieved June 24, 2020, from https://www.un.org/sustainabledevelopment/es/objetivos-de-desarrollo-sostenible/

Protocolos y tecnologías de IoT | Microsoft Azure. (n.d.). Retrieved June 7, 2020, from https://azure.microsoft.com/es-es/overview/internet-of-things-iot/iot-technology-protocols/

Qué es el protocolo SSL/TLS | Redalia. (n.d.). Retrieved June 7, 2020, from https://www.redalia.es/ssl/protocolo-ssl/

Spring Framework. (n.d.). Retrieved May 17, 2020, from https://spring.io/projects/spring-framework

Tabuenca, B., García-Alcántara, V., Gilarranz-Casado, C., & Barrado-Aguirre, S. (2020). Fostering Environmental Awareness with Smart IoT Planters in Campuses. *Sensors*, *20*(8), 2227. https://doi.org/10.3390/s20082227

Telefónica. (2019). *Estudio Things Matter 2019 | Telefónica IoT*. https://iot.telefonica.com/es/whats-new/multimedia/estudio-things-matter-2019/

The ThingsBoard Authors. (2020). *Getting Started with Rule Engine | ThingsBoard*. https://thingsboard.io/docs/user-guide/rule-engine-2-0/re-getting-started/

Zanella, A., Bui, N., Castellani, A., Vangelista, L., & Zorzi, M. (2014). Internet of things for smart cities. *IEEE Internet of Things Journal*, *1*(1), 22–32. https://doi.org/10.1109/JIOT.2014.2306328

Tabla de ilustraciones