ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

UNIVERSIDAD DE CANTABRIA

Trabajo Fin de Grado

IMPLEMENTACIÓN DE INTERFACES ASÍNCRONOS SEGUROS EN UNA PLATAFORMA DE LA INTERNET DE LAS COSAS EN LA CIUDAD INTELIGENTE

(Implementation of Secure Asynchronous Interfaces for an Internet of Things Platform in the Smart City)

Para acceder al Título de

Graduado en
Ingeniería de Tecnologías de Telecomunicación

Autor: Laura Martín González

Julio - 2020

E.T.S. DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACION

GRADUADO EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

CALIFICACIÓN DEL TRABAJO FIN DE GRADO

Realizado por: Laura Martín González
Directores del TFG: Luis Sánchez González y Pablo Sotres García
Título: "Implementación de Interfaces Asíncronos Seguros en una Plataforma de la Internet de las Cosas en la Ciudad Inteligente"
Title: "Implementation of Secure Asynchronous Interfaces for an Internet of Things Platform in the Smart City"

Presentado a examen el día: 27 de Julio de 2020

para acceder al Título de

GRADUADO EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

Composición del Tribunal:
Presidente (Apellidos, Nombre): Madruga Saavedra, Francisco Javier
Secretario (Apellidos, Nombre): Lanza Calderón, Jorge
Vocal (Apellidos, Nombre): Sánchez González, Luis

Este Tribunal ha resuelto otorgar la calificación de: .....

Fdo.: El Presidente

Fdo.: El Secretario

Fdo.: El Vocal

Fdo.: El Director del TFG
(sólo si es distinto del Secretario)

Vº Bº del Subdirector

Trabajo Fin de Grado Nº
(a asignar por Secretaría)

Resumen

En la actualidad, las *Ciudades Inteligentes* están cobrando una gran importancia como consecuencia del crecimiento exponencial de la población urbana y la necesidad de mejora de los servicios ofrecidos a los ciudadanos. Un ejemplo de Ciudad Inteligente es Santander, que hospeda el proyecto SmartSantander, con múltiples dispositivos IoT desplegados por la ciudad que disponen de sensores para diferentes parámetros y fenómenos físicos como el estado del tráfico, nivel de ruido o contaminación atmosférica. SmartSantander ofrece un acceso asíncrono a la información recogida por los sensores en tiempo real a través de tecnologías como RestHooks y OML, entre otras. El objetivo de este proyecto es dotar a la plataforma SmartSantander de nuevas formas de soportar el acceso seguro y asíncrono a las observaciones y medidas que generan los dispositivos IoT desplegados. Se han diseñado e implementado dos interfaces para proporcionar el acceso seguro y asíncrono mediante las tecnologías Websockets y MQTT, respectivamente. Para validar y permitir una visualización de las funcionalidades de los interfaces se han desarrollado diferentes tipos de aplicaciones cliente: cliente de línea de comando y cliente web con interfaz gráfica.

Abstract

Nowadays, Smart Cities are gaining great importance as a result of the exponential growth of the urban population and the need to improve the services offered to citizens. Santander is an example of a Smart City, which hosts the SmartSantander project, with multiple IoT devices deployed throughout the city including sensors for different physical phenomena and parameters such as noise level, pollution or traffic status. SmartSantander provides an asynchronous access to the sensors' collected real-time information through technologies such as RestHooks and OML, among others. The aim of this BSc Degree Thesis is to provide the SmartSantander platform with new ways of supporting secure and asynchronous access to the observations and measurements generated by the deployed IoT devices. Two interfaces have been designed and implemented using Websockets and MQTT technologies, respectively. In order to validate and allow the proof-of-concept demonstration of the interfaces' functionalities, different types of client applications have been developed: command-line client and web client with graphic interface.

Índice general

Índice de tablas

Índice de figuras

Lista de acrónimos

ACL Access Control List.

API Application Programming Interface.

HTTP Hypertext Transfer Protocol.

HTTPS Hypertext Transfer Protocol Secure.

IoT Internet of Things.

JSON JavaScript Object Notation.

LPWAN Low Power Wide Area Network.

M2M Machine To Machine.

MQTT Message Queuing Telemetry Transport.

REST REpresentational State Transfer.

SoC System on Chip.

TCP Transmission Control Protocol.

TLS Transport Layer Security.

URI Uniform Resource Identifier.

Capítulo 1

Introducción

1.1. Motivación

El siglo XXI está llamado a ser el siglo de las ciudades [1]. La población urbana ha crecido de manera exponencial en los últimos años y continuará haciéndolo en los próximos, llegando a la estimación de un 70 % de población urbana para el año 2050 [2]. Esto supone numerosos retos para las ciudades, teniendo que atender una gran variedad de servicios públicos.

La *Smart City* (en castellano *Ciudad Inteligente*) aparece como un espacio urbano que une las infraestructuras y redes con sensores y actuadores, incluyendo incluso en esta segunda parte a las personas y sus teléfonos móviles. Gracias a esta unión es posible proporcionar información para la mejora de los servicios prestados a los ciudadanos. Entre las *Ciudades Inteligentes* más desarrolladas en el mundo se encuentra Singapur, donde en 2014 se desplegaron un gran número de sensores por toda la ciudad que captan todo tipo de información, llegando a medir desde el nivel de limpieza de un área hasta la cantidad de personas que se encuentran en un lugar concreto. Otro ejemplo es Oslo que, en este caso, se está centrado en impulsar el entorno ecológico y sostenible. Este plan *eco* se ve implementado en las numerosas bombillas LED que se encuentran desplegadas por la ciudad cuya potencia se ajusta de manera inteligente, y en un método de lectura de matrículas para aliviar las congestiones de tráfico [3]. Y por último, sin ir más lejos, Santander hospeda el proyecto de ciudad inteligente denominado *SmartSantander*. Como ya se ha indicado, el nexo común de estos y otros ejemplos de Ciudades Inteligentes es la aplicación de la tecnología embebida en el entorno para alcanzar un mayor y preciso conocimiento del mismo con el que poder tomar las mejores decisiones en cada momento.

SmartSantander tiene desplegados múltiples dispositivos (fijos y móviles) que alojan sensores para diferentes medidas como la temperatura, el ruido o la humedad. El proyecto engloba varios casos de uso y escenarios. Entre ellos: la monitorización medio ambiental, proporcionando información sobre la polución, los niveles de ruido y luminosidad; la cantidad de aparcamientos libres, colocando los sensores bajo el asfalto; y el control de los niveles y horarios de riego en parques y jardines, con objetivo de reducir costes al realizar dicha tarea de manera eficiente e inteligente [4].

1.2. Objetivos

Con el desarrollo del concepto de la Internet de las Cosas (IoT, por sus siglas en inglés) y su aplicación al paradigma de la ciudad inteligente se plantea la necesidad de homogeneizar el acceso a la información que una infraestructura masiva de dispositivos electrónicos desplegados por toda la ciudad y con un perfil heterogéneo tanto a nivel de capacidades de cómputo como de dominio de aplicación es capaz de generar. El objetivo principal de este proyecto es el desarrollo de dos interfaces que permitan el acceso seguro y asíncrono a las observaciones recogidas por los dispositivos desplegados por la ciudad. Los objetivos específicos que se articulan para la consecución del objetivo principal son:

- Analizar y comprender el diseño del acceso asíncrono a la plataforma SmartSantander.
- Estudiar las tecnologías Websockets y MQTT para su empleo en la plataforma.
- Diseñar los nuevos interfaces Websockets y MQTT para el acceso asíncrono a las medidas.
- Implementar los componentes incluidos en el diseño.
- Validar funcionalmente la implementación realizada.

1.3. Resumen ejecutivo

Este documento se estructura en cinco capítulos en los cuales se describe el trabajo realizado para alcanzar los objetivos descritos en el apartado anterior.

Tras haber introducido la motivación y los objetivos del proyecto en el Capítulo 1, en el Capítulo 2 se revisan y analizan los conceptos teóricos relativos a las tecnologías y protocolos utilizados a lo largo del proyecto, así como el marco tecnológico en el que se ha llevado a cabo. En él se revisan las tecnologías y soluciones ya existentes, se describe brevemente el proyecto y plataforma SmartSantander, haciendo hincapié en sus características más importantes y relevantes para el proyecto y se introducen las tecnologías que se usarán para su desarrollo.

En el Capítulo 3 se detallan las características básicas y comunes que debe cumplir cualquier tecnología para el desarrollo de los interfaces objetivo. Se expone la estructura básica de los módulos a desarrollar y cómo se gestiona el ciclo de vida de las suscripciones así como la distribución de las notificaciones a los clientes.

En el Capítulo 4 se hace énfasis en las diferencias de diseño e implementación entre las dos tecnologías utilizadas. Se analiza la arquitectura, el manejo de las notificaciones y el diseño del acceso seguro en mayor detalle. Asimismo, se exponen los tipos de clientes utilizados y desarrollados para la validación funcional del proyecto.

Por último, las principales conclusiones extraídas tras la finalización del proyecto, así como las posibles líneas de trabajo futuro se han resumido en el Capítulo 5.

Capítulo 2

Marco de desarrollo del proyecto

2.1. IoT

La Internet de las Cosas o Internet of Things (IoT) describe la agrupación e interconexión de dispositivos a través de una red, en la cual todos son visibles y pueden interaccionar entre ellos sin la intervención humana, es decir, una interacción máquina a máquina o Machine To Machine (M2M). El objetivo principal de la IoT es que todos los dispositivos estén conectados y sean capaces de recopilar información y transmitirla a otros para su posterior análisis y uso en la mejora del funcionamiento del propio dispositivo o del resto [7].

La IoT está en continua evolución, lo cual genera grandes diferencias en el proceso de comunicación entre los dispositivos más nuevos y los más antiguos. Es por esto que el protocolo utilizado debe ser válido para la comunicación entre dispositivos de distinta antigüedad o, incluso, fabricante. IBM ofrece una solución con el protocolo MQTT (Message Queuing Telemetry Transport), que, demandando unos recursos más, define una serie de funcionalidades para habilitar la comunicación entre cualquier dispositivo que lo implemente [7].

Los dispositivos utilizados en la IoT deben integrar, como mínimo, sensores, elemento esencial, para la obtención de información, un procesador y un transceptor para la comunicación en red. Contando con estas tres partes, el dispositivo además debe ser de pequeño tamaño y de bajo consumo, por lo que supone un reto adicional, dando gran importancia a los System on Chip (SoC) ó sistema en chip. Un SoC es un circuito integrado que contiene todos los módulos necesarios que tendría, por ejemplo, un ordenador [7].

La tecnología de comunicaciones utilizada para el despliegue de la IoT juega un papel fundamental. No se utiliza una tecnología por defecto, sino que se puede hablar de varias para su implementación. En cualquier caso, el requisito fundamental es que sean inalámbricos. Entre las tecnologías más generalistas, la tecnología WiFi permite una tasa alta de transferencia pero a relativamente poca distancia y con un consumo elevado. Las redes móviles (3G, 4G...) permiten un mayor alcance y menos consumo.[7]. En paralelo al desarrollo de la IoT y a causa de que las tecnologías existentes n se ajustaban perfectamente a las necesidades de este tipo de redes, han aparecido una serie de tecnologías

alrededor del concepto de área extensa de baja potencia o Low Power Wide Area Network (LPWAN). Entre ellas destaca SigFox (modelo cerrado) [5], LoRa (modelo abierto) [6] y NB-IoT. Esta última es una tecnología que nace como alternativa para los operadores móviles, ajustando su rango de cobertura a la del operador y ofreciendo una buena velocidad de bajada de datos [8].

2.2. Plataformas de acceso a la información de la IoT

Desplegar los dispositivos para embeberlos en el entorno y que estos recojan la información y transmitirla hacia la red es sólo una parte de la solución habilitada por la IoT, pero es necesario que dicha información esté disponible para que se puedan proveer servicios inteligentes basados en el conocimiento generado gracias a ella.

2.2.1. Requisitos básicos de las plataformas

Una plataforma IoT se comporta como un *Middleware*, (esto es, un software que actúa como una capa de traducción y adaptación que permite la comunicación entre un sistema operativo y las aplicaciones que se ejecutan en él), que se encarga de conectar el hardware (i.e. los dispositivos IoT) con las aplicaciones de usuario final mediante un software ajustado que habilita una comunicación transparente y la prestación de los servicios necesarios [9].

Pese a que existen grandes diferencias entre las múltiples plataformas IoT existentes, se pueden establecer unos requisitos básicos y características comunes entre ellas [10]:

- Conectividad y normalización: con diferentes protocolos y formatos de datos se garantiza la correcta transmisión de la información y la interacción con todos los dispositivos.
- Gestión de dispositivos: asegura que todos los dispositivos conectados funcionan correctamente.
- Persistencia de datos: almacenamiento escalable de los datos provenientes de los dispositivos.
- Procesado y gestión de eventos: aporta datos basados en reglas de acción de evento-disparadores que permitan la ejecución de acciones inteligentes basados en datos específicos del sensor.
- Análisis de datos: realiza análisis complejos sobre la información en crudo para generar información de valor añadido y de aprendizaje automático.
- Visualización: permite al usuario observar la representación gráfica de los datos.
- Herramientas adicionales: permiten a los desarrolladores visualizar, gestionar y controlar los dispositivos conectados.
- Interfaces externas: se integran a través de interfaces de programación de aplicaciones (API), kits de desarrollo de software (SDK) y puertas de enlace (*gateways*).

2.2.2. El API de la plataforma

Un interfaz de programación de aplicaciones o Application Programming Interface (API) es un conjunto de definiciones y protocolos que se utiliza para desarrollar e integrar el software de las aplicaciones. Permite que unos servicios se comuniquen con otros, sin necesidad de diseñar permanentemente una infraestructura de conectividad nueva, proporcionando un medio simplificado y flexible en cuanto a diseño, administración y uso de aplicaciones [11].

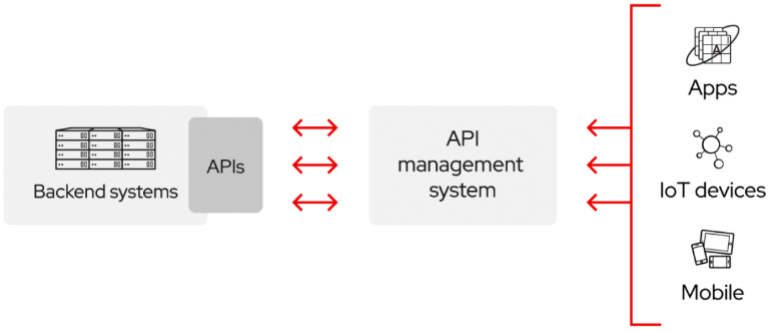

Figura 2.1: Esquema de un API [11].

Un API puede considerarse como un contrato, un acuerdo entre dos partes con documentación que representa la interacción entre el cliente y el servidor. Si el cliente envía una solicitud con un determinado formato, el API determinará cómo responde el servidor. Para conectarse a un API y crear aplicaciones que utilicen los datos o servicios ofrecidos, se puede hacer a través de plataformas de integración distribuida que conecten todos los elementos, incluyendo la IoT [11].

2.2.3. Servicios web basados en REST

REpresentational State Transfer (REST) es un estilo de arquitectura software apoyada sobre el protocolo HTTP (Hypertext Transfer Protocol), que permite crear servicios o aplicaciones que pueden ser usadas por cualquier dispositivo que implemente dicho protocolo. Hoy en día, la mayoría de los APIs diseñadas para servicios orientados a Internet utilizan la arquitectura REST y se denominan API RESTful [11].

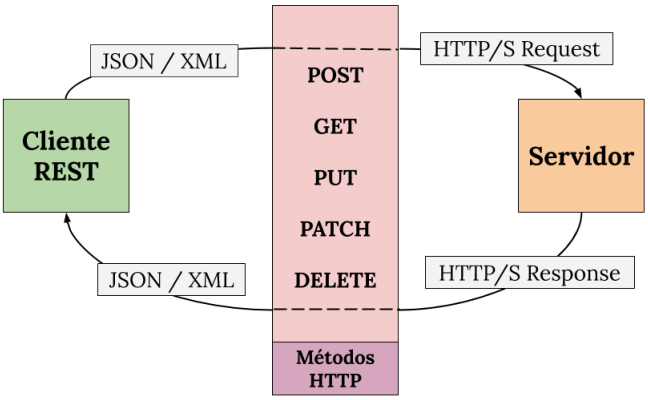

Figura 2.2: Arquitectura REST.

Los API RESTful han aportado a la web una mayor escalabilidad gracias, en gran medida a las siguientes características [11] [12]:

- Arquitectura cliente-servidor: la arquitectura está compuesta por clientes que envían peticiones y servidores que procesan dichas peticiones y devuelven las respuestas. Esto aporta independencia al tipo de implementación y desarrollo tanto de la parte cliente como de la servidora.
- Sin estado: las peticiones y respuestas HTTP siempre contienen toda la información necesaria para ser procesadas, de manera que únicamente el cliente es el encargado de guardar la información sobre su sesión.
- Capacidad de caché: el almacenamiento en caché puede eliminar la necesidad de algunas interacciones cliente-servidor.
- Sistema en capas: la interacción cliente-servidor puede estar intermediada por capas adicionales, que pueden ofrecer otras funciones, como el equilibrio de carga, las cachés compartidas o la seguridad.
- Interfaz uniforme:
  - Identificación de recursos/servicios en las solicitudes: los recursos/servicios son identificados mediante un identificador único de recurso o Uniform Resource Identifier (URI), que deben ser únicos e independientes del recurso, así como seguir una jerarquía lógica y homogénea.
  - Los recursos/servicios se describen utilizando lenguajes y formatos de modelado de la información. Estas descripciones del concepto abstracto del recurso/-servicio se denominan representaciones y contienen toda la información necesaria para manipular los recursos/servicios. Las operaciones sobre un recurso se codifican a través de los diferentes métodos HTTP (i.e. GET, POST, PUT, etc) y de las representaciones que se incluyan en cada mensaje intercambiado.
  - Mensajes autodescriptivos: las respuestas recibidas en el cliente se basan en códigos de estado HTTP. Estos códigos de tres dígitos representan y aportan la información sobre el estado de la respuesta, es decir, si todo ha ido correctamente, si ha habido un error por parte del servidor o por parte del cliente, etc.

- HATEOAS (Hypermedia As The Engine Of Application State): tras acceder a un recurso, el cliente debe ser capaz de descubrir mediante hipervínculos todas las otras acciones disponibles.

2.2.4. Servicios web asíncronos

Debido a la dependencia con HTTP de los API RESTful, estas tienen un comportamiento síncrono que se basa en el paradigma petición-respuesta. Sin embargo, la interacción entre los servicios web y sus clientes no siempre puede ser asíncrona, ya que es posible que incluyan tareas u operaciones que no se resuelven de manera local en el cliente, sino en el servidor y, por tanto, puede que dicha operación sea resuelta en el momento o en un futuro. El comportamiento de estas operaciones no debe ser bloqueante, lo que quiere decir que el cliente debe poder continuar realizando otras tareas mientras espera la respuesta del servidor. Gracias a esto se ahorra tiempo y la comunicación es más óptima.

Hay algunas tecnologías que, aún basándose también en el protocolo HTTP, sí que habilitan un comportamiento asíncrono y que se utilizan para complementar a los API RESTful en este aspecto. Algunas de estas tecnologías son: Long Polling, RestHooks o Websockets.

Long Polling

El Polling, la tecnología predecesora al Long Polling, se basa en el modelo clásico petición-respuesta, donde el cliente envía su petición al servidor y este devuelve la respuesta asociada. Para cada dupla petición-respuesta se crea y cierra una conexión TCP (Transmission Control Protocol). Asimismo, la respuesta del servidor debe ser inmediata, es decir, en caso de que la información requerida por el cliente no esté disponible o no sea nueva respecto a la petición anterior, el servidor enviará su respuesta vacía (o idéntica a la anterior). Este intercambio de información inútil, junto con la necesidad de una conexión TCP por cada intercambio petición-respuesta, genera una gran sobrecarga en la red, malgastando recursos y empobreciendo el rendimiento del sistema.

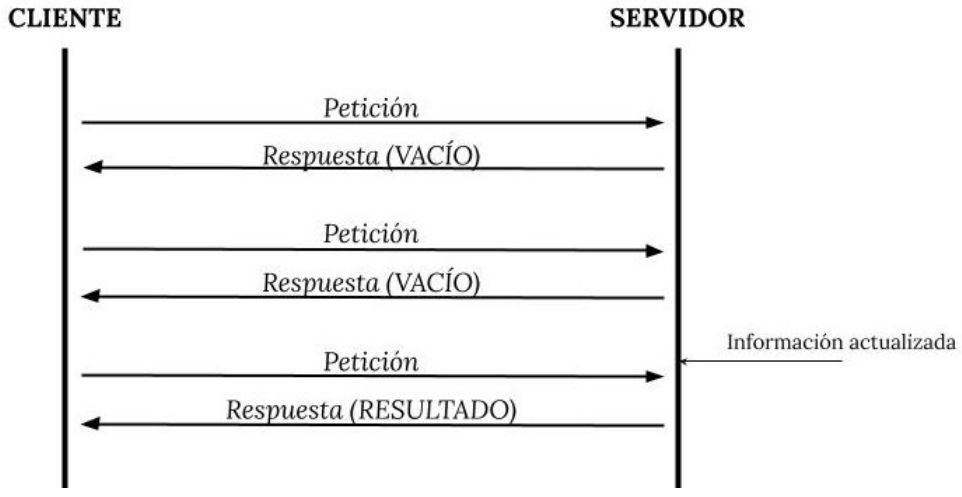

Figura 2.3: Polling.

La tecnología Long-Polling aparece para dar solución al envío de información innecesaria por parte del servidor (respuestas vacías), y por tanto del resto de peticiones de cliente y apertura y cierre de conexiones TCP que desencadena. El método Long-Polling permite al servidor mantener abierta la petición del cliente hasta que tenga una respuesta disponible, o pase un tiempo determinado, es decir, haya un *timeout* [13].

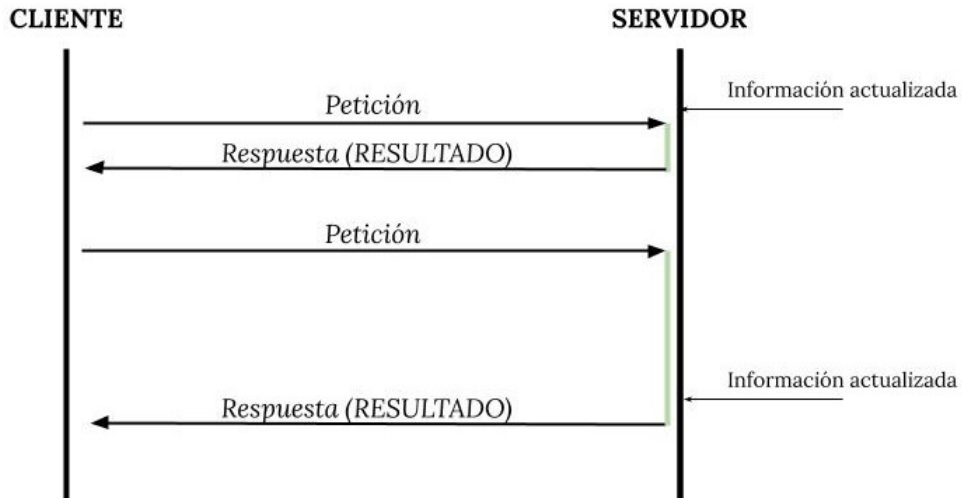

Figura 2.4: Long-Polling.

En la Figura 2.4 se observa cómo en la primera petición el servidor tiene información disponible para el cliente y por tanto, su respuesta es inmediata. Para la segunda petición, el servidor mantiene en espera al cliente y, en cuanto la información requerida es actualizada, la envía como respuesta.

El Long-Polling es una técnica sencilla de implementar y soluciona la sobrecarga del Polling, pero también tiene sus desventajas. Si el tráfico es intenso se puede encontrar latencia en la transmisión continua de datos al cliente, así como se puede generar una acumulación de procesamiento en la cola y en el servidor, implicando el uso intensivo de recursos y pérdidas en la información. Además, la configuración debe asegurar que las conexiones TCP no se queden abiertas indefinidamente, ya que eso puede dar lugar a reconexiones constantes. [14].

RestHooks

La tecnología RestHooks implementa el método PUB/SUB de manera que el cliente es notificado por el servidor en el momento en que ocurre un evento determinado.

RestHooks (RESTful WebHooks) es un conjunto de patrones que trata los *webhooks* como suscripciones, las cuales son manipuladas a través de un API RESTful [15]. Un webhook es una retrollamada HTTP, una solicitud HTTP POST en forma de notificación que se lanza cuando ocurre algo [16].

RestHooks tiene 4 requisitos básicos [17]:

- Mecanismo para almacenar suscripciones: las suscripciones deben contener cuatro elementos básicos: a qué suscribirse, el propietario de la suscripción, una URL

(webhook) a donde enviar la respuesta y, opcionalmente, el estado de la suscripción (activa o inactiva).

- Mecanismo para modificar suscripciones a través de un API: métodos CRUD (Create, Read, Update, Delete).
- Lista de tipos e implementaciones de eventos.
- Mecanismo de envío de hooks.

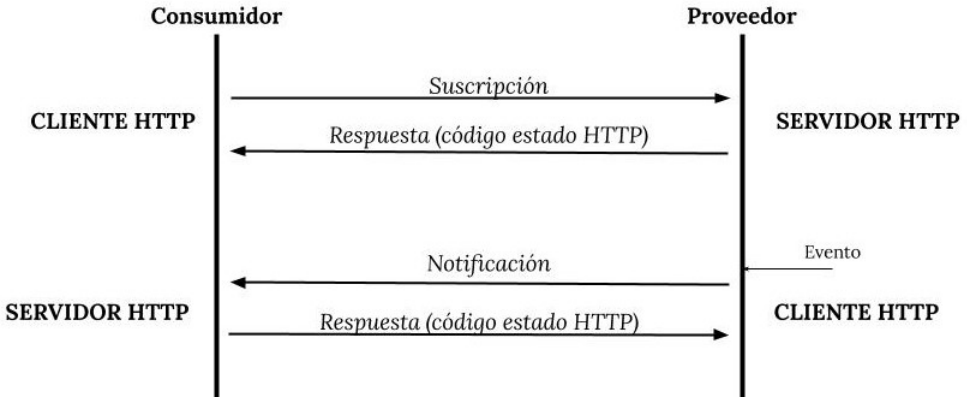

Figura 2.5: RestHooks.

En la Figura 2.5 se observa el intercambio de tramas entre el consumidor y el proveedor del servicio. Un requisito básico y la principal desventaja de RestHooks es que el consumidor debe disponer de un servidor HTTP. La necesidad de un *endpoint* por parte del consumidor se debe a que, aparte de recibir la respuesta a su petición, tiene la obligación de comunicar al proveedor el estado de la llegada de dicha notificación. Así, la respuesta en la segunda trama de la Figura 2.5 se refiere al estado de la suscripción mientras que la respuesta de la cuarta trama se refiere al estado de la notificación [18].

WebSockets

El protocolo WebSocket ha mejorado las posibilidades de comunicación sobre Internet y las ha proporcionado de un verdadero carácter de comunicación bidireccional en tiempo real.

El protocolo WebSocket habilita una comunicación bidireccional, *full-dúplex* y asíncrona entre un cliente y un servidor remoto, manteniendo una única conexión TCP, lo que elimina los problemas de latencia y sobrecarga en la red con el envío de cabeceras innecesarias. Se apoya en HTTP, ya que fue diseñado en sus inicios para páginas y aplicaciones web, aunque no se limita a dicho protocolo [19].

La comunicación Websocket establecida entre cliente y servidor se identifica a través de un websocket ID, de manera que dicha conexión es única e intransferible.

Un problema a tener en cuenta es el proceso de reconexión, ya que el protocolo no lo contempla, así que debe ser el cliente el que lo implemente de alguna manera. Pese a esta desventaja, los WebSockets son la mejor opción si se quiere diseñar un acceso asíncrono

a una plataforma web, ya que sólo se necesita una conexión TCP, el consumidor no debe tener un servidor HTTP y la sobrecarga es mucho menor, mejorando la velocidad y, en definitiva, aportando esa comunicación en tiempo real que se buscaba.

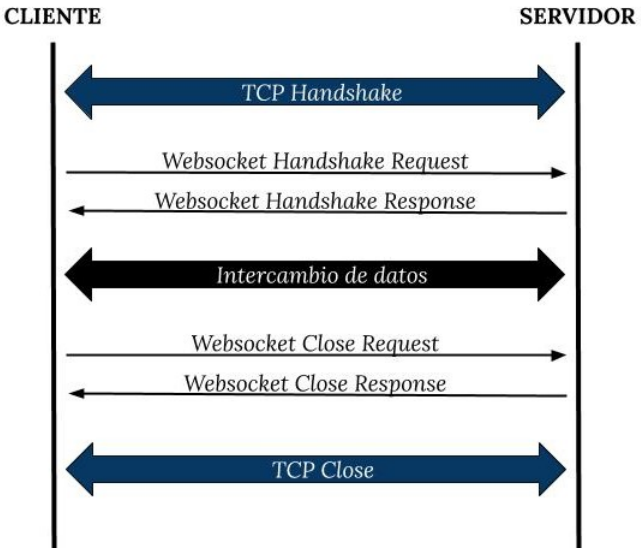

Figura 2.6: WebSockets.

En la Figura 2.6 se observa cómo la conexión TCP es válida para toda la comunicación WebSocket entre cliente y servidor, no sólo para una asociación petición-respuesta, como en las tecnologías explicadas anteriormente. Esta figura representa una conexión WebSocket genérica sin vinculación a HTTP.

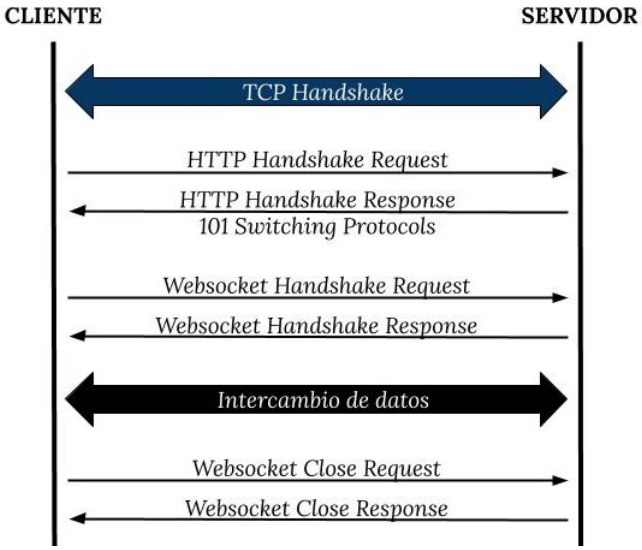

Figura 2.7: WebSockets con HTTP.

En la Figura 2.7 se observa el ejemplo de una conexión WebSocket sobre HTTP. En

primer lugar se realiza un *handshake* HTTP y, tras ello, se cambia de protocolo a WebSocket para poder establecer la conexión requerida.

La tecnología Websockets permite tanto la opción segura (Websockets Seguros, WSS), como no segura. Al igual que ocurre con HTTP, la seguridad se asegura mediane una sesión TLS (Transport Layer Security) a nivel de transporte. Es decir, al intercambio que se muestra en la Figura 2.6 y la Figura 2.7, simplemente habría que añadir el establecimiento de una sesión TLS tras la conexión TCP.

2.3. SmartSantander

2.3.1. Proyecto y despliegue físico

El proyecto SmartSantander tenía como objetivo la creación de un centro de pruebas Europeo para la investigación y experimentación de arquitecturas, nuevas tecnologías, servicios y aplicaciones para la IoT en el contexto de una Ciudad Inteligente. La plataforma SmartSantander está en constante evolución e incorporación de nuevos dispositivos. Pero este centro de pruebas no se queda en la experimentación de nuevas tecnologías y dispositivos, sino que además estudia el progreso e impacto socio-económico que genera la aceptación de la IoT, así como la respuesta de la ciudadanía a la implementación de los servicios ofrecidos [20].

Entre la gran cantidad de dispositivos IoT, SmartSantander dispone de alrededor de 3000 dispositivos IEEE 802.15.4, 200 módulos GPRS y unas 2000 etiquetas NFC y códigos QR colocados en diferentes zonas de la ciudad, así como unos 300 dispositivos colocados en vehículos como autobuses o taxis. El ciudadano es una parte muy importante del proyecto, tanto por su contribución como cliente de los servicios, como por su teléfono móvil, que también forma parte del recuento de los dispositivos IoT que forma parte del proyecto [20].

Con el fin de crear espacios y servicios útiles para la comunidad, los casos de uso implementados son [20]:

- Monitorización ambiental: los dispositivos se encargan de medir la cantidad de ruido, polución, tráfico y temperatura.
- Monitorización ambiental móvil: los dispositivos están instalados en vehículos como autobuses o taxis para contribuir a los datos recogidos de manera estática.
- Aparcamiento exterior: los dispositivos colocados bajo el asfalto de las plazas de aparcamiento son capaces de detectar la disponibilidad de aparcamiento en diferentes zonas de la ciudad.
- Guiado hacia aparcamientos libres: a lo largo de la ciudad se han colocado paneles informativos que indican dónde hay aparcamientos libres.
- Monitorización de la intensidad del tráfico: los dispositivos se encargan de medir el volumen del tráfico, la velocidad y la longitud de las retenciones de vehículos.
- Riego de parques y jardines: con el objetivo de hacer el riego mucho más eficiente, los dispositivos se encargan de medir la humedad y temperatura del suelo, las condiciones del viento o las precipitaciones.

- Etiquetas NFC/QR: estas etiquetas proveen de información a los usuarios que las lean con sus dispositivos móviles. Esta información puede ser sobre el horario de los autobuses o sobre algún monumento.
- Participación ciudadana: los teléfonos móviles servirán de dispositivo sensor al ciudadano, que podrá reportar eventos o incidencias que ocurran en la ciudad.

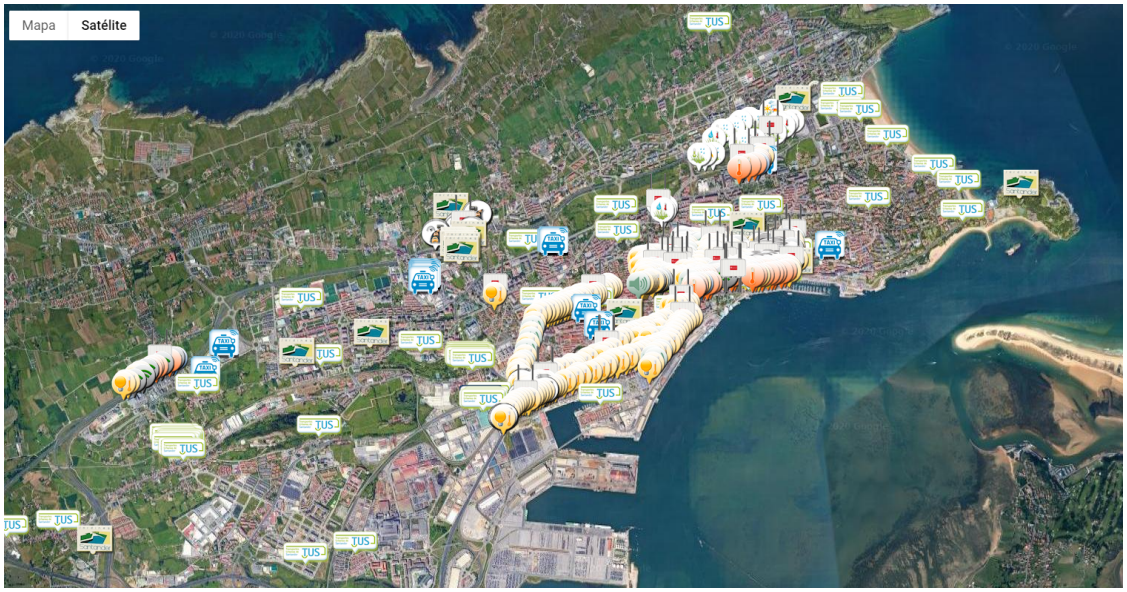

Figura 2.8: Despliegue de dispositivos IoT en la ciudad de Santander [22].

En la Figura 2.8 se observan los nodos desplegados por la ciudad de Santander. Los diferentes nodos se encargan de iluminación, temperatura ambiente y presencia de vehículos, entre otros.

La Figura 2.9 muestra la arquitectura de red del despliegue. En ella se resume la manera en la que los dispositivos se comunican entre ellos y con los servidores en la nube que soportan la plataforma IoT de SmartSantander.

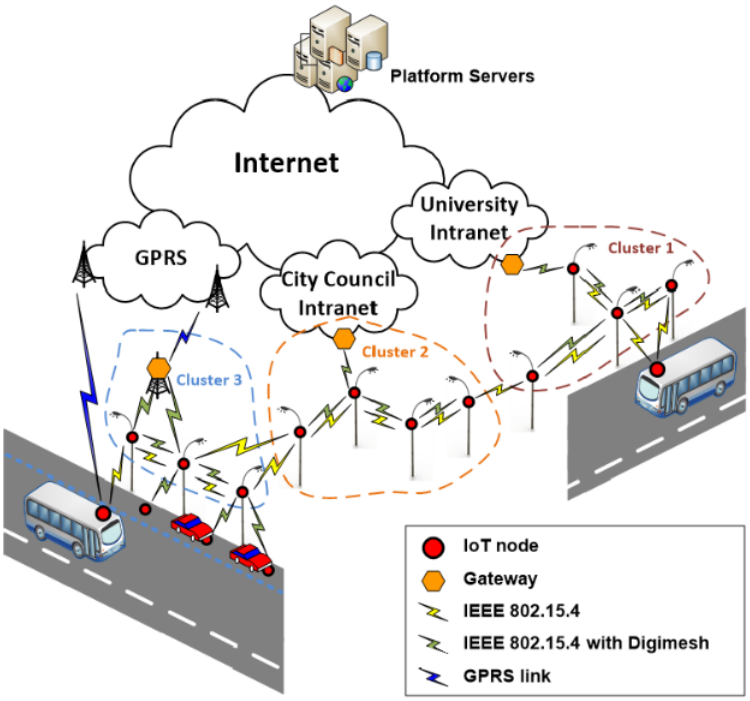

Figura 2.9: Diagrama del despliegue físico del centro de pruebas [4].

Los nodos estáticos están distribuidos en clústers que forman una red mallada de nodos que proporciona conectividad a un salto (a través de la interfaz nativa de IEEE 802.15.4) y transferencia de datos multisalto a la pasarela o *gateway* y al servidor (a través de la interfaz radio de Digimesh). Los dispositivos que no pertenezcan al mismo clúster, y por tanto, a la misma red mallada, no pueden compartir sus observaciones. Toda la información recogida por los dispositivos IoT es transmitida a través del *gateway* [4].

Los nodos móviles no forman parte de ningún clúster, sino que utilizan la tecnología GPRS para su conexión [4].

Los *gateways*, en la medida de lo posible, tienen acceso directo a una intranet cableada. En caso de que no sea así, se utiliza una conexión GPRS para conectar el *gateway* con la red troncal. Finalmente, los servidores de la plataforma SmartSantander están directamente conectados con la red troncal (Internet) [4].

2.3.2. Plataforma SmartSantander

Las plataformas IoT, sin exceptuar la de SmartSantander, tiene como uno de sus grandes objetivos ofrecer al consumidor los datos de manera eficiente sin perjudicar la usabilidad de la interfaz. Es por esto, que en SmartSantander se decidió proveer de dos tipos de acceso. Un acceso síncrono, para los datos históricos, y un acceso asíncrono para los datos en tiempo real.

Acceso síncrono a la información

El acceso síncrono dota al usuario de datos históricos generados por los dispositivos IoT desplegados por la ciudad en cualquier momento pasado. El acceso síncrono utiliza dos tipos de consultas: consulta en línea o *inline query* y consultas más complejas o *referenced queries* [20].

Una *inline query*, como su propio nombre indica, es una consulta en una sola línea, es decir, no tiene ninguna clase de complejidad pero la información obtenida va a ser limitada. Una *referenced query* o *complex query* es una consulta más grande y por tanto, la información obtenida va a ser mayor. Gracias a esto SmartSantander es apto para diferentes niveles de usuario [20].

Acceso asíncrono a la información

El acceso asíncrono provee al usuario de los datos en tiempo real medidos por los sensores. Para acceder a esta información el usuario debe enviar una suscripción que, en comparación con los tipos de consultas del acceso síncrono, es similar a la *complex query* en cuanto a complejidad y funcionalidad. Incluso en cuanto al formato, exceptuando que debido al carácter síncrono del enlace, la *complex query* no incluye los campos (*target* y *format*) que se refieren a la forma de recibir las notificaciones [20].

Las suscripciones le permiten al usuario configurar notificaciones asíncronas cuando una nueva observación o medida sea generada en la infraestructura IoT si coincide con lo requerido en la suscripción.

2.4. MQTT

2.4.1. Introducción al protocolo

Message Queuing Telemetry Transport (MQTT) es un protocolo de transporte de mensajes que implementa el método PUB/SUB y se basa en el modelo cliente-servidor. La arquitectura consta de diferentes elementos [23]:

- Topic: tema al que se realizan suscripciones y publicaciones de información. Por ejemplo: */dogs*.
- Clientes.
  - Cliente consumidor: es el cliente que se suscribe a determinada información en los *topic*.
  - Cliente proveedor: es el cliente que suministra/publica determinada información en los *topic*.
- Servidor: intermediario entre los dos tipos de clientes. Se encarga de procesar y manipular las suscripciones y la información de los *topics*. En MQTT se le denomina *broker*.
- Suscripción: un cliente consumidor se suscribe a un *topic* con el fin de recibir la información publicada en él.
- Sesión: conexión TCP establecida entre un cliente y el servidor. Se pueden realizar varias suscripciones en una misma sesión, así como publicaciones a diferentes *topics*.

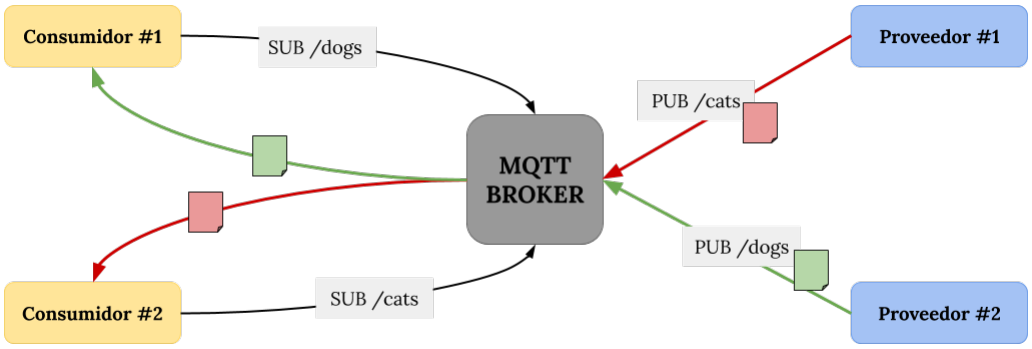

Figura 2.10: Arquitectura y ejemplo MQTT.

En la Figura 2.10 se observa un ejemplo sencillo de la arquitectura y su funcionamiento. El *Consumidor #1* ha establecido una conexión (sesión) con el broker y se ha suscrito al topic */dogs*. En el momento en el que algún otro cliente publica información en ese topic, el broker envía dicha información a todos los clientes suscritos a dicho topic. Es por eso que el *Consumidor #1* recibe la información verde y no la roja. Al tratarse de un ejemplo sencillo, cada cliente solo interactúa con un topic, pero en la realidad puede suscribirse a más de un topic y publicar en más de un topic (suscribirse y publicar -no en el mismo topic- no es excluyente).

Los topics se organizan de manera jerárquica, utilizando la barra diagonal (/) como separador [25]. Por ejemplo: `a/b/c/d`

Existen caracteres comodín, que se pueden utilizar por separado (en diferentes topics) o se pueden combinar:

- +: sólo para un nivel de la jerarquía
- #: para todos los niveles de la jerarquía que van por debajo.

Por ejemplo:
`a/b/+/d` ⇒ se obtendría información de todo el nivel *3*.
`a/#` ⇒ se obtendría la información de todo lo que esté por debajo del nivel de *a*.

El broker es el elemento central del protocolo, y por tanto, de su configuración dependen los niveles de seguridad implementados y de la calidad del servicio ofrecido.

MQTT ha sido ampliamente utilizado en entornos IoT por su sencillez y ligereza. Esto se traduce en un menor consumo de energía, lo cual es muy interesante para los dispositivos que funcionan constantemente y para los que son alimentados por batería. Otra consecuencia de su ligereza es la necesidad de un ancho de banda mínimo, lo que permite el uso de este protocolo en redes inalámbricas o en aquellas conexiones con problemas de calidad [24].

2.4.2. El broker Mosquitto

El broker Mosquitto es uno de los más utilizados y completos para la versión 3.1.1 de MQTT. El broker es el elemento central del protocolo y en él se puede configurar cómo, quiénes y hasta dónde entran/se conectan los clientes, es decir, los niveles de seguridad.

La configuración del broker se implementa en el archivo *mosquitto.conf* y tiene como parámetros importantes y que se han utilizado en la implementación [26]:

- *allow_anonymous true/false*: permite, en caso de que esté a TRUE, que se conecten usuarios sin autenticarse.
- *acl_file path*: define el directorio en el que se encuentra el archivo con la lista de control de acceso o Access Control List (ACL). Este archivo determina qué usuarios acceden a qué topics y con qué permisos (solo escritura, solo lectura o escritura y lectura).
- *password_file path*: define el directorio en el que se encuentra el archivo que registra las contraseñas de los usuarios. Sólo los usuarios cuyas contraseñas se encuentran en este archivo pueden conectarse al broker.
- *allow_zero_length_clientId true/false*: permite que el usuario que se conecte no disponga de un identificador de cliente.
- *use_username_as_clientId true/false*: utiliza el nombre de usuario del cliente como su identificador.
- *listener port*: define un puerto por el que escuchar.
- *cafile file_path*: el archivo con la cadena de certificación en la que confía el broker.
- *certfile file_path*: el certificado de servidor del broker.
- *keyfile file_path*: clave privada del broker.
- *requiere_certificate true/false*: determina si el usuario es autenticado vía certificado. En caso de que no sea especificado se entiende que no.
- *use_identity_as_username true/false*: determina que el nombre de usuario se corresponderá con el campo *Common Name* (CN) del certificado de cliente. En caso de que sea *true*, la autenticación por contraseñas no se aplicará.

En conclusión, utilizando los parámetros adecuados en el archivo de configuración es posible hacer que el broker incluya interfaces (*listeners*) con distintos niveles de acceso y/o con acceso a distintos topics. Por ejemplo un *listener* con un nivel mínimo de seguridad (dupla usuario-contraseña), otro en el que el servidor (broker) también se autentique mediante su certificado, y otro en el que la autenticación por certificados sea mutua (cliente-servidor). Asimismo, es importante configurar bien el *acl_file* para moderar el acceso de los usuarios a los topics y controlar que la información confidencial o no deseada no esté a su alcance.

Capítulo 3

Acceso asíncrono a la plataforma SmartSantander

En este capítulo se describen las características básicas del sistema que soporta el acceso asíncrono a la plataforma de SmartSantander. A través del API que provee este sistema es posible acceder a la información recogida por los sensores desplegados por la ciudad de Santander.

3.1. Arquitectura del sistema

SmartSantander ofrece un acceso asíncrono a los usuarios para poder acceder a la información emitida por los sensores en tiempo real. La arquitectura de este sistema es la detallada en la Figura 3.1.

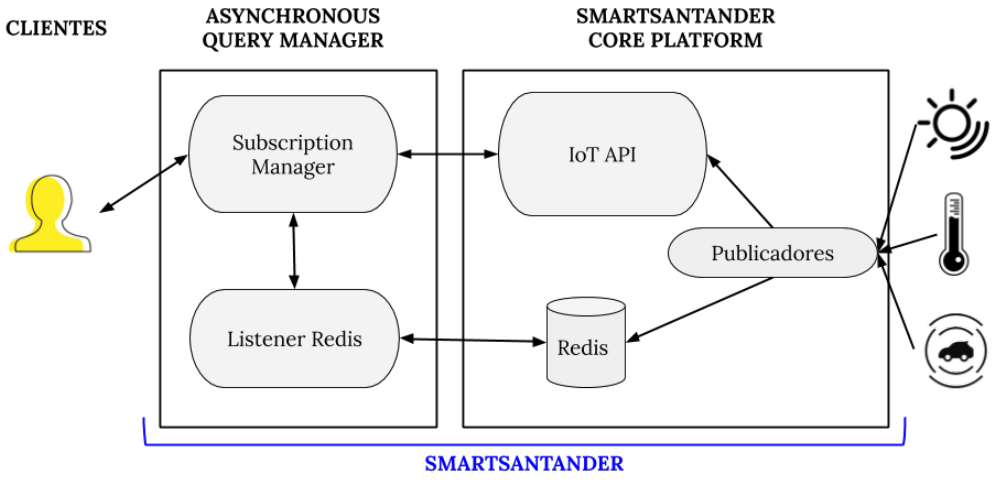

Figura 3.1: Arquitectura del sistema para acceso asíncrono.

Los componentes principales de la arquitectura son:

- Cliente: las aplicaciones que hacen uso de los servicios ofrecidos por SmartSantander toman el rol de cliente. Se pueden suscribir a uno o varios tipos de medidas con el fin

de ser notificado cuando se cumplan los parámetros especificados en la suscripción.

- Asynchronous Query Manager (AQM): este módulo puede incluir diferentes componentes según la tecnología utilizada para gestionar las notificaciones, pero que en su conjunto actuará como intermediario entre el cliente y la plataforma de SmartSantander. Se encargará de procesar las peticiones de los clientes y de distribuir las notificaciones asociadas a ellos.
- SmartSantander Core Platform (SCP): reune las funcionalidades principales de la plataforma IoT de SmartSantander. Entre todos sus componentes, los que principalmente interactuan con el Async Query Manager son el IoT API y la base de datos Redis encargada de la generación y envío de eventos relativos a las medidas en tiempo real.
- Sensores: los sensores reportan de manera periódica la información que recogen del ambiente.

Los módulos ya desarrollados e inherentes a la plataforma SmartSantander nativa son los sensores y la Core Platform. Los sensores se encargan de realizar medidas, en base a las cuales se realizan observaciones que son reportadas a los servidores centrales de la plataforma para su gestión y almacenamiento. La plataforma SmartSantander ya dispone de los mecanismos que habilitan el acceso asíncrono mediante el cual un cliente puede suscribirse y recibir notificaciones. En este proyecto se han desarrollado dos interfaces, basadas en los protocolos Websockets y MQTT, que hacen uso de esta capacidad y permiten ofrecer diferentes posibilidades para que las aplicaciones cliente obtengan las medidas y observaciones de los sensores en el momento en que éstas se producen. La implementación de los componentes que subyacen a estos interfaces se describirá en el siguiente capítulo.

El flujo de la comunicación comienza en el cliente, elemento más simple de toda la arquitectura. Establece una conexión con el AQM a través de la cual envía sus peticiones y recibe las respuestas y notificaciones pertinentes. Estas peticiones contienen la operación a realizar (crear, modificar, etc), las credenciales del cliente y, en caso de que sea necesario, el cuerpo de la suscripción o su identificador.

El AQM puede constar de más módulos de los representados en la Figura 3.1, dependiendo de la tecnología utilizada, pero los módulos *Subscription Manager* y *Listener Redis* son los básicos para gestionar un acceso asíncrono a la SCP.

El *Subscription Manager* recoge la información enviada por el cliente, la adapta al formato requerido por el IoT API y envía la suscripción formada para hacerla efectiva. Una vez recibe la respuesta a su petición se la devuelve al cliente. En este punto el cliente sabe el resultado de su operación (i.e. suscripción creada o modificada con éxito, fallo en la eliminación de una suscripción, etc.).

El *Listener Redis* se encarga de gestionar el envío de notificaciones al cliente. Este módulo se encuentra escuchando constantemente la información que llega a la base de datos de Redis, situada en la SCP, y se ocupa del control del envío de las notificaciones a todos los clientes reutilizando la tecnología (Websockets, MQTT, RestHooks, OML, etc.) que el cliente haya seleccionado al realizar su suscripción.

3.2. Gestión del ciclo de vida de una suscripción

La suscripción es el elemento más importante para el usuario ya que es su manera de comunicar sus preferencias. El cliente recibe una notificación cuando se cumplan las condiciones establecidas en la suscripción. Dependiendo de la tecnología utilizada, los componentes del AQM varían, pero en conjunto, será este módulo el que se encargue de adaptar la información recibida por el cliente.

Dentro de una suscripción hay dos campos principales: el *target* y la *query*. El *target* contiene la información sobre el cliente que solicita la suscripción, es decir, qué tecnología desea utilizar para recibir las notificaciones y, debido a su carácter asíncrono, un identificador de cliente para la gestión de las respuestas del IoT API y las notificaciones. Este campo es diferente para cada tecnología y por tanto, en el Capítulo 4 (Tablas 4.3 y 4.7) se detallará su formato y contenido para los interfaces Websockets y los interfaces MQTT.

La *query* contiene el cuerpo de la suscripción, lo que, en esencia, solicita el cliente. Consta de dos campos importantes: *what* o *qué* y *where* o *dónde*. El *what* contiene la información sobre qué tipo de suscripción es y a qué fenómeno/s y criterios se suscribe. El *where* contiene la especificación del área o los dispositivos IoT sobre los que quiere tener notificaciones (coordenadas, tamaño de la superficie, identificadores, etc.).

```
{
  "target": {...},
  "query": {
    "what": {
      "format": "measurement",
      "_anyOf": [{
        "phenomenon": "temperature:ambient",
        "filter": {
          "uom": "degreeCelsius",
          "value": {
            "_gt": 15
          }
        }
      }]
    },
    "where": {
      "_anyOf": [{
        "area": {
          "type": "Circle",
          "coordinates": [
            -3.810011,
            43.462403
          ],
          "radius": 1,
          "properties": {
            "radius_units": "km"
          }
        }
      }]
    }
  }
}
```

Figura 3.2: Detalle del objeto *query* de una suscripción.

En la Figura 3.2 se muestra un ejemplo del campo *query* de una suscripción en la que se está solicitando recibir notificaciones de medidas de temperatura ambiente en una determinada zona de un kilómetro de radie al rededor del punto especificado por las coordenadas facilitadas si se cumple que dicha temperatura es mayor de 15 grados Celsius. El atributo *format* permite seleccionar los dos tipos de datos que maneja la plataforma de SmartSantander, medidas (relativas a un único fenómeno físico) y observaciones (que agregan todas las medidas hechas por un mismo dispositivo en un determinado instante temporal).

```
{
  "target": {...},
  "query": {
    "what": {
      "format": "observation",
      "_anyOf": [{
        "phenomenon": "temperature:ambient",
        "filter": {
          "uom": "degreeCelsius",
          "value": {
            "_gt": 15
          }
        },
        "phenomenon": "temperature:ambient",
        "filter": {
          "uom": "degreeCelsius",
          "value": {
            "_lt": 25
          }
        }
      }]
    },
    "where": {
      "_anyOf": [{
        "area": {
          "type": "Circle",
          "coordinates": [
            -3.810011,
            43.462403
          ],
          "radius": 0.5,
          "properties": {
            "radius_units": "km"
          }
        }
      }]
    }
  }
}
```

Figura 3.3: Suscripción de observación.

Si la suscripción es una observación, la notificación recibida dependerá de que se incluya el atributo *_anyOf* o el atributo *_allOf*. En caso de que se use el primero, con que se cumpla una de las condiciones, se recibirá una notificación. Si se trata del segundo, han de cumplirse todas las condiciones.

Una propiedad importante de las suscripciones es el estado en el que se encuentran. Para ilustrar estos estados y sus posibles cambios el diagrama que se muestra en la Figura 3.4, resume el ciclo de vida de una suscripción. La Figura 3.4 también especifica las operaciones y parámetros asociados que fuerzan el paso de un estado a otro.

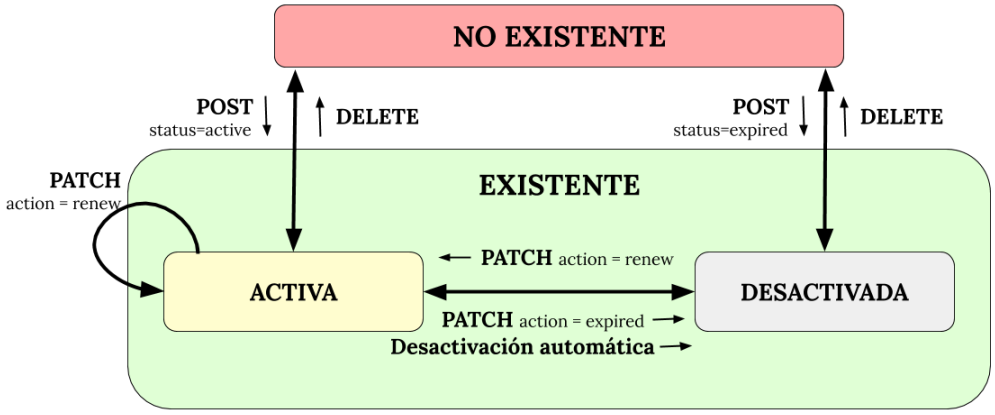

Figura 3.4: Ciclo de vida de una suscripción [27].

Una suscripción puede encontrarse en dos estados globales: existente o no existente. Dentro de existente, hay otros dos estados: activa o desactivada. Jugando con las operaciones que se pueden realizar a una suscripción, esta va variando entre todos los estados. Aún así, estos no son todos los métodos que soporta una suscripción. A continuación se detallan todas la operaciones soportadas por las suscripciones en la plataforma de SmartSantander.

3.2.1. Creación de una suscripción

La creación de una suscripción es el primer paso y la operación básica que debe llevar a cabo un cliente para poder recibir las notificaciones que le interesen.

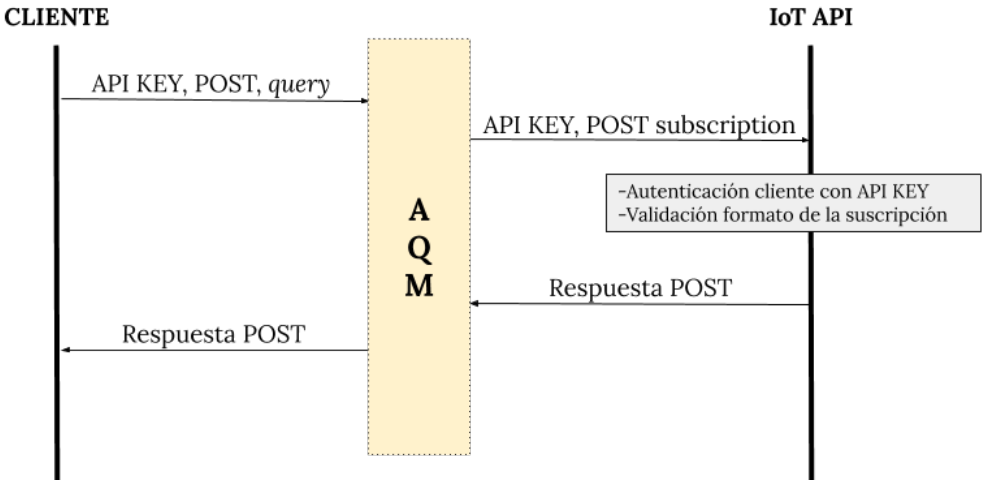

Figura 3.5: Proceso de creación de una suscripción.

El cliente debe enviarle al AQM tres parámetros en la petición:

- API KEY: credenciales de autenticación del cliente.
- Método=POST: comando que indica la solicitud de creación de una suscripción.

- *query*: cuerpo de la suscripción, donde indica a qué fenómeno/s y criterios se suscribe.

Una vez el AQM recibe esta petición se encarga de componer la suscripción, adaptándola al formato requerido por el IoT API (Figura 3.2 y Figura 3.3). El AQM debe añadir el campo *target* (Figura 4.3 y Figura 4.7), que depende de cada tecnología. Finalmente, realiza el envío de información al IoT API mediante una petición HTTP con método POST, incluyendo la suscripción como cuerpo de la petición y utilizando el API KEY del cliente para la autenticación.

El IoT API recibe la petición HTTP y realiza dos funciones: la autenticación del cliente con las credenciales aportadas y la validación del formato de la suscripción. Dependiendo del resultado de estas dos operaciones, el IoT API responde con diferentes código de estado y/o parámetros, que se muestran a continuación [28].

Código de respuesta	**Descripción**	**Extra**
201	La suscripción se ha creado correctamente y ha sido activada.	Añade a la respuesta el contenido de la suscripción creada (Figura 3.6).
400	Formato de suscripción inválido.	
401	Error en la autenticación.	
403	Permiso denegado para realizar la operación.	

Tabla 3.1: Respuestas del IoT API al método POST.

Como se observa en el objeto JSON mostrado en la Figura 3.6, el IoT API, una vez hace efectiva la suscripción, le añade los campos: *id*, *createdAt*, *updatedAt*, *status* y *expirationDate*. Estos campos suponen un identificador y sello temporal para la suscripción, además de marcar el límite de su periodo de activación.

```
{
  "createdAt": "2020-06-23T17:07:01.427Z",
  "updatedAt": "2020-06-23T17:07:01.427Z",
  "target": {...},
  "query": {
    "what": {
      "format": "measurement",
      "_anyOf": [{
        "phenomenon": "temperature:ambient",
        "filter": {
          "uom": "degreeCelsius",
          "value": {
            "_gt": 15
          }
        }
      }]
    },
    "where": {
      "_anyOf": [{
        "area": {
          "type": "Circle",
          "coordinates": [
            -3.810011,
            43.462403
          ],
          "radius": 1,
          "properties": {
            "radius_units": "km"
          }
        }
      }]
    }
  },
  "id": "nfctxx",
  "expirationDate": "2020-06-30T17:07:01.427Z",
  "status": "active"
}
```

Figura 3.6: Contenido de una suscripción.

3.2.2. Listado de suscripciones y obtención de una en concreto

SmartSantander da la posibilidad al usuario de tener múltiples suscripciones a la vez, y por ello debe ofrecer una manera de acceder a información sobre ellas que resulte útil para el cliente. La plataforma ofrece dos versiones de acceso al contenido de las suscripciones: un listado con todas las suscripciones asociadas a un cliente, divididas en activas y expiradas; y acceso a una única suscripción, a petición del cliente a través del identificador de la misma. Esta segunda opción resulta muy interesante para el cliente, ya que obtener todo el listado de las suscripciones puede generar una gran sobrecarga visual cuando únicamente se está interesado en una.

En la Figura 3.7 se detalla qué información debe enviar el cliente según el acceso que quiera realizar.

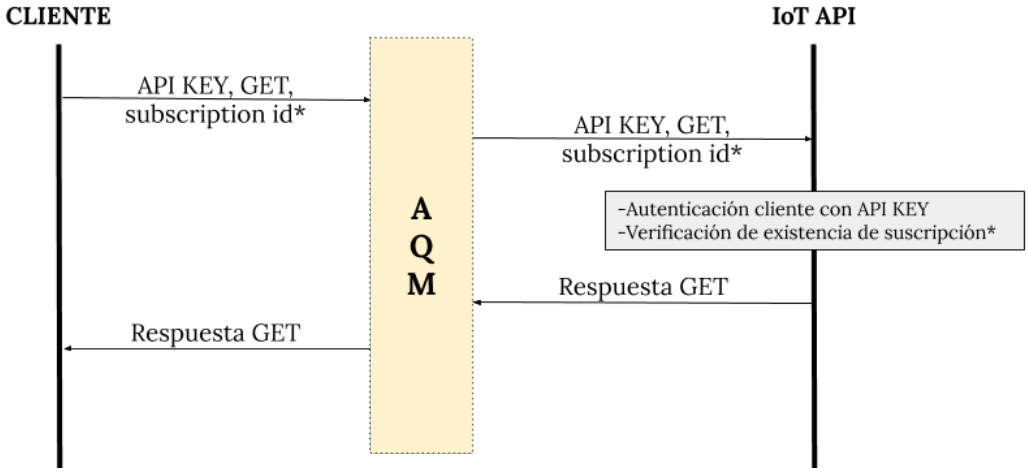

Figura 3.7: Proceso de listado u obtención de una única suscripción*.

Como se ha comentado, el usuario puede acceder al listado de sus suscripciones, diferenciadas según estén desactivadas o activas, así como a una suscripción en concreto. Para ello la información que debe hacer llegar al AQM es sencilla, dos ó tres parámetros:

- API KEY: credenciales de autenticación del cliente.
- Método=GET: este comando indica la solicitud de obtención del listado de suscripciones o de una determinada*.
- Subscription id*: identificador de la suscripción a la que se quiere acceder.

***** Opcional. Se usa en caso de querer acceder a una suscripción en concreto.

El AQM recoge la información recibida del cliente y realiza la petición HTTP GET al IoT API, incluyendo las credenciales del cliente, y en caso necesario, el identificador de suscripción como un *queryParameter* de la URL.

La plataforma de SmartSantander autentica al cliente y devuelve su respuesta, compuesta por un código de estado, y si es preciso, un *body*. En la siguiente tabla (Tabla 3.2) se observan las respuestas a la petición de listado de suscripciones [28]:

Código de respuesta	**Descripción**	**Extra**
200	El resultado solo incluye las suscripciones activas o expiradas.	Añade a la respuesta el listado de suscripciones (Figura 3.8).
401	Error en la autenticación.	
403	Permiso denegado para realizar la operación.	

Tabla 3.2: Respuestas del IoT API al método GET (listado).

```
{
  "active": [{
    "id": "nfctxx",
    "createdAt": "2020-06-23T17:16:21.022Z",
    "updatedAt": "2020-06-23T17:16:21.022Z",
    "target": {...},
    "query": {...}
  }],
  "expired": [{
    "id": "y71io9",
    "createdAt": "2020-06-23T17:17:36.167Z",
    "updatedAt": "2020-06-23T17:17:59.903Z",
    "target": {...},
    "query": {...}
  }]
}
```

Figura 3.8: Listado de suscripciones asociadas a un cliente.

En caso de que la petición realizada incluyese un identificador de suscripción, es decir, sólo se quisiese obtener información sobre una suscripción en concreto, las diferentes respuestas ofrecidas por el IoT API son [28]:

Código de respuesta	**Descripción**	**Extra**
200	-	Añade a la respuesta la descripción de la suscripción requerida (Figura 3.6).
401	Error en la autenticación.	
403	Permiso denegado para realizar la operación.	
404	La suscripción requerida no existe.	

Tabla 3.3: Respuestas del IoT API al método GET con identificador de suscripción.

3.2.3. Modificación de parámetros de una suscripción

Las necesidades e intereses de los humanos cambian, dando lugar a variaciones en investigaciones, proyectos y planes de servicios, entre otros. Es posible que, por ejemplo debido al cambio de estación, un cliente prefiera ser notificado cuando la temperatura aumente a partir de 22ºC en vez de 15ºC. O que el Ayuntamiento necesite un mayor control sobre el tráfico para poder moderar las congestiones, variando el límite de vehículos para ser notificado. Estos cambios de condiciones podrían solucionarse eliminando la suscripción existente y generando una nueva, pero la plataforma de SmartSantander ofrece la posibilidad de modificar la suscripción manteniendo el mismo identificador.

En la Figura 3.9 se observa el intercambio de mensajes y los procesos internos que se producen cuando el cliente solicita la modificación de una suscripción.

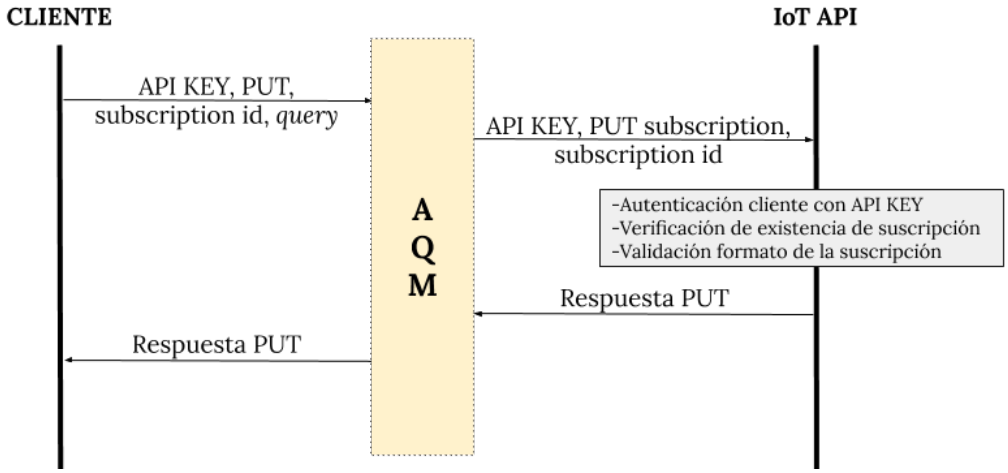

Figura 3.9: Proceso de modificación de suscripción.

La modificación de parámetros se da sobre una suscripción en concreto, y es por ello, que el cliente debe enviar los siguientes parámetros al AQM:

- API KEY: credenciales de autenticación del cliente.
- Método=PUT: este comando indica la solicitud de modificación de la suscripción.
- Subscription id: identificador de la suscripción que se quiere modificar.
- *query*: nuevo contenido de la suscripción a modificar.

Una vez el cliente envía esta petición, el AQM las recoge y conforma la suscripción, adaptándola al formato requerido por el IoT API, es decir, añadiendo el campo *target* según la tecnología aplicada (Figura 4.3 y Figura 4.7). A continuación, realiza la petición HTTP empleando el método PUT, incluyendo las credenciales del cliente, la suscripción como cuerpo de la petición y el identificador de suscripción como parámetro en la URI.

El IoT API de la plataforma SmartSantander autentica al cliente, verifica la existencia de la suscripción requerida, comprueba el formato de la suscripción y devuelve su respuesta, compuesta por un código de estado, y, si es preciso, un *body*. En la siguiente tabla (Tabla 3.4) se observan las respuestas a la petición de modificación de suscripción [28]:

Código de respuesta	**Descripción**	**Extra**
200	La suscripción ha sido modificada con éxito.	Añade a la respuesta el contenido de la suscripción modificada (Figura 3.6).
400	Formato de suscripción inválido.	
401	Error en la autenticación.	
403	Permiso denegado para realizar la operación.	
404	La suscripción requerida no existe.	

Tabla 3.4: Respuestas del IoT API al método PUT.

3.2.4. Eliminación de una suscripción

Los clientes pueden sufrir cambios en sus intereses y querer modificar ciertos parámetros de suscripciones, pero también puede que quieran simplemente deshacerse de una de ellas. La plataforma SmartSantander soporta la posibilidad de eliminar suscripciones a petición del usuario.

En la Figura 3.10 se observa el procedimiento que se lleva a cabo en la eliminación de una suscripción, desde el inicio en el cliente hasta su final con la respuesta del IoT API en el cliente también.

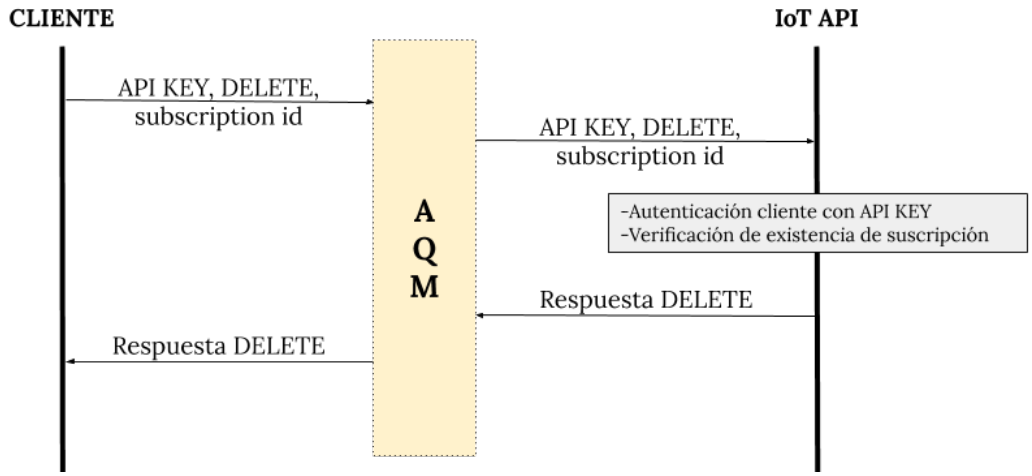

Figura 3.10: Proceso de eliminación de suscripción.

Para eliminar una suscripción el cliente debe enviar al AQM los siguientes parámetros en su petición:

- API KEY: credenciales de autenticación del cliente.
- Método=DELETE: este comando indica la solicitud de eliminación de la suscripción.
- Subscription id: identificador de la suscripción que se quiere eliminar.

El AQM recoge la información enviada por el cliente y realiza la petición al IoT API. Es una petición HTTP con método DELETE que incluye las credenciales del cliente y el identificador de la suscripción que se quiere eliminar como parámetro URI.

El IoT API de la plataforma SmartSantander autentica al cliente, verifica la existencia de la suscripción requerida y devuelve su respuesta, compuesta por un código de estado. En la siguiente tabla (Tabla 3.5) se observan las posibles respuestas a la petición de eliminación de una suscripción [28]:

Código de respuesta	Descripción
200	La suscripción ha sido eliminada con éxito.
401	Error en la autenticación.
403	Permiso denegado para realizar la operación.
404	La suscripción requerida no existe.

Tabla 3.5: Respuestas del IoT API al método DELETE.

3.2.5. Activación y desactivación de una suscripción

El estado de las suscripciones es un parámetro clave, determinando si la suscripción existente se encuentra activa o desactivada. La diferencia entre ambos estados es el hecho de que una suscripción activa generará notificaciones, mientras que una desactivada no. El estado desactivado puede confundirse con la eliminación de la suscripción, pero si se encuentra desactivada se queda registrada en los servidores de la plataforma. En cambio, los datos de una suscripción eliminada han sido borrados de dichos servidores.

El estado activo de las suscripciones en la plataforma de SmartSantander tiene un periodo de validez, por defecto, de una semana, pero el AQM se encarga de renovar automáticamente las suscripciones de clientes con una conexión establecida usando para ello credenciales de administrador, es decir, el cliente no participa en dicha renovación.

El estado desactivado se alcanza de dos maneras: por petición del cliente o por desconexión. El cliente puede realizar una petición de inhabilitación de una suscripción en concreto porque ya no le interese recibir más notificaciones asociadas a dicha suscripción. Con respecto a la desconexión, cuando el cliente se desconecta del AQM, todas sus suscripciones pasan a estado desactivado de manera automática.

El estado desactivado de una suscripción puede ser temporal mientras que el cliente siga conectado, ya que tiene la posibilidad de realizar una petición de activación de la suscripción y volver a recibir las notificaciones asociadas a ella.

La utilización de unas u otras credenciales (cliente o administrador) se debe a la necesidad de registrar todas las operaciones y así saber quién ha realizado qué, para, por ejemplo, saber el motivo de la expiración de una suscripción.

Por tanto, se tienen tres posibles escenarios, que se detallan en las Figuras 3.11, 3.12 y 3.13.

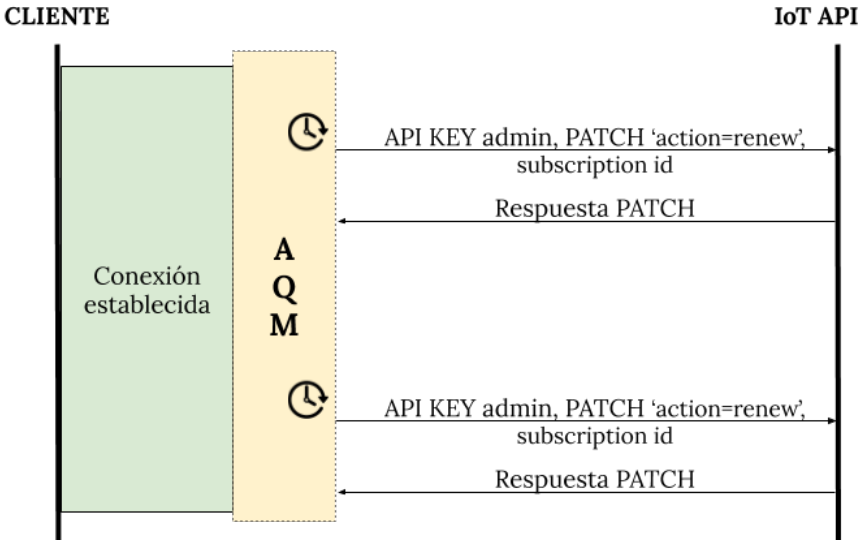

Figura 3.11: Renovación automática.

En la Figura 3.11 se observa como se renueva una suscripción cuando se cumple un periodo de tiempo (una semana). El AQM se encarga de gestionar la caducidad de cada una de las suscripciones activas y realiza la petición HTTP al IoT API por cada suscripción que esté próxima a expirar. Esta petición HTTP PATCH incluye un *queryParameter* cuyo valor será *renew*, las credenciales de administrador y el identificador de suscripción como parámetro en la URI.

El IoT API realiza la renovación de la suscripción y según el resultado de dicha operación, compone su respuesta con un código de estado, y en caso necesario, un campo *body*. Las posibles respuestas se recogen en la Tabla 3.6.

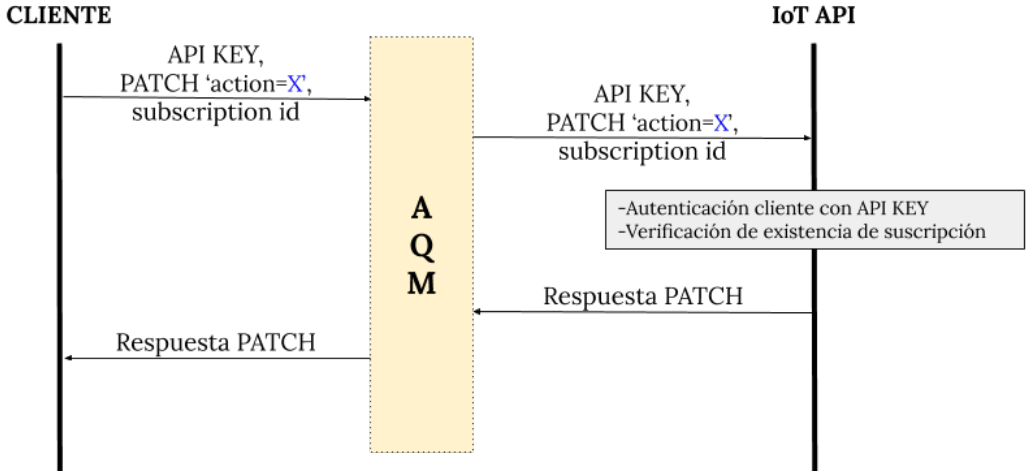

Figura 3.12: Proceso de inhabilitación de suscripción.

En la Figura 3.12 se observa el intercambio de mensajes desde la solicitud del cliente

con su petición de cambio de estado de una suscripción hasta que recibe la respuesta del IoT API. El cliente debe enviar al AQM cuatro parámetros:

- API KEY: credenciales de autenticación del cliente.
- Método=PATCH: este comando indica la modificación del estado de la suscripción.
- *queryParameter action*=X: debe tomar el valor de la acción a llevar a cabo, es decir, *renew* para renovar una suscripción o *expire* para su desactivación.
- Subscription id: identificador de la suscripción que se quiere expirar.

El AQM recibe la solicitud de modificación del estado de la suscripción por parte del cliente y realiza la petición HTTP PATCH con *queryParameter action*=X, incluyendo las credenciales del cliente y el identificador de suscripción como parámetro de la URI.

El IoT API de la plataforma SmartSantander realiza la autenticación del cliente en base a sus credenciales y verifica la existencia de la suscripción. Según el resultado de estas operaciones, la respuesta del IoT API será una de las mostradas en la Tabla 3.6.

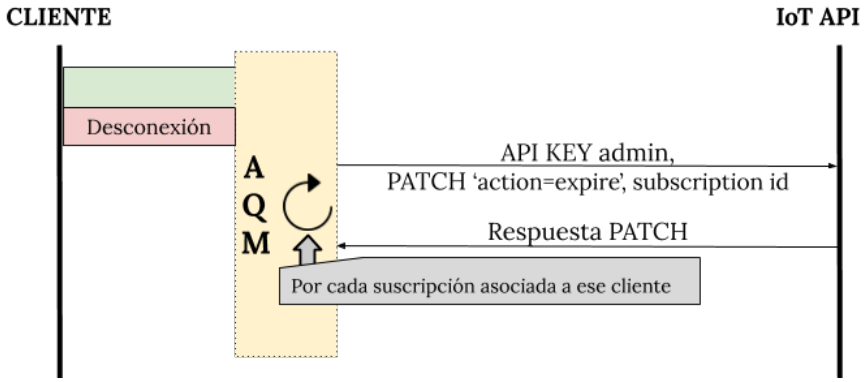

Figura 3.13: Proceso de desconexión del cliente.

En la Figura 3.13 se observa el proceso que ocurre entre el AQM y el IoT API cuando un cliente se desconecta dejando suscripciones activas. En este caso, para evitar que el sistema siga gestionando notificaciones que no podrán ser entregadas, el AQM debe enviar una petición de desactivación a la plataforma SmartSantander por cada suscripción asociada al cliente que se ha desconectado.

El IoT API inhabilita dichas suscripciones y según el resultado de esas operaciones, responde con alguna de las respuestas detalladas en la Tabla 3.6.

Código de respuesta	Descripción	Extra
200	El estado de la suscripción ha sido modificado con éxito.	Añade a la respuesta el contenido de la suscripción modificada (Figura 3.6).
401	Error en la autenticación.	
403	Permiso denegado para realizar la operación.	
404	La suscripción requerida no existe.	
409	Conflicto entre el estado actual de la suscripción y el estado que se requiere. No se puede expirar una suscripción que ya esta expirada.	

Tabla 3.6: Respuestas del IoT API al método PATCH.

3.3. Notificaciones

Los sensores son los encargados de generar medidas de determinados fenómenos físicos observados en la ciudad (temperatura, polución, ruido...) y publicar esas observaciones en la plataforma SmartSantander.

Estas observaciones se publican en una base de datos Redis, que debido a su arquitectura y funcionamiento, permite una gran cantidad de accesos y submódulos, denominados colas, para cada servicio.

Hay que tener en cuenta que cada uno de los dispositivos IoT desplegados puede contener varios sensores y es aquí donde radicará la diferencia en el formato de las notificaciones, detalladas en las Figuras 3.14 y 3.15.

```
{
  "subscriptionId": "nfctxx",
  "notification": {
    "value": 26.45,
    "timestamp": "2020-06-23T19:24:18.672085+02:00",
    "location": {
      "coordinates": [
        -3.81054,
        43.46368
      ],
    "type": "Point"
  },
  "phenomenon": "temperature:ambient",
  "urn": "urn:x-iot:smartsantander:u7jcfa:t373",
  "uom": "degreeCelsius"
  },
  "target": {...}
}
```

Figura 3.14: Formato de una notificación de una suscripción de medida única.

```
{
  "subscriptionId": "k558lo",
  "notification": {
    "urn": "urn:x-iot:smartsantander:u7jcfa:t575",
    "measurements": [{
      "phenomenon": "batteryLevel",
      "value": 59,
      "uom": "percent"
      },
      {
      "phenomenon": "temperature:ambient",
      "value": 24.19,
      "uom": "degreeCelsius"
      },
      {
      "phenomenon": "illuminance",
      "value": 0,
      "uom": "lux"
    }],
    "timestamp": "2020-06-23T19:24:16.057221+02:00",
    "location": {
      "coordinates": [
        -3.81214,
        43.4559
      ],
      "type": "Point"
    }
  },
  "target": {...}
}
```

Figura 3.15: Formato de una notificación de una suscripción de observación.

Cuando se trata de una observación se recoge la información sobre las medidas de todos los sensores que se generan en el mismo momento en el dispositivo al que pertenecen.

3.3.1. Envío de datos a REDIS

La plataforma SmartSantander utiliza una base de datos Redis para la publicación de las observaciones de los sensores. La gran cantidad de sensores que hay desplegado hace que se genere una gran cantidad de medidas en cortos periodos de tiempo, lo que puede suponer una gran sobrecarga en el sistema y, por tanto, antes de publicar las observaciones, estas pasan un control para asegurar que serán útiles para algún cliente. La arquitectura que gestiona la publicación de estas observaciones se muestra en la Figura 3.16.

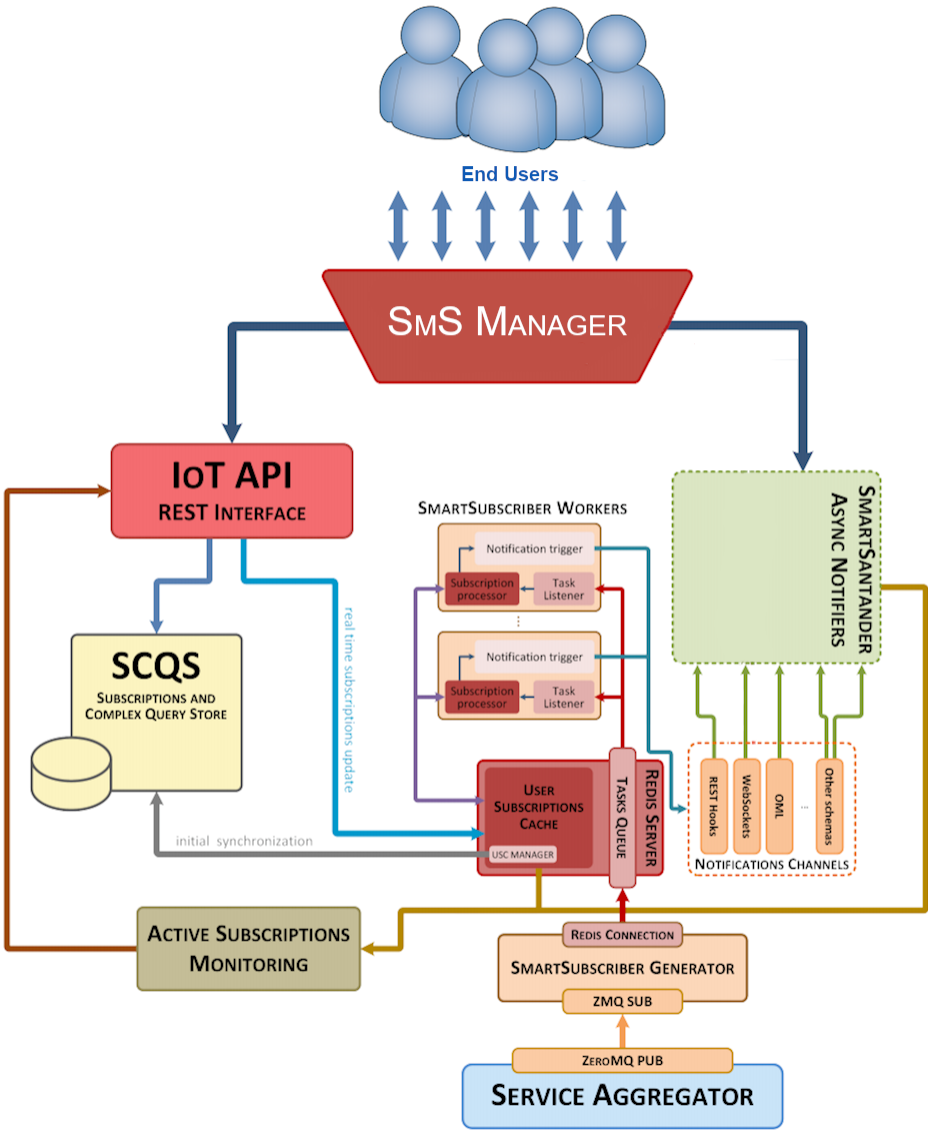

Figura 3.16: Sistema de publicación de observaciones de la Plataforma de SmartSantander.

Los clientes establecen una conexión bidireccional con el AQM que hace de intermediario con el IoT API para la publicación de suscripciones. La información que finalmente llega al IoT API se almacena en dos bases de datos sincronizadas: *SCQS* (Subscriptions and Complex Query Store) y Redis.

La base de datos *SCQS* se encarga de llevar un registro de todas las peticiones relativas a suscripciones realizadas al IoT API.

La base de datos Redis, denominada en la Figura 3.16 como *Redis Server* tiene una cola *Task Queue*, donde se almacenan todas las observaciones emitidas por los sensores. Como ya se ha comentado, las observaciones pasan por un control para no sobrecargar el sistema. Este control lo realizan los *SmartSubscriber Workers*, comparando las suscrip-

ciones activas con las observaciones recibidas. En caso de que exista una correspondencia, publicará dicha observación en la cola de la tecnología correspondiente en la base de datos *Redis SmS*. Será de esta última de la que el AQM obtiene las medidas que incluye en las notificaciones a los clientes.

3.3.2. Manejo de las notificaciones

La plataforma SmartSantander ofrece distintas opciones a través de diferentes tecnologías, para que los clientes puedan consumir las medidas generadas por los dispositivos IoT en tiempo real. Tecnologías como RestHooks u OML con las que ya contaba la plataforma antes del inicio de este poyecto, o Websockets y MQTT que son el resultado fundamental de este Trabajo de Fin de Grado (TFG).

Los servicios prestados conllevan notificaciones por parte de los sensores, y es en la base de datos *Redis SmS* en la que estas notificaciones finales son almacenadas.

Redis SmS consta de un sistema de colas y su funcionamiento se basa en la inserción y extracción de elementos entre ellas. Las colas son de tipo FIFO (First In First Out), es decir, lo primero que entra es lo primero que sale.

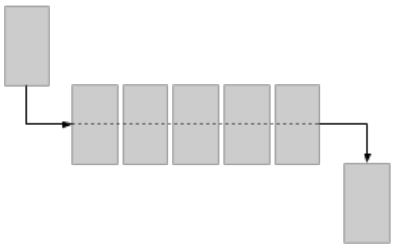

Figura 3.17: Cola FIFO.

Cada tecnología tiene sus propias colas. En el proyecto se han desarrollado interfaces para dos tencologías diferentes cuyos detalles se describirán en el siguiente capítulo.

En este apartado, para poder detallar la interacción entre las colas, se supondrá una tecnología genérica X. En el caso de la tecnología X se tienen las colas *x, x_processing y failed*. La cola *x* contiene las notificaciones que deben ser procesadas y enviadas a sus clientes, la cola *x_processing* contiene las notificaciones que están siendo procesadas y la cola *failed* contiene los identificadores de suscripción asociados a las notificaciones que no han podido ser entregadas.

Como se ha mostrado en la Figura 3.1, el AQM incluye un componente llamado *Listener Redis*, que se encuentra siempre escuchando a la base de datos *Redis SmS*, de manera que cuando una observación se encuentra en primera posición (cola FIFO) el *Listener Redis* hace una copia de ella en la cola *x_processing* mientras el AQM la procesa. La primera acción de dicho procesado es comprobar si los clientes asociados a esas notificaciones mantienen la conexión abierta.

En la Figura 3.18 se muestra el caso en el que el cliente sigue conectado.

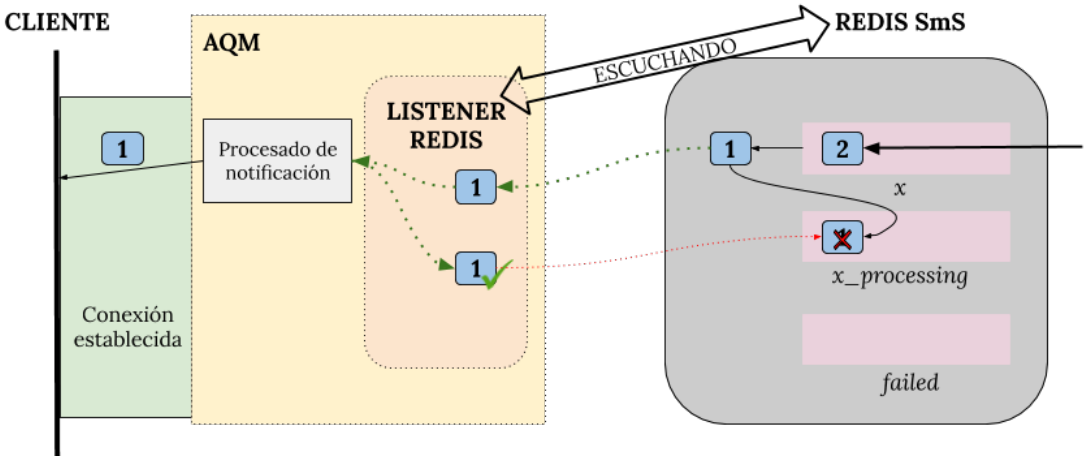

Figura 3.18: Procesado de notificaciones con éxito.

Se observa cómo el AQM entrega la notificación al cliente y comunica al Listener Redis este hecho. En ese momento, el Listener Redis elimina la observación de la cola *x_processing*, dando por finalizada dicha notificación.

En la Figura 3.19 se observa el caso contrario, donde el cliente se ha desconectado.

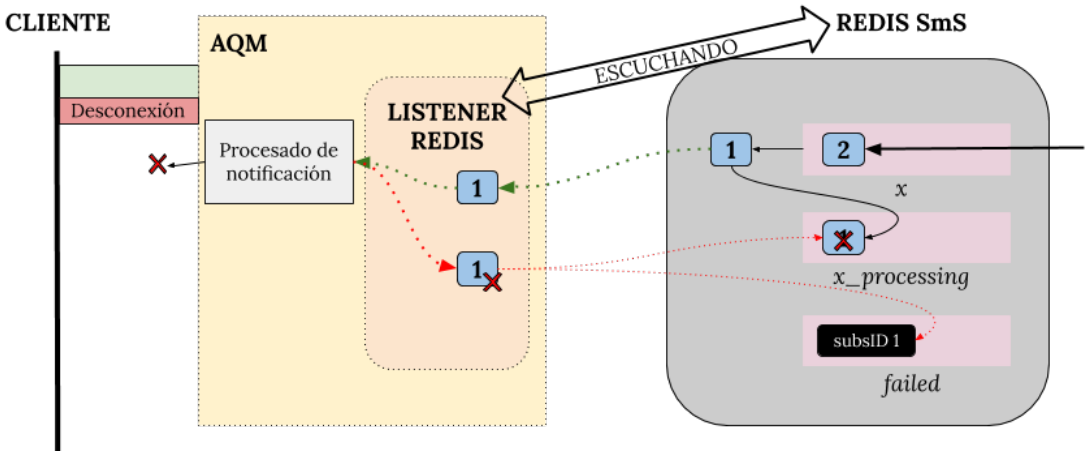

Figura 3.19: Procesado de notificaciones sin éxito.

Se observa cómo el AQM comunica al Listener Redis que la notificación no ha podido ser entregada y, por tanto, este coloca el identificador de la suscripción asociada a la notificación en la cola *failed*. Esto provoca un proceso interno en la plataforma SmartSantander que automáticamente expira dicha suscripción, para que dejen de recibirse notificaciones asociadas a ella.

Capítulo 4

Diseño e implementación del interfaz Websocket y del interfaz MQTT

En este capítulo se describen las diferencias en el desarollo e implementación de dos interfaces, basados en diferentes tecnologías (Websockets y MQTT), para ofrecer un acceso asíncrono a la información ofrecida por los sensores desplegados por la ciudad de Santander.

4.1. Interfaz Websocket

4.1.1. Arquitectura del sistema

El interfaz Websocket está desarollado sobre una arquitectura que consta de cuatro entidades: cliente, AQM, SmartSantander Core Platform y sensores. Es importante destacar que tanto el AQM como la SCP son parte de la Plataforma IoT de SmartSantander, pero mientras que la SCP es común a todas las tecnologías utilizadas para habilitar accesos asíncronos, hay un AQM específico por tecnología. En este sentido, la contribución de este proyecto ha sido precisamente el desarrollo de este AQM.

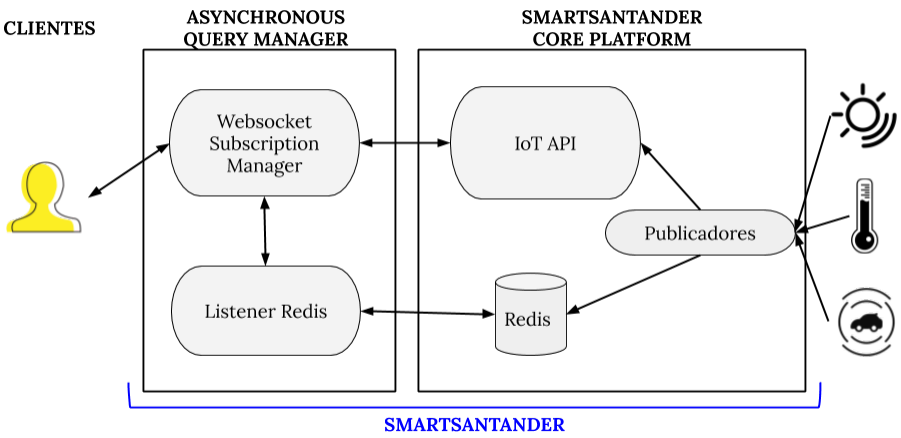

Figura 4.1: Arquitectura del sistema con interfaz websocket.

Comparando la Figura 3.1, donde se detallaba la arquitectura de un sistema general, con la Figura 4.1, se observa que el módulo AQM, en este caso, especifica que el Susbcription Manager es específico para el interfaz basado en Websockets.

El Websocket Subscription Manager (WSM) se encarga de la comunicación bidireccional, full-dúplex y asíncrona con el cliente, convirtiéndose en el elemento central. El *Listener Redis* se encarga, como se comentó en el anterior capítulo, de gestionar la comunicación entre el WSM y la base de datos Redis para la distribución de las notificaciones.

Por tanto, el interfaz Websocket utiliza el modelo de comunicación general explicado en el capítulo anterior.

En cuanto al manejo de las notificaciones, las colas que componen la base de datos *Redis SmS*, se denominan *wss*, *wss_processing* y *failed*, con *wss* atendiendo a WebSockets Seguros, que se explicará en la siguiente sección. En la Figura 4.2 se observa la disposición de estas colas.

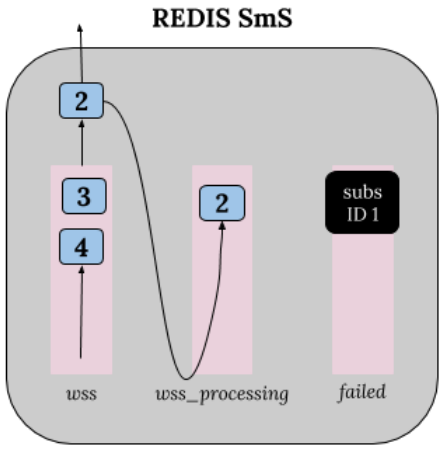

Figura 4.2: Sistema de colas Websocket.

Se ha representado un ejemplo en el tiempo donde la notificación asociada a la suscripción 1 no ha podido ser entregada y se está procesando la notificación asociada a la suscripción 2.

En cuanto al campo *target* de las suscripciones, específico para cada tecnología, si el cliente desea utilizar Websockets seguros para recibir las notificaciones, el contenido del objeto *target*, tal y como se muestra en la Figura 4.3, debe incluir el atributo *technology* (con valor *wss*) y un objeto *parameters* con el atributo *wsid* cuyo valor debe ser el identificador de la conexión Websockets establecida por el cliente. Este atributo permite el AQM enviar las notificaciones asociadas a cada suscripción de manera sencilla puesto que simplemente debe enviarlas a través del Websocket con ese identificador.

```
1
2 "target": {
3   "technology": "wss",
4   "parameters": {
5     "wsid": "GPb9oWzK0JjaI/5jS8K5Sw=="
6   }
7 }
```

Figura 4.3: Ejemplo de *target* para interfaz Websocket.

El ejemplo de la Figura 4.3 se corresponde a un *target* añadido a una suscripción del cliente cuyo websocket id es *GPb9oWzK0JjaI/5jS8K5Sw==*. Este campo va incluido en la suscripción final registrada en el IoT API y en las notificaciones. Al contener el identificador del cliente, es muy sencillo para el WSM devolver las respuestas del IoT API y distribuir las notificaciones a los clientes.

4.1.2. Seguridad en la conexión

La tecnología Websockets permite una comunicación bidireccional, full-dúplex y asíncrona entre cliente y servidor, que se genera después de establecer una conexión TCP, tal y como se ha descrito en el Capítulo 2. Esta comunicación se puede y debe securizar para proteger la información y asegurar la autenticidad de los extremos. La seguridad se garantiza a nivel de transporte mediante el protocolo TLS (Transport Layer Security), que genera un túnel dentro del cual la comunicación es protegida y los extremos son confiables.

El túnel TLS se establece en la fase de *handshake* con elección de *Cipher Suites* (algoritmos de intercambio de claves, de cifrado y de hash) y autenticación de los extremos. El servidor está obligado a autenticarse vía certificado y puede requerir o no que el cliente haga lo mismo. La configuración utilizada en el proyecto no obliga al cliente a presentar su certificado debido a que ya deberá autenticarse con sus credenciales (API KEY) frente a la plataforma SmartSantander.

De esta manera, el servidor presenta su certificado y el cliente verifica su validez y la confianza en la CA (Autoridad de Certificación) emisora. En caso de que esta operación finalice con éxito, se establecerá el túnel TLS y comenzará la comunicación Websocket, que será cifrada y segura.

4.2. Interfaz MQTT

4.2.1. Arquitectura del sistema

El interfaz MQTT está desarrollado sobre una arquitectura que consta de cinco módulos: consumidor, broker, proveedor, SmartSantander Core Platform y sensores, tal y como se puede ver en la Figura 4.4.

Al igual que con el desarrollo del interfaz Websocket el AQM es parte de la Plataforma SmartSantander, pero en la Figura 4.4 se ha distinguido en la arquitectura puesto que se trata del componenete que se ha desarrollado en el proyecto, mientras que el SCP ya estaba desarrollado.

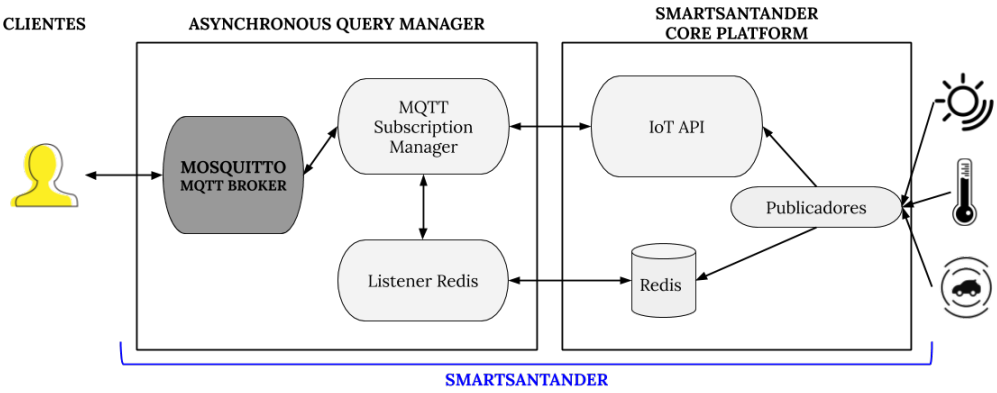

Figura 4.4: Arquitectura del sistema con interfaz MQTT.

Como se puede ver en la Figura 4.4, el AQM incluye un nuevo bloque funcional si lo comparamos con el diseño genérico recogido en la Figura 3.1, el MQTT Broker. Esto se debe al modo de funcionamiento del protocolo MQTT en el que la comunicación entre proveedores y consumidores se hace de manera intermediada por el broker.

En el Capítulo 2 se explicaron los fundamentos del protocolo MQTT, y en el Capítulo 3 la interacción entre el cliente y la plataforma SmartSantander para la obtención de datos recogidos por los sensores en tiempo real. Uniendo ambos aspectos, se ha desarrollado el interfaz MQTT y, a continuación, se expondrá cómo se han resuelto las operaciones del ciclo de vida de una suscripción y el proceso de notificación para observar su diseño e implementación.

MQTT se basa en el paradigma PUB/SUB y, por tanto, para generar la comunicación bidireccional, el cliente consumidor y el cliente proveedor deberán suscribirse a unos determinados topics y publicar en otros. En el caso del interfaz desarrollado, el rol de cliente consumidor lo toman las aplicaciones de usuario que van a recibir las medidas de los sensores. Por su parte, el MQTT Subscription Manager (MQTT-SM) actuará como cliente productor ya que será él quien obtenga esas medidas de la SCP y se las envíe a las aplicaciones.

Para el desarrollo del MQTT Broker, en el proyecto se ha empleado el Broker Mosquitto [26]. En la configuración del Broker se han definido los siguientes topics que se utilizarán para la interacción entre clientes y AQM:

- Topics de Operación: para cada una de las solicitdes de gestión del ciclo de vida de las suscripciones se ha creado un topic. De esta forma, los topics POST, GET, PUT, PATCH, DELETE, se utilizarán para publicar las peticiones de creación, listado, actualización y eliminación respectivamente.
- Topics de Resultado de Operación: las respuestas del IoT API a las peticiones de cada cliente se enviarán a un topic compuesto por el identificador del cliente y la palabra clave *feedback*. Por ejemplo, para el cliente con id 1234 las respuestas se publicarán en el topic "1234/feedback".
- Topics de Notificaciones: las notificaciones asociadas a cada suscripción se enviarán a un topic cpmpuesto por el identificador del cliente y el identificador de la suscripción. Por ejemplo, para el cliente con id 1234, las notificaciones de su suscripción con id 4321, se publicarán en el topic "1234/4321".

El cliente consumidor se suscribirá a "clientID/#", de manera que recibirá todo lo que se publique en los topics relacionado con su identificador de cliente, y publicará en el topic de operación que se corresponda con la petición que desea realizar, es decir, si quiere crear una suscripción, publicará su petición en el topic "POST".

A su vez, el cliente proveedor hará justo lo contrario, se suscribirá a los topics de los métodos para recibir las peticiones de los clientes y publicará en los topics asociados a los clientes las respuestas del IoT API o las notificaciones.

En la Figura 4.5 se observa una esquema de las suscripciones a los topics según el tipo de cliente.

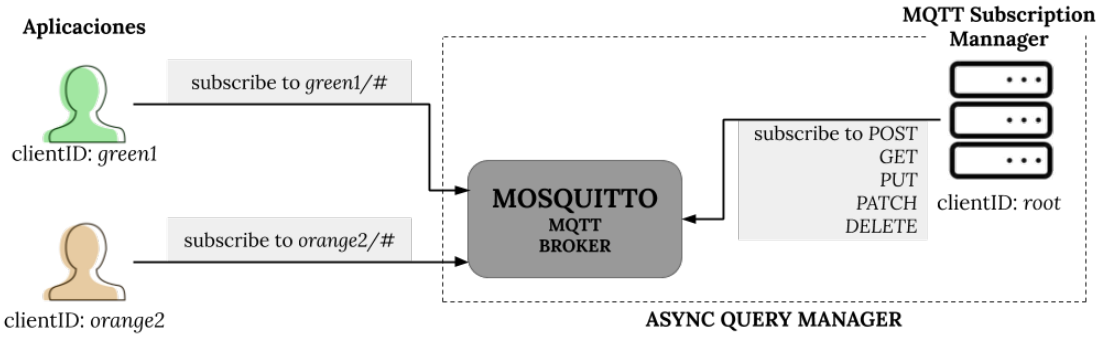

Figura 4.5: Esquema de suscripciones a topics.

Para detallar el diseño y la implementación del interfaz MQTT, se muestra en la Figura 4.6 la interacción entre cliente, broker Mosquitto y MQTT-SM para la creación de una suscripción en el IoT API de la plataforma SmartSantander.

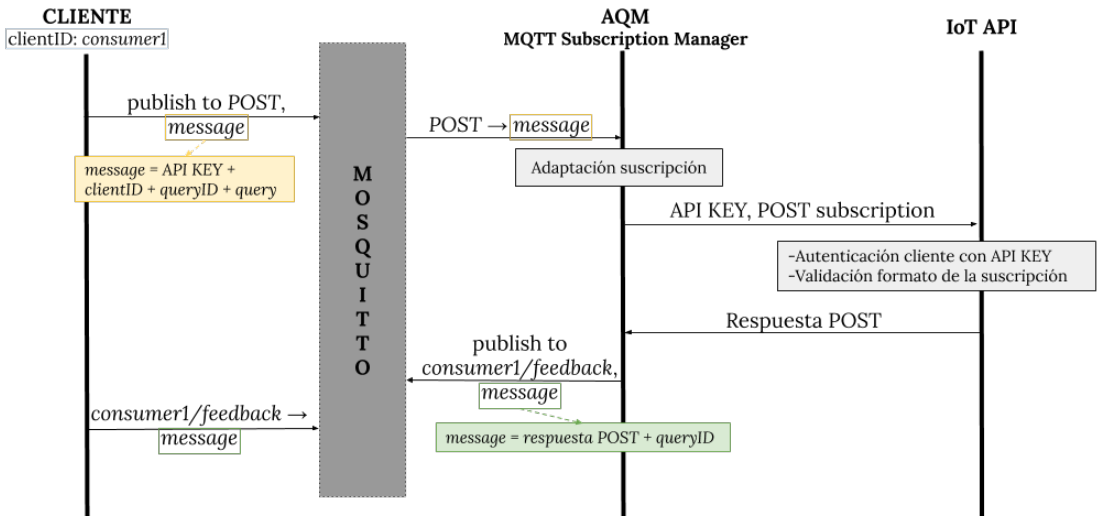

Figura 4.6: Creación de una suscripción a través del interfaz MQTT.

El cliente publica en el topic *POST* un mensaje con los parámetros necesarios para poder generar su suscripción en la plataforma Smartantander. Estos parámetros son:

- API KEY: credenciales de autenticación del cliente.
- clientID: le indica al AQM en qué topic debe publicar sus respuestas.
- queryID: para llevar un registro, por parte del cliente, de las peticiones que han obtenido respuesta.
- *query*: cuerpo de la suscripción, donde indica a qué fenómeno/s y criterios se suscribe.

El broker Mosquitto notifica al AQM que se ha realizado una publicación en el topic *POST* enviándole el contenido de la misma. El *MQTT Subscription Manager* se encarga de adaptar la suscripción al formato requerido por el IoT API, añadiendo el campo *target* específico de la tecnología MQTT mostrado en la Figura 4.7. El AQM realiza la petición y recibe la respuesta del IoT API, tras haber realizado sus operaciones internas, explicadas en el capítulo anterior.

El AQM publica en el topic *consumer1/feedback* un mensaje que contiene la respuesta de la plataforma SmartSantander y el identificador de la petición asociada a dicha respuesta. El broker Mosquitto notifica ahora al cliente que se ha producido una publicación en el topic con su identificador de cliente y le pasa su contenido. El cliente obtiene la respuesta del IoT API y registra que dicha petición ha recibido su respuesta.

```
1
2 "target": {
3   "technology": "mqtts",
4   "parameters": {
5     "mqttid": "consumer1",
6     "queryId": "query_1243"
7   }
8 }
```

Figura 4.7: Ejemplo de *target* para interfaz MQTT.

El parámetro *mqttid* permite al AQM determinar a qué cliente, y por tanto topic, le corresponde qué respuesta del IoT API que publicará en el topic generado a partir del valor de este atributo y la palabra clave *feedback*, o notificación, que publicará en el topic generado a partir del valor de ese atributo y el identificador de la suscripción a la que corresponde dicha notificación.

El parámetro *queryId* permite al cliente manetener un registro sobre qué peticiones esperan aún su respuesta.

El manejo de las notificaciones utilizando la tecnología MQTT varía ligeramente del esquema general en la interacción cliente-servidor, al igual que el nombre de las colas de la base de datos *Redis SmS*. En la Figura 4.8 se observa la disposición de las colas y la comunicación ofrecida por el interfaz MQTT.

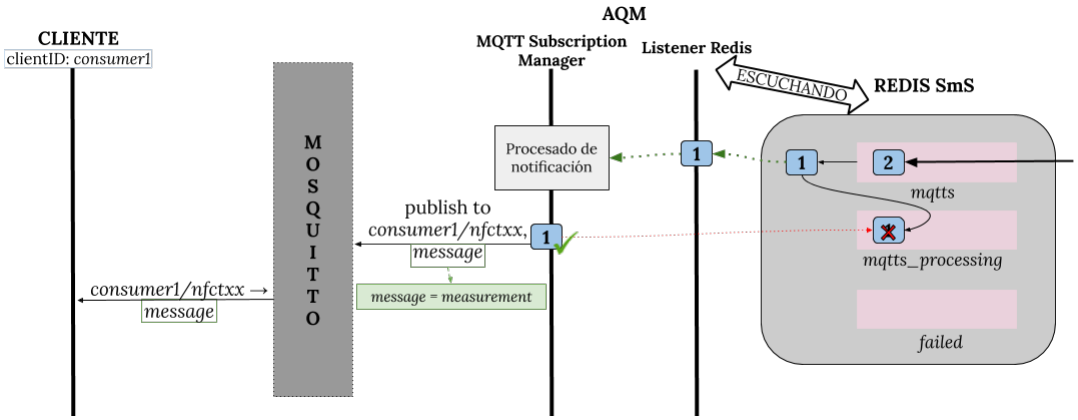

Figura 4.8: Gestión de notificaciones para el interfaz MQTT.

El AQM publica la notificación en el topic *consumer1/nfctxx*, indicando que la notificación es para el cliente con identificador *consumer1* y que está asociada a la suscripción *nfctxx*.

El broker Mosquitto le envía al cliente la publicación en ese topic, y así el cliente obtiene la notificación con una de las medidas u observaciones a las que se suscribió.

4.2.2. Seguridad en la conexión

La arquitecura MQTT ofrece diferentes opciones y niveles de seguridad, que se pueden configurar en el broker, tal y como se ve en la Figura 4.9, para asegurar la información y la autenticidad de los clientes, consumidores y proveedores, que se conecten a él.

La manera más sencilla para controlar la conexión de los clientes, *listener 1* en la Figura 4.9, es utilizar la dupla usuario-contraseña como método de autenticación. El broker Mosquitto mantiene registrada esta información en el archivo *passwordfile*, mostrado en la Figura 4.11, de los usuarios que pueden acceder, generando así un primer nivel de seguridad. Aún así, con este método la información no está protegida, es decir, viaja en claro entre cliente y Broker.

MQTT soporta la seguridad a nivel de transporte mediante el protocolo TLS, protegiendo la información entre cliente y Broker. TLS crea un túnel seguro para el intercambio de información.

El servidor determina si el cliente debe presentar su certificado, de manera que, en caso de que no lo requiera, el cliente se autentica a través de la dupla usuario-contraseña y verifica la validez y la confianza en la CA (Autoridad de Certificación) emisora del certificado del servidor, siendo esta opción la correspondiente con la configuración del *listener 2* en la Figura 4.9. En el supuesto de que el servidor requiera un certificado por parte del cliente, *listener 3* en la Figura 4.9, la verificación de la validez y confianza en la CA emisora es mutua.

El broker Mosquitto ofrece servicios de autorización basados en listas de control de acceso (ACL), archivo mostrado en la Figura 4.10, determinando qué clientes acceden a qué topics y con qué permisos.

```
#--------MOSQUITTO CONFIGURATION FILE--------

##COMMON PARAMETERS
acl_file /etc/mosquitto/conf.d/aclfile
password_file /etc/mosquitto/conf.d/passwordfile
allow_anonymous false
allow_zero_length_clientid true

##LISTENER 1 - UNSECURE
listener 1883
use_username_as_clientid true

##LISTENER 1 - UNSECURE - BROWSER
listener 9001
protocol websockets
use_username_as_clientid true

##LISTENER 2 - SECURE - SERVER CERT
listener 1884
use_username_as_clientid true
cafile /etc/mosquitto/ca_certificates/crt.pem
certfile /etc/mosquitto/ca_certificates/crt.pem
keyfile /etc/mosquitto/certs/key-nociphered.pem

##LISTENER 2 - SECURE - SERVER CERT - BROWSER
listener 9883
protocol websockets
use_username_as_clientid true
cafile /etc/mosquitto/ca_certificates/crt.pem
certfile /etc/mosquitto/ca_certificates/crt.pem
keyfile /etc/mosquitto/certs/key-nociphered.pem

##LISTENER 3 - SECURE - SERVER AND CLIENT CERTS
listener 1885
use_username_as_clientid true
cafile /etc/mosquitto/ca_certificates/crt.pem
certfile /etc/mosquitto/ca_certificates/crt.pem
keyfile /etc/mosquitto/certs/key-nociphered.pem
require_certificate true
use_identity_as_username true

##LISTENER 3 - SECURE - SERVER AND CLIENT CERT - BROWSER
listener 9003
protocol websockets
use_username_as_clientid true
cafile /etc/mosquitto/ca_certificates/crt.pem
certfile /etc/mosquitto/ca_certificates/crt.pem
keyfile /etc/mosquitto/certs/key-nociphered.pem
require_certificate true
use_identity_as_username true
```

Figura 4.9: Archivo de configuración *mosquitto.conf* del broker Mosquitto.

En la Figura 4.9 se muestra el archivo de configuración utilizado en el broker, resaltando las diferentes líneas utilizadas para la implementación de los distintos niveles de seguridad.

- *acl_file /etc/mosquitto/conf.d/aclfile*: define el directorio en el que se encuentra la lista de control de acceso (ACL), que en este caso es el archivo aclfile, que se muestra en la Figura 4.10.
- *password_file /etc/mosquitto/conf.d/passwordfile*: define el directorio en el que se encuentra el archivo que registra la dupla usuario-contraseña, que se muestra en la Figura 4.11.
- *allow_anonymous false*: no permite el acceso de clientes sin autenticar, lo que supone una primera barrera de seguridad.
- *allow_zero_length_clientid true*: permite que el cliente que se conecte no disponga de un clientid, ya que se autenticarán aportando un nombre de usuario.
- *listener 1883*: define el puerto, 1883 en este caso, por el que se escuchará con una determinada configuración, definida bajo esta línea. De esta manera, se implementan los diferentes niveles de seguridad.
- *use_username_as_clientid true*: convierte el nombre de usuario del cliente en su identificador.
- *protocol websockets*: especifica el protocolo (Websockets en este caso) que se aceptará en ese *listener*. Se habilitan *listeners* para el protocolo Websockets para realizar la validación funcional con clientes web.
- *cafile /etc/mosquitto/ca_certificates/crt.pem*: determina el archivo con la cadena de certificación en la que confía el Broker.
- *certfile /etc/mosquitto/ca_certificates/crt.pem*: determina el certificado de servidor del Broker.
- *keyfile /etc/mosquitto/ca_certificates/crt.pem*: determina el archivo con la clave privada del Broker.
- *require_certificate true*: el Broker exige al cliente que presente un certificado para su autenticación.
- *use_identity_as_username true*: esta línea se define cuando el Broker obliga al cliente a aportar su certificado, de manera que ahora su nombre de usuario será el campo *Common Name (CN)* del certificado. Además, dado que el cliente se autentica vía certificado, la línea *password_file* queda inhabilitada.

La lista de control de acceso (ACL) ofrece un servicio de autorización determinando qué clientes acceden a qué topis y con qué permismos. En la Figura 4.10 se detalla este archivo.

```
##-------- ACL FILE ---------
##All users
pattern read %u/#

##Client with username/clientId/Common Name from certificate
user MDkxY2EzZjQ0YzkzYmYwZmQwMjRlZTkzYmQ3Zjk5ODA6
topic write POST
topic write GET
topic write PUT
topic write DELETE
topic write PATCH
topic write CLOSE

##AQM -> root
user root
topic #
```

Figura 4.10: Lista de control de acceso (ACL) *aclfile* del broker Mosquitto.

En la Figura 4.10 se observa un extracto del archivo *aclfile*, donde se muestra la configuración para un cliente y el AQM.

La primera línea afecta a todos los usuarios y permite acceso de lectura al topic cuyo patrón se corresponde con el nombre de usuario y el caracter comodín #. Es decir, sólo el cliente 1234 tiene acceso de lectura al topic 1234/#. El siguiente bloque detalla el tipo de permiso que tiene el usuario definido en *user* en los topic de Operación. El cliente tiene únicamente permiso de escritura en los topics de Operación ya que publicará en ellos sus solicitudes de creación de suscripción, modificación, etc. En este caso, sólo se ha incluido un cliente, pero todos aquellos que tengan acceso al Broker seguirán este patrón, con permiso de escritura en los topics de Operación, y serán incluidos en el fichero. Por último, el AQM es un cliente especial y tiene acceso tanto de escritura como lectura a todos los topics del Broker. No tiene ningún tipo de restricción ya que es uno de los módulos desarrollados, y al fin y al cabo, su comportamiento está totalmente controlado.

En el fichero de configuración del Broker Mosquitto, en la Figura 4.9, se resaltaba una línea sobre un fichero que registra la dupla usuario-contraseña de los clientes autorizados a conectarse al Broker. Este fichero, *passwordfile*, se muestra en la siguiente figura.

```
MDkxY2EzZjQ0YzkzYmYwZmQwMjRlZTkzYmQ3Zjk5ODA6:$6$8iDeq9j+
    vLmOkmG9$mxQZtskqQIcH5AgJvro5X/VV/ORceaTGShgEORLbbHssTmHtigVMO2M+
    pD0gz3f/0FlavFQjB6dmI70vCtEcBA==
root:$6$owEgBf5C6dASY3Y5$7ad181G0vHOSRhGzBJzg95fiWZkpoV/aCG+
    fLzzpoPeEv5bzlAIOf8oLKVs6XIHM0Qb/kC0/Bz3ErzkSWchJXQ==
```

Figura 4.11: Archivo con el registro de las contraseñas de los clientes, *passwordfile*, del broker Mosquitto.

En la Figura 4.11 se muestra un extracto del fichero utilizado en el desarrollo, un cliente y el AQM. La estructura de este fichero es usuario:E(contraseña), es decir, el usuario aparece en claro, pero la contraseña es encriptada. La funcionalidad de este archivo es comprobar si el usuario que intenta conectarse está registrado, y en caso de que sí lo esté,

si lo hace con la contraseña correcta. Es el método de autenticación usado en los *listener 1 y 2*.

Tras detallar todas las configuraciones implementadas en el Broker, es importante comentar que el *listener 1*, tanto el usado por el cliente de línea de comando como el usado por el cliente web, deberán ser eliminados una vez el proyecto se integre en la plataforma de SmartSantander, ya que suponen un gran agujero de seguridad. Su único propósito era servir de ayuda para el desarrollo de la seguridad sobre una configuración base.

4.3. Desarrollo de una prueba de concepto

La contribución principal del proyecto es un desarrollo en el *backend* de la plataforma SmartSantander para proporcionar accesos asíncronos a través de dos tecnologías, Websockets y MQTT. Estos desarrollos consisten en varios módulos software que, en conjunto, proveen las funcionalidades requeridas. Sin embargo, tanto para validar funcionalmente estos desarrollos como para permitir una visualización de las funcionalidades, es necesario desarrollar la parte del *frontend*, que, en esencia, son los clientes que acceden a los servicios que provee la parte servidora desplegada.

Se han desarrollado dos tipos de clientes: el cliente de línea de comando y el cliente web con interfaz gráfica. Ambos validan las funcionalidades de los módulos desarrollados pero la gran diferencia se encuentra en la parte visual. El cliente de línea de comando es un cliente básico y por tanto, así lo es su presentación de la información, mientras que el cliente web es más complejo, constando de dos pestañas en el navegador para la realización de las peticiones y la presentación de respuestas y notificaciones.

4.3.1. Cliente de línea de comando

Las funcionalidades proporcionadas por los módulos desarrollados para los accesos asíncronos Websockets o MQTT son validadas, de manera visualmente básica, a través de clientes de línea de comando. Estos clientes consisten en sencillas aplicaciones que realizan un conjunto de peticiones y que, dependiendo de las respuestas permite comprobar el correcto funcionamiento del sistema.

Websockets

El cliente Websockets de línea de comando es un programa de Node.js relativamente sencillo. Al ser ejecutado establece la conexión Websocket con el servidor y comienza la comunicación.

En la Figura 4.12 se observa el terminal del cliente, y en la Figura 4.13 el terminal del AQM.

```
administrator@laura-martin:~/tfg$ node websockets/Secure/cliente-wss.js
OPTIONS:
POST -> make a subscription    ||    PUT -> edit a specific
    subscription    ||    GETunique -> get information about a specific
    subscription    ||    GETlist -> get information about all your
    subscriptions    ||    ACTIVE -> active a specific subscription    ||
       EXPIRE -> expire a specific subscription    ||    DELETE ->
    delete a subscription    ||    close -> close your connection

ID cliente: t++ypSPQqB2+xkmh/3fxXA==
Credentials: MDkxY2EzZjQ0YzkzYmYwZmQwMjRlZTkzYmQ3Zjk5ODA6

Request?: POST
Query file: query.json
LA SUSCRIPCIÓN m9lp9h SE HA CREADO CON ÉXITO Y HA SIDO ACTIVADA.
{...}


NOTIFICATION
SUSCRIPCIÓN: m9lp9h
{"subscriptionId":"m9lp9h","notification":{"value":24.96,"timestamp
    ":"2020-07-15T19:37:38.801618+02:00","location":{"coordinates
    ":[-3.81107,43.46369],"type":"Point"},"phenomenon":"temperature:
    ambient","urn":"urn:x-iot:smartsantander:u7jcfa:t376","uom":"
    degreeCelsius"},"target":{"parameters":{"wsid":"t++ypSPQqB2+xkmh/3
    fxXA=="},"technology":"wss"}}


Request?: close
```

Figura 4.12: Terminal de la aplicación cliente Websocket.

```
administrator@laura-martin:~/tfg$ node websockets/Secure/websocketServer-wss.js
Connecting to redis database
Waiting for notification from Redis list wss
C: Hola, soy un nuevo cliente y quiero conectarme
(node:3870) Warning: Setting the NODE_TLS_REJECT_UNAUTHORIZED
    environment variable to '0' makes TLS connections and HTTPS
    requests insecure by disabling certificate verification.
[1] Processing notification {"subscriptionId":"m9lp9h","notification
    ":{"value":24.96,"timestamp":"2020-07-15T19:37:38.801618+02:00","
    location":{"coordinates":[-3.81107,43.46369],"type":"Point"},"
    phenomenon":"temperature:ambient","urn":"urn:x-iot:smartsantander:
    u7jcfa:t376","uom":"degreeCelsius"},"target":{"parameters":{"wsid
    ":"t++ypSPQqB2+xkmh/3fxXA=="},"technology":"wss"}}
[1] Notification for subscription <m9lp9h> is being delivered
[1] Removing notification from processing queue
Waiting for notification from Redis list wss

****Se desconecta el cliente con id: t++ypSPQqB2+xkmh/3fxXA==****

ESTADO DE LA SUSCRIPCIÓN m9lp9h CAMBIADO CON ÉXITO A expire
```

Figura 4.13: Terminal del AQM Websocket.

La aplicación cliente, nada más ejecutarse, muestra una especie de menú que indica qué operaciones puede realizar y en qué consisten, y establece una conexión con el AMQ, que se encontrará activo. El AQM le proporciona un identificador de cliente cuando la conexión se ha establecido con éxito. Una vez que la aplicación cliente ha recibido su identificador se le pide las credenciales (API KEY) y que indique qué operación desea realizar con la pregunta *Request?:*. Según la petición elegida, se le realizarán otras preguntas, como el *query* de la suscripción o el identificador de suscripción. En el caso de la imagen, se realiza un POST y por tanto, el cliente aporta el archivo JSON con el *query* y se realiza la petición al AQM.

El AQM le devuelve a la aplicación cliente la respuesta de la IoT API, que en el caso del POST es el contenido de la suscripción creada, y en cuanto recibe una medida de la base de datos Redis Sms, la procesa y se la envía. La aplicación cliente muestra por pantalla la respuesta de la IoT API recibida a través del AQM y está lista para volver a realizar otra operación de las mostradas en el menú inicial. En el momento en el que recibe una notificación la muestra por pantalla, diferenciándola con líneas horizontales.

Existe una operación para el cliente, *close*, equivalente al Ctrl+Z, que detiene la ejecución de la aplicación cliente, desconectándose del websocket. El AQM, en este momento, muestra por pantalla qué cliente se ha desconectado y desactiva todas las suscripciones asociadas a este.

MQTT

El cliente MQTT, al igual que el cliente websocket, es un sencillo programa de Node.js. Al ejecutarse establece la conexión con el Broker, presentando su certificado o la dupla usuario-contraseña, según el nivel de seguridad que se desee usar, y se suscribe al topic asociado a su identificador de cliente.

En la Figura 4.14 se observa el terminal de la aplicación cliente y en la Figura 4.15 el terminal del AQM.

```
administrator@laura-martin:~/tfg$ node mqtt/Secure1/consumidorAPIKEY-mqtts1.js
OPTIONS:
POST -> make a subscription    ||    PUT -> edit a specific
    subscription    ||    GETunique -> get information about a specific
    subscription    ||    GETlist -> get information about all your
    subscriptions    ||    ACTIVE -> active a specific subscription    ||
       EXPIRE -> expire a specific subscription    ||    DELETE ->
    delete a subscription    ||    close -> close your connection

Me suscribo a MDkxY2EzZjQ0YzkzYmYwZmQwMjRlZTkzYmQ3Zjk5ODA6/#
Credentials: MDkxY2EzZjQ0YzkzYmYwZmQwMjRlZTkzYmQ3Zjk5ODA6

Request?: POST
Query file: query.json

TOPIC: MDkxY2EzZjQ0YzkzYmYwZmQwMjRlZTkzYmQ3Zjk5ODA6/feedback
LA SUSCRIPCIÓN 1k4c8l SE HA CREADO CON ÉXITO Y HA SIDO ACTIVADA.
{...}

TOPIC: MDkxY2EzZjQ0YzkzYmYwZmQwMjRlZTkzYmQ3Zjk5ODA6/1k4c8l
SUSCRIPCIÓN: 1k4c8l
{"subscriptionId":"1k4c8l","notification":{"value":20.58,"timestamp
    ":"2020-07-16T09:21:05.670721+02:00","location":{"coordinates
    ":[-3.80368,43.4622],"type":"Point"},"phenomenon":"temperature:
    ambient","urn":"urn:x-iot:smartsantander:u7jcfa:t250","uom":"
    degreeCelsius"},"target":{"parameters":{"mqttid":"
    MDkxY2EzZjQ0YzkzYmYwZmQwMjRlZTkzYmQ3Zjk5ODA6","queryId":"
    query_1d4dafde"},"technology":"mqtts"}}

Request?: close
```

Figura 4.14: Terminal de la aplicación cliente MQTT.

```
administrator@laura-martin:~/tfg$ node mqtt/Secure1/productor-mqtts1.js
Connecting to redis database
Waiting for notification from Redis list mqtts


TOPIC: POST
{"credentials":"MDkxY2EzZjQ0YzkzYmYwZmQwMjRlZTkzYmQ3Zjk5ODA6","
    clientId":"MDkxY2EzZjQ0YzkzYmYwZmQwMjRlZTkzYmQ3Zjk5ODA6","queryId
    ":"query_1d4dafde","body":{"query":{"what":{"format":"measurement
    ","_anyOf":[{"phenomenon":"temperature:ambient","filter":{"uom":"
    degreeCelsius","value":{"_gt":20}}}]},"where":{"_anyOf":[{"area
    ":{"type":"Circle","coordinates":[-3.810011,43.462403],"radius
    ":5,"properties":{"radius_units":"km"}}}]}}},"subscription":null}


(node:19579) Warning: Setting the NODE_TLS_REJECT_UNAUTHORIZED
    environment variable to '0' makes TLS connections and HTTPS
    requests insecure by disabling certificate verification.
[1] Processing notification {"subscriptionId":"lk4c8l","notification
    ":{"value":33.2,"timestamp":"2020-07-16T09:20:00+02:00","location
    ":{"coordinates":[-3.80947,43.4696],"type":"Point"},"phenomenon":"
    temperature:ambient","urn":"urn:x-iot:smartsantander:u7jcfa:f3115
    ","uom":"degreeCelsius"},"target":{"parameters":{"mqttid":"
    MDkxY2EzZjQ0YzkzYmYwZmQwMjRlZTkzYmQ3Zjk5ODA6","queryId":"
    query_1d4dafde"},"technology":"mqtts"}}
[1] Notification for subscription <lk4c8l> is being delivered
[1] Removing notification from processing queue
Waiting for notification from Redis list mqtts


TOPIC: CLOSE
{"credentials":null,"clientId":"
    MDkxY2EzZjQ0YzkzYmYwZmQwMjRlZTkzYmQ3Zjk5ODA6","queryId":null,"body
    ":null,"subscription":null}


ESTADO DE LA SUSCRIPCIÓN lk4c8l CAMBIADO CON ÉXITO A expire
```

Figura 4.15: Terminal del AQM MQTT.

Tanto la aplicación cliente como el AQM, al ejecutarse, establecen su conexión con el Broker, utilizando, en este caso, el *listener 2*, es decir, seguridad a nivel de transporte mediante el protocolo TLS con autenticación a través de la dupla usuario-contraseña. Una vez la conexión se ha establecido con éxito, la aplicación cliente y el AQM se suscriben a sus respectivos topics. El AQM se suscribe a los topics de Operación y el cliente a su topic personal (*clientId/#*).

Tras el correcto establecimiento de la conexión y la suscripción a determinados topics, el AQM, Figura 4.15, se encarga de sus funciones, esperando nuevos clientes, procesando peticiones y distribuyendo las medidas que recoge de la base de datos Redis Sms. La aplicación cliente, Figura 4.14, muestra por pantalla una especie de menú, al igual que el cliente Websockets, donde se detallan todas las operaciones que puede realizar y en qué consisten. Asimismo, le muestra al cliente a qué topic se suscribe y le pide las credenciales (API KEY).

En este momento la aplicación cliente ya está preparada para mandar peticiones. Dependiendo de la operación que quiera realizar, pedirá una serie de parámetros. En el ejemplo de la Figura 4.14 se solicita la creación de una suscripción, y por tanto, el cliente debe aportar el fichero JSON donde se encuentra el *query* de la suscripción a crear. La aplicación cliente publica en el topic *POST* su solicitud, y tras ello, muestra por pantalla de nuevo la posibilidad de realizar otra operación. El Broker notifica al AQM de la publicación que se ha realizado en el topic *POST*, y éste lo muestra por pantalla, tal y como se observa en la Figura 4.15.

El AQM procesa la solicitud del cliente y realiza la petición a la IoT API. La plataforma SmartSantander devuelve su respuesta al AQM, que lo publica en el topic compuesto por el identificador de cliente que realizó la petición y la palabra clave *feedback*. El Broker notifica a la aplicación cliente de la publicación en su topic y ésta la muestra por pantalla, indicando el topic y su contenido.

De manera paralela, el AQM procesa y distribuye las medidas que recibe por parte de la base de datos Redis Sms. El proceso de distribución consiste en publicar las medidas en los topics compuestos por el identificador de cliente y el identificador de suscripción asociados a dicha medida. El Broker notifica al cliente correspondiente de la publicación en ese topic, y la aplicación cliente muestra por pantalla tanto el topic como el contenido de la notificación.

Por último, el cliente realiza la operación *close* para notificar al AQM de que se va a desconectar. La publicación de la solicitud de desconexión en el topic *CLOSE* llega al AQM, que se encarga de desactivar todas las suscripciones asociadas a dicho cliente, mostrando por pantalla la notificación de publicación en el topic y el cambio de estado de las suscripciones.

4.3.2. Cliente web con interfaz gráfica

Un usuario real que demanda y utiliza los servicios ofrecidos por la plataforma SmartSantander es ajeno a la estructura interna e intercambios de mensajes subyacentes. Por ello, se ha implementado también una aplicación web que valida las funcionalidades de los desarrollos del proyecto y resulta más intuitiva y útil para un usuario final. Esta aplicación también resulta útil para demostrar todas las funcionalidades implementadas.

El cliente web muestra dos pestañas en el navegador. La primer pestaña, mostrada en la Figura 4.16, presenta un formulario que el usuario debe rellenar para conformar la petición al IoT API de SmartSantander.

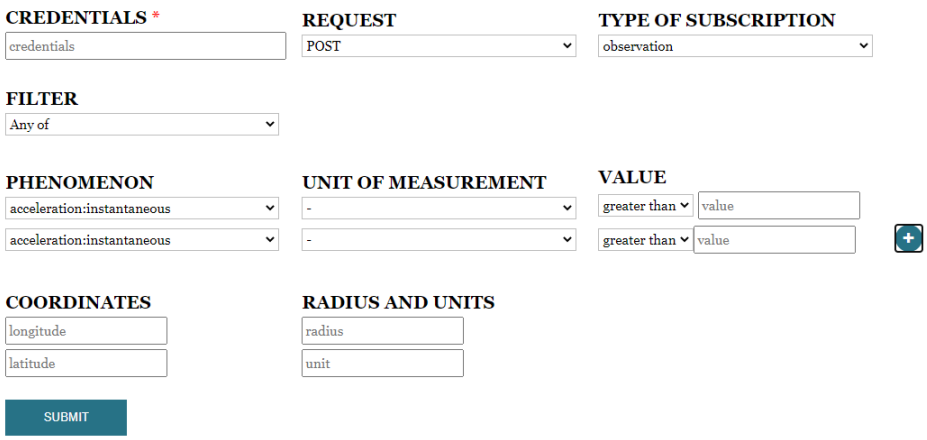

Figura 4.16: Formulario para el usuario.

El usuario debe introducir sus credenciales (API KEY), eligir el tipo de petición que desea realizar, completar el resto de campos de la suscripción y pulsar el botón de enviar.

En otra pestaña se encuentra el registro de respuestas y notificaciones asociadas a este cliente. Esta página presenta un menú con tres opciones: Respuestas, Notificaciones y Mapa. La primer opción incluye todas las respuestas recibidas por parte del IoT API a las peticiones del cliente, tal y como se observa en la Figura 4.17.

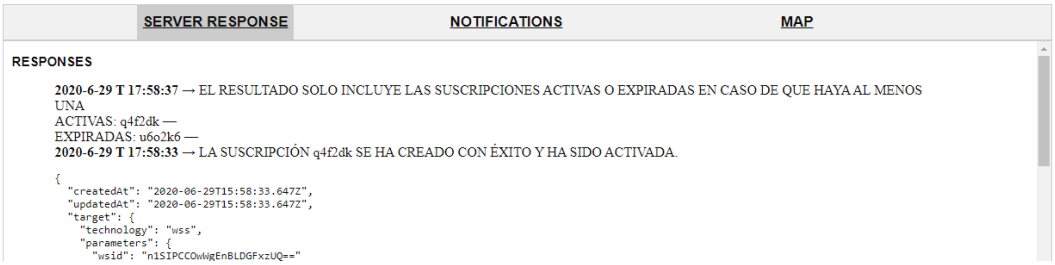

Figura 4.17: Registro de respuestas del IoT API.

De este modo, el usuario puede comprobar si se ha producido algún error en sus peticiones o si se han realizado correctamente, mostrando la respuesta devuelta por el IoT API.

La Figura 4.18 muestra como en el menú Notificaciones se presentan en texto plano todas las notificaciones obtenidas y a qué suscripción están asociadas.

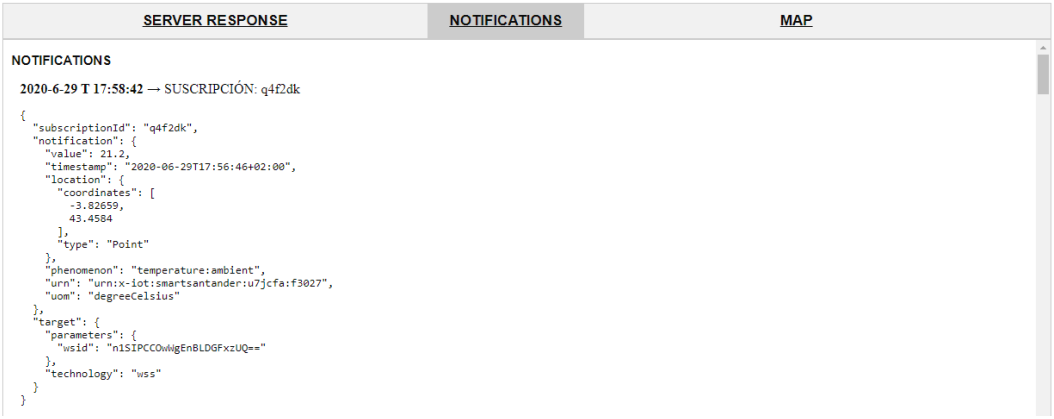

Figura 4.18: Registro de notificaciones.

Tanto el Menú de Respuestas como el de Notificaciones se ordena por antigüedad, es decir, las respuestas o notificaciones más recientes se muestran en la parte de arriba del registro, haciendo más fácil e intuitiva la estructura para el usuario.

Por último, el Menú de Mapa es el más visual para el cliente, ya que muestra un mapa de la ciudad de Santander en el que se van insertando chinchetas asociadas a las notificaciones recibidas. La Figura 4.19 muestra une ejemplo de esta opción.

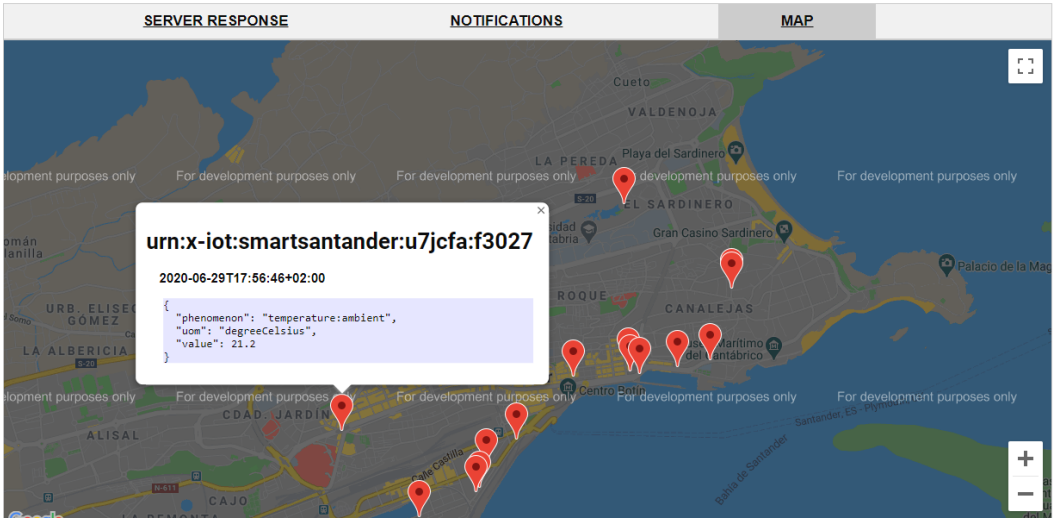

Figura 4.19: Mapa de Santander con el registro de notificaciones.

Capítulo 5

Conclusiones y líneas futuras

El propósito final del proyecto es dotar a la plataforma SmartSantander de diferentes maneras de soportar el acceso seguro y asíncrono a las observaciones y medidas generadas por los dispositivos IoT desplegados por la ciudad de Santander. Para ello, se han desarrollado dos interfaces con dos tecnologías diferentes (Websockets y MQTT) con las que gestionar dicho acceso y ofrecer al usuario dos maneras de acceder a los servicios, según su preferencia. En este capítulo se exponen las conclusiones extraídas en el desarrollo del trabajo, así como posibles ampliaciones y mejoras.

5.1. Conclusiones

SmartSantander se creó con objeto de ser un centro de pruebas para investigación y experimentación de dispositivos IoT en una Smart City. Esta plataforma estudia el avance de las tecnologías y su implementación en la ciudad, así como la aceptación de la ciudadanía a los servicios ofrecidos. Uno de los servicios más importantes es el acceso asíncrono a medidas y observaciones tomadas por los sensores distribuidos por la ciudad. El objetivo principal es ofrecer más tecnologías para el acceso asíncrono al cliente, de manera que elija según sus preferencias o necesidades.

Llegados a este punto, se puede concluir que todos los objetivos que se plantearon al inicio del proyecto se han alcanzado.

En primer lugar, se ha analizado en detalle el diseño del acceso asíncrono a las medidas generadas por la plataforma SmartSantander, así como los protocolos Websockets y MQTT. Este estudio ha permitido establecer una base sobre la que poder hacer el diseño de los nuevos interfaces que se perseguían en el TFG. Es importante destacar el reto que supuso analizar todas las características de la plataforma SmartSantander y los protocolos Websockets y MQTT, ya que se trata de tecnologías novedosas no estudiadas durante el Grado, y su perfecta comprensión es indispensable para poder acometer el diseño de los nuevos interfaces.

A continuación, se ha completado la especificación de los componentes necesarios para integrar las dos nuevas interfaces al sistema de suscripción de la plataforma SmartSantander. Para esta especificación, se realizó un diseño que fuera común para las dos tecnologías (Websockets y MQTT) y fácilmente integrable en la arquitectura de la plataforma SmartSantander. En este sentido, se ha especificado un componente denominado Asynchronous Query Manager (AQM), que actúa como intermediario entre el cliente y el IoT API de la

plataforma SmartSantander. Sobre la base de diseño común del AQM se han especificado, para cada tecnología, las funcionalidades que tenían que soportar cada uno de los módulos que componían el AQM específico para cada tecnología.

Una vez finalizada la fase de diseño, la implementación del AQM específico para cada protocolo (Websockets y MQTT) se ha llevado a cabo utilizando Node.js como lenguaje de programación. Aún siendo un lenguaje de programación que no había utilizado hasta ahora, lo cual incrementa el reto que supone el trabajo, la naturaleza asíncrona inherente a este lenguaje y el hecho de que buena parte de la plataforma SmartSantander esté desarrollada con este lenguaje, hace que su selección fuera la más apropiada para la implementación de las dos nuevas interfaces de la plataforma SmartSantander.

Por último, para cerrar el proyecto, se debían validar funcionalmente los módulos desarrollados. Para ello, se diseñaron e implementaron sencillas aplicaciones cliente, tanto por línea de comando como con interfaz web. El cliente web resulta más útil para demostrar todas las funcionalidades implementadas, además de ser un entorno mucho más visual e intuitivo.

A nivel académico, se ha conseguido el producto final deseado. Pero a nivel personal, todo el proyecto ha sido un aprendizaje: nuevas arquitecturas, nuevas tecnologías, nuevo lenguaje de programación, nuevos métodos y procedimientos y mejora en diseño e implementación.

5.2. Líneas futuras

Todos los objetivos planteados al principio se han ido completando a lo largo de su desarrollo. Aún así, para seguir completando y enriqueciendo el proyecto, a continuación se detallan posibles ampliaciones y líneas de trabajo futuras:

- Integrar los desarrollos en la instancia de producción de la plataforma. Destacar que los desarrollos se han hecho sobre una réplica de pruebas, que es exacta a la instancia de producción por lo que únicamente quedan los pasos de depuración del código, ejecución de test unitarios y de integración que permitan automatizar y simplificar el proceso, facilitando el mantenimiento futuro de los componentes software desarrollados, y despliegue en las máquinas de la instancia de producción de la plataforma de SmartSantander.
- Realizar pruebas de rendimiento y comparativa entre los distintos interfaces para poder tener un análisis cuantitativo además de cualitativo.
- Implementar nuevos interfaces utilizando protocolos como COAP o AMQP.

Bibliografía

[1] Wellington E.Webb, antiguo alcalde de Denver. *"The 19th century was a century of empires, the 20th century was a century of nation states, the 21th century will be a century of cities"*.

[2] Naciones Unidas, *https://www.un.org/development/desa/es/news/population/2018-world-urbanization-prospects.html*. 16 de Mayo de 2018.

[3] Bismart, *https://blog.bismart.com/es/top-7-de-smart-cities*. Último acceso 12 de Junio de 2020.

[4] Luis Sánchez, Luis Muñoz, José Antonio Galache, Pablo Sotres, Juan R. Santana, Verónica Gutiérrez, Rajiv Ramdhany, Alex Gluhak, Srdjan Krco, Evangelos Theodoridis y Dennis Pfisterer, *"SmartSantander: IoT Experimentation over a Smart City Testbed"*. Marzo 2014.

[5] SigFox España, *https://www.sigfox.es/*. Último acceso 13 de Julio de 2020.

[6] Lora Alliance, *https://lora-alliance.org/*. Último acceso 13 de Julio de 2020.

[7] María Gracia, Deloitte. *https://www2.deloitte.com/es/es/pages/technology/articles/IoT-internet-of-things.html*. Último acceso 12 de Junio de 2020.

[8] Universitat Oberta de Catalunya, *http://informatica.blogs.uoc.edu/2018/11/22/que-es-nb-iot/*. Noviembre de 2018.

[9] José Francisco Gay Medina, Universidad de Jaén. Trabajo de Fin de Grado, *"Análisis y despliegue de una plataforma IoT para el Smart Lab del CEATIC"*. Septiembre de 2018.

[10] Álvaro Cárdenas, Secmotic. *https://secmotic.com/plataforma-iot/*. Noviembre de 2016.

[11] RedHat, *https://www.redhat.com/es/topics/api/what-are-application-programming-interfaces*. Último acceso 15 de Junio de 2020.

[12] Pablo Martínez Pérez, Universidad de Cantabria. Trabajo de Fin de Grado, *"Aplicación móvil para un Sistema de Gestión Educativa"*. Febrero de 2014.

[13] Ably, *https://www.ably.io/concepts/long-polling*. Último acceso 17 de Junio de 2020.

[14] Mariano Furriel, *https://blog.intive-fdv.com.ar/websockets-vs-long-polling/*. Octubre 2016.

[15] RestHooks, Zapier. *http://resthooks.org/docs/*. Agosto 2013.

[16] Gabriela Gavrailova, *https://es.mailjet.com/blog/news/que-es-webhook/*. Mayo 2019.

[17] Stormi Hoebelheinrich, *https://www.olioapps.com/blog/rest-hooks/*. Noviembre 2019.

[18] Luis Sánchez, *"Servicios Web". Asignatura Protocolos y Servicios para Redes de Nueva Generación*. Último acceso 17 de Junio de 2020.

[19] IETF, *"The WebSocket Protocol". RFC 6455*. Diciembre 2011.

[20] Jorge Lanza, Pablo Sotres, Luis Sánchez, Jose Antonio Galache, Juan Ramón Santana, Verónica Gutiérrez y Luis Muñoz. *"Managing Large Amounts of Data Generated ny a Smart City Internet of Things Deployment"*. Diciembre 2016.

[21] Leo Turi, Universidad de Padova, Trabajo de Fin de Master. *"Contribution to the Federation of asynchronous SmartSantander service layer within the European Fed4FIRE context"*. Octubre 2015.

[22] SmartSantander, Google Maps. *http://maps.smartsantander.eu/*. Último acceso 18 de Junio de 2020.

[23] OASIS, *"MQTT Version 3.1.1 Plus Errata 01"*. Diciembre 2015.

[24] Luis Llamas, *https://www.luisllamas.es/que-es-mqtt-su-importancia-como-protocolo-iot/*. Abril 2019.

[25] Roger Light, *https://mosquitto.org/man/mqtt-7.html*. Último acceso 18 de Junio de 2020.

[26] Roger Light, *https://mosquitto.org/man/mosquitto-conf-5.html*. Último acceso 18 de Junio de 2020.

[27] Pablo Sotres, *SmartSantander. Experimentation on a real smart city deployment.* Fed4FIRE-GENI Research Experiment Summit. Ghent, Julio 2016.

[28] SmartSantander, *https://api-dev.smartsantander.eu:10443/docs*. Último acceso 22 de Junio de 2020.