UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

Grado en Ingeniería Informática

Integración de dispositivos IoT para el desarrollo de Edificios Inteligentes.

IoT Devices' integration for Smart Buildings.

Realizado por
Alejandro José Ruzafa Casás

Tutorizado por
Mónica Pinto Alarcón
Vicente Jesús Benjumea García

Departamento
Lenguajes y Ciencias de la Computación
UNIVERSIDAD DE MÁLAGA

MÁLAGA, febrero de 2024

Resumen

0.1. Español

IoT representa una red de objetos físicos que se integran con electrónica, software, sensores y conectividad de red, lo que posibilita la recopilación y el intercambio de datos entre estos dispositivos. Es una tecnología emergente que está experimentando un crecimiento vertiginoso y, al ser relativamente nueva, está acompañada de desafíos, incógnitas y beneficios. En este proyecto de fin de grado, se explorará la viabilidad de implementar soluciones concretas de IoT utilizando la plataforma de FIWARE, analizando las herramientas que ofrece, las estrategias de las que dispone para mitigar posibles inconvenientes y cómo aprovecha las ventajas inherentes al IoT. Además, evaluaremos cómo la resolución de ciertos problemas o la explotación de soluciones complejas podrían generar dificultades inesperadas en otros aspectos del sistema.

Palabras clave: IoT, FIWARE, Edificios Inteligentes

0.2. English

IoT represents a network of physical objects that are integrated with electronics, software, sensors and network connectivity, enabling the collection and sharing of data between these devices. It is an emerging technology that is experiencing rapid growth and, being relatively new, is accompanied by challenges, unknowns and benefits. In this final degree project, we will explore the feasibility of implementing concrete IoT solutions using the FIWARE platform, analysing the tools it offers, the strategies it has available to mitigate potential drawbacks and how it leverages the inherent advantages of IoT. In addition, we will assess how solving certain problems or exploiting complex solutions could lead to unexpected difficulties in other aspects of the system.

Keywords: IoT, FIWARE, Smart Buildings

Índice

1 Introducción

1.1. Motivación

Según el estudio de Estevez et. al. (2021) [4], el crecimiento vertiginoso de la densidad de población en entornos urbanos es innegable. Se proyecta que para el 2030, el 60 % de las personas residirá en las ciudades.

Este éxodo masivo de áreas rurales hacia núcleos urbanos impulsa una acelerada evolución del entorno de la propia ciudad y los desafíos a los que estas se enfrentan, o se anticipa que enfrentarán, cambian radicalmente: transporte, sostenibilidad medioambiental, salud pública, servicios y mucho más. Son retos críticos que reclaman una atención especial para mantener y elevar la calidad de vida en nuestras urbes.

Todo este panorama se ve magnificado por el explosivo crecimiento de infraestructuras de telecomunicación, desde banda ancha y WiFi hasta la omnipresente 5G, así como por las inversiones masivas en inteligencia artificial y big data. ¿El resultado? El Internet de las cosas (IoT, *Internet of Things* en inglés) emerge como un candidato prometedor para implementar soluciones innovadoras y ofrecer mejoras sustanciales tanto a los gobiernos como a los ciudadanos. El IoT puede ser la clave para optimizar la gestión de las ciudades, mejorar la calidad de vida de sus habitantes y, al mismo tiempo, preservar nuestro preciado entorno.

Mi especial interés por el IoT ha sido alimentado por mi participación activa en proyectos relacionados con esta tecnología. He vivido parte de cómo, lo que comenzó como una idea sencilla de conectar dispositivos a Internet, ha evolucionado de manera impresionante. De hecho, según Elnashar and El-saidny (2018) [3], en los últimos años, el uso del IoT ha experimentado un aumento exponencial en presupuestos y el número de dispositivos conectados.

También es importante mencionar que la situación global de la COVID-19 ha incentivado y acelerado la aplicación del IoT en diversos sectores, incluida la salud pública, la seguridad y

la privacidad.

Por todo esto, la principal motivación de este trabajo es indagar en el desarrollo de un sistema que sea capaz de aprovechar el potencial que ofrece IoT en un contexto concreto como el de las ciudades inteligentes, también conocido como *Smart City*, teniendo especial consideración en puntos importantes que se deben tener en cuenta a la hora de trabajar en un proyecto de IoT, como la seguridad, el rendimiento o la gestión de dispositivos y datos.

1.2. Objetivos

El objetivo principal de este trabajo será estudiar cómo funciona una plataforma IoT real y ser capaz de desplegar los servicios necesarios para poder recopilar datos de sensores y poder disponer de esos datos. Trabajaremos con FIWARE, que es una plataforma orientada al desarrollo de la IoT moderna, en continuo desarrollo y que utilizan diversas empresas y organismos en sus entornos de producción. Será de especial interés examinar los siguientes aspectos del sistema:

- **Infraestructura** ¿Qué es la plataforma FIWARE? ¿Cómo se puede implementar una infraestructura que soporte IoT con FIWARE? ¿Qué problemas presenta IoT y qué recursos utiliza FIWARE para mitigarlos? ¿Qué desventajas podría suponer usar una plataforma como FIWARE?

- **Generación de datos** ¿Cómo se pueden hacer compatibles con la plataforma la mayor cantidad posible de dispositivos?

- **Consumo de datos** ¿Qué debe incluir el desarrollo de una interfaz gráfica que haga peticiones al servicio para mostrar los resultados?

Para poder aproximarnos a una solución realista, nos basaremos en una problemática real, como es la de las ciudades inteligentes, y, así, aterrizar en una solución coherente. El objetivo será desarrollar una plataforma virtual común que permita: gestionar y organizar la información de cualquier edificio dentro de una ciudad e interconectar cualquier elemento integrable a la red de Internet (dispositivos IoT) que se encuentren en un edificio (*Smart Building*) usando interfaces de comunicación que ya son estándar. Dado que el software desarrollado hará uso de las tecnologías del "Internet" de las Cosas para la gestión de dispositivos conectados a la red

de un "Edificio" Inteligente (*Smart Building*), se ha decidido darle el nombre de `IBuilding`. Esta será la **infraestructura** de nuestro sistema. Por otro lado, para visualizar los datos necesitaremos una interfaz gráfica que **consuma** los datos del API, que denominaremos `Frontal Building`. Además, para poder **alimentar** el servicio con datos, desarrollaremos un dispositivo con Arduino.

`IBuilding` será un gestor de los edificios de una ciudad con el que se podrá consultar información sobre los mismos y manipular sus dispositivos. Para ello, `IBuilding` deberá almacenar datos de los edificios y deberá disponer de herramientas para acceder y manipular los dispositivos que se instalen en los mismos.

De esta manera, se puede enriquecer el valor de una ciudad, permitiendo digitalizar más datos sobre ella y poder generar una información más fiel y útil para que otras personas y servicios digitales puedan hacer uso de ella.

Las funcionalidades principales que va a tener este software, son las siguientes:

- Listar y filtrar los edificios de una ciudad para tener información sobre ellos de una forma organizada.
- Conocer y gestionar la información de cada edificio individualmente.
- Potenciar los edificios equipados con dispositivos IoT con servicios y funcionalidades adicionales para mejorar la información disponible y permitir interacciones como la apertura de puertas o la llamada de ascensores.

1.3. Estructura del documento

El documento pretende ser una guía que expone el desarrollo y despliegue de una plataforma de gestión de datos en la IoT de manera simplificada. Dispondrá de una serie de secciones en las que se expondrán los conocimientos técnicos necesarios, una descripción de la problemática en detalle con casos de uso y requisitos, una descripción de una posible solución, utilizando los conocimientos explicados en secciones previas, y una descripción de parte del desarrollo de esa solución. Más concretamente, dispondrá de las siguientes secciones:

- **Antecedentes**: Breve descripción de conocimientos y experiencias previas relacionados con proyectos de IoT.

- **Estudio del arte**: Análisis de las tecnologías a utilizar para el desarrollo del software. Nos centraremos en analizar en detalle los aspectos más importantes del producto que se va a a utilizar como solución para desarrollar la plataforma IoT.

- **Descripción del problema y análisis de requisitos**: Captura de requisitos y organización del proyecto.

- **Diseño de la solución**: Utilizando la información que poseemos de la sección "Estudio del arte", se puede elegir una arquitectura basada en FIWARE para cubrir los requisitos expuestos en la sección "Descripción del problema y análisis de requisitos". Esta sección definirá esa arquitectura y se analizarán las decisiones de diseño que se han planteado durante el desarrollo del mismo.

- **Desarrollo del proyecto**: Esta sección describe una guía de los pasos que se han seguido para desplegar los diferentes microservicios destinados a hacer funcionar el sistema. Expone también diversos problemas que se encontraron durante el desarrollo y la solución que se tomó.

- **Conclusiones y líneas de trabajo futuras**: Esta sección constará de una breve reflexión sobre la problemática de IoT y cómo una implementación como FIWARE, por un lado ofrece estrategias para mitigar esos problemas, pero por otro nos introduce en un marco de trabajo que nos obliga a enfocarnos en otras problemáticas y dificultades.

2

Antecedentes

Previo a la realización de este trabajo fin de grado, participé en el desarrollo de dos proyectos de IoT en concursos de programación. Estos proyectos me han aportado ciertos conocimientos base que me han resultado muy útiles a la hora de proceder con este trabajo. Ambos concursos tenían como objetivo realizar un proyecto con dispositivos IoT utilizando el protocolo oneM2M [13] para interconectarlos.

OneM2M Hackaton Málaga Organizado por la Universidad de Málaga en conjunto con la ETSI [1] (European Telecomunications Standards Institute), el objetivo era mejorar la vida de los ciudadanos y ayudar de alguna forma a combatir la COVID.

Solución. Un ascensor inteligente controlado desde el móvil, lo cual facilita la vida a personas con dificultades de movilidad y evita tener que tocar el ascensor, dando más seguridad frente a la COVID. Por temas de seguridad, para que no fuera susceptible a ser llamado desde cualquier lugar en cualquier momento, generábamos una contraseña de un solo uso (OTP, en inglés One Time Password) que era generada cada vez que una persona se acercaba al ascensor y era mostrada en una pantalla para introducirlo en el móvil. Una vez usada, si quisieras volver a llamar al ascensor, tendrías que volver a acercarte para generar otro OTP, el ultimo generado ya quedó inservible.

Intelligent IoT oneM2M Hackathon Organizado por la universidad de Sejong, ETSI y KETI (Korea Electronics Technology Institute), el objetivo era construir una solución IoT que pudiera ayudar a los ciudadanos o resolver problemas medioambientales o sociales importantes.

[1] https://www.etsi.org/

Solución. En este concurso realizamos un proyecto de Smart Building, en el que controlábamos de manera automática la puerta de un garaje. Cuando el coche llegaba a la puerta del garaje era detectado por un sensor de presencia que notificaba a una cámara para hacer una foto. Esta foto era procesada por una IA que extraía la matrícula como texto. De esta forma decidíamos si abrir automáticamente la puerta del garaje comprobando si la matrícula estaba en una lista de permitidos. Una vez hecho esto, notificábamos al edificio que la persona estaba entrando al edificio, el cual avisaba al ascensor realizado en el otro proyecto para ir bajando al parking y avisábamos al módulo de Smart Home para realizar las operaciones que esa persona hubiera configurado con sus dispositivos de casa.

2.1. Conocimientos

En ambos concursos se recibió una formación previa sobre qué es el protocolo oneM2M, como utilizarlo y cómo desarrollar con él. Fueron unas charlas por videoconferencia donde daban las herramientas básicas para empezar a trabajar con el protocolo. Durante el resto de las semanas teníamos a disposición expertos para transmitirle dudas.

2.1.1. Protocolo OneM2M

El protocolo oneM2M es un protocolo máquina-a-máquina (en inglés *machine to machine*), es decir, aquel que permite el intercambio de información entre dos dispositivos (comunicación y envío de datos). La comunicación que se produce entre las máquinas o dispositivos es autónoma, es decir, no hace falta intervención humana para que se produzca este intercambio de datos. Este protocolo se ha definido en conjunción por varias entidades importantes: CCSA (China), TTA (Corea), ATIS (Estados Unidos), TIA (Estados Unidos), ETSI (Europa), TSDSI (India), ARIB (Japón), TTC (Japón).

Entidades Según [12], oneM2M organiza los datos en un árbol cuyos nodos se denominan entidades. Las entidades más comunes son:

Entidad de servicios comunes (CSE, en inglés *Common Service Entity*) que funcionará como raíz del resto de entidades.

Entidad de aplicación (AE, en inglés *Application Entity*) que agrupará todas las entidades relacionadas con un dispositivo específico.

Contenedor (CNT, en inglés *Container*) que albergará un listado de instancias de contenedor (CI, en inglés *Container Instance*) que serán datos.

Políticas de control de acceso (ACP, en inglés *Access Control Policies*) que contendrá información sobre quién tiene acceso y bajo qué condiciones.

Estructuración del árbol de entidades Para seguir una buena arquitectura de desarrollo, en los talleres nos recomendaron seguir la siguiente estructura:

Todos los AE contienen los siguientes contenedores (CNT)

- **DESCRIPTION**: con la descripción del AE
- **DATA**: para almacenar las información en instancias (CI)
- **COMMAND**: si el AE es un actuador, en este contenedor se puede instanciar con la instrucción (CI) que queremos que el actuador ejecute.

Otro patrón a destacar era el de Monitor, en el cual cada aplicación se desarrolla independiente y la lógica de negocio (suscripciones y envío de datos) se concentra en una aplicación especial, denominada el Monitor.

2.2. Estado

2.2.1. Ascensor Inteligente

En el vídeo del proyecto [14] se pueden ver cuales son las entidades que se desarrollan:

- AE Lightsensor: Sensor que avisa cuando una persona se aproxima al ascensor.
- AE Motor: Actuador que mueve al ascensor a una planta especificada.
- AE Screen: Actuador que va mostrando información al usuario (la OTP, mensajes si hubo un error, o el ascensor está en camino, ...)

- AE App: Es el monitor de la aplicación. Tiene toda la lógica de suscripciones y envío de datos del ascensor. Se comunica con un frontal para ofrecer una interfaz de usuario.
- AE Statistics: Almacena estadísticas de uso

2.2.2. Edificio Inteligente

En la web de Hacksters [2] se expone el proyecto, donde se pueden ver cuales son las entidades que se desarrollan:

- AE para el garaje: Proximity Sensor AE, Camera AE, License Plate Recognition AE, Motor AE y Semaphore AE
- Las AE del proyecto del ascensor: Elevator AE, Elevator Motor AE, Elevator Proximity Sensor AE, Elevator Screen AE
- Monitor AE: El monitor de todo el sistema. Se comunica con un frontal

Una serie de frontales y backends

- Microservicio para almacenar los datos.
- frontal del Monitor
- frontal del monitor del ascensor
- frontal para registrarte en el edificio con tu matrícula

3

Estudio del arte

En esta sección se va a realizar un análisis de las tecnologías que se van a emplear para el desarrollo del Software. Para ello, se va a comenzar con un análisis sobre el IoT: ¿Qué es?, ¿qué podemos hacer con esta tecnología y qué problemas presenta que se deban de tener en cuenta? Para tener el contexto necesario para analizar en profundidad los aspectos más importantes de FIWARE.

3.1. Internet de las Cosas

El internet de las Cosas, o IoT, para abreviar, es una red de objetos físicos que están incrustados con electrónica y poseen conectividad a la red, lo que permite a estos objetos recopilar e intercambiar datos.

Si bien internet, que consiste en una red de usuarios y servidores conectados entre sí, es muy potente y ambicioso, IoT lo es más todavía, ya que además de abarcar a lo anterior, considera cualquier objeto físico como posible participante de la red. Esto permite que los objetos sean detectados y controlados de forma remota a través de la infraestructura de red existente, creando oportunidades para una integración más directa del mundo físico en los sistemas basados en computadora, y resultando en una mayor eficiencia, precisión y beneficio económico además de una reducción de la intervención humana. En general, se distinguen dos tipos de dispositivos:

- **Sensores**: Son aquellos dispositivos destinados a obtener datos de cómo se encuentra el medio continuamente. Esto se podrán utilizar para decidir que hacer en un momento determinado o para almacenarlos como histórico de evolución.
- **Actuadores**: Son aquellos dispositivos que están destinados a realizar una operación o modificación en el medio. Suelen transformar un input digital en una acción física.

Gracias a esto, se obtienen una serie de ventajas que son interesantes de tener en cuenta:

- **Sensorización**: Permite obtener datos que anteriormente no podían ser recogidos de manera automática.
- **Acciones remotas**: Los objetos que se conectan a la red pueden realizar acciones físicas desencadenadas por internet, por lo que no es necesaria ninguna presencia humana ante el objeto para poder manipularlo o monitorizar su estado.
- **Interacción entre dispositivos IoT**: Gracias a los modelos de comunicación, se pueden realizar tareas complejas comunicando diferentes sensores y actuadores, de esta forma, se producen una secuencia de acciones sin necesidad de intervención humana, que produce unos resultados más inteligentes, ya que, al estar conectadas con el resto de dispositivos, pueden disponer de información adicional útil para su objetivo.

Con todo esto en mente, tenemos que IoT es una tecnología relativamente reciente y posee un amplio espectro de aplicaciones que pueden generar beneficios significativos y dotan al sistema de un mayor nivel de inteligencia:

- **Orquestación IoT**, en inglés *IoT orchestration* [11], que permite organizar los dispositivos para combinar sus operaciones y resultados, concediendo al sistema un mayor nivel de inteligencia.
- **Computación en el borde**, en inglés *Edge Computing* [16], que permite optimizar la comunicación entre dispositivos, evitando llamadas innecesarias a los servidores, mejorando la eficiencia y sostenibilidad.
- **Gemelo digital**, en ingles *Digital Twin* [10], que se trata de una representación digital, alimentada con información capturada con sensores y otros dispositivos digitales para que sea lo mas fiel posible, de un sistema real, del cual se hacen preguntas y se intenta gestionar. De esta forma se puede conocer el estado del sistema de una manera sencilla y se pueden simular supuestos para observar qué ocurriría y en qué afectaría al sistema en general (este concepto se denomina *Sandbox* en inglés).

Sin embargo, analizando el avance del IoT desde sus comienzos, es evidente que ha experimentado un crecimiento vertiginoso, dejando aún muchos aspectos por desarrollar, como por ejemplo:

- **Interfaces de Estandarización**: Existen infinidad de modelos de datos para hablar de los mismos conceptos. ¿Cuál es mejor? ¿El que tiene más datos? ¿Se pueden establecer equivalencias?
- **Protocolos de descubrimiento**: ¿Cómo puedo ser capaz de manera automática de localizar y comunicarme con cualquier dispositivo IoT disponible?
- **El pantano de datos**, en inglés *Data Swamp*: Un sensor puede generar muchos datos en una fracción de segundo, ¿Qué pasa en el momento en el que tenga una ciudad llena de dispositivos así? ¿Y un país? ¿Cómo se organizan esos datos para acceder al que nos interesa de una manera óptima? ¿Cómo puedo extraer información de entre tantos datos?
- **Seguridad**: Por un lado, la cantidad de agentes conectados a la red es tan elevada que es fácil suplantar su identidad, es decir, utilizar dispositivos fraudulentos que se hagan pasar por los dispositivos reales para introducir datos falsos o manipular parte del sistema. Por otro lado, obtener acceso a ciertos dispositivos, como por ejemplo la cerradura de la puerta de una casa, puede comprometer la seguridad. También es posible provocar un mal uso de los dispositivos para acelerar su desgaste (casos famosos como *Stuxnet* [15]) o provocar daños mayores. Además, sería interesante considerar los problemas de privacidad en este contexto. La adquisición masiva de datos puede comprometer la privacidad de los usuarios, lo que nos lleva a la gestión de los datos obtenidos, su posible anonimización y la importancia de controlar el acceso a los mismos.

A la hora de realizar un desarrollo de IoT, se podría abordar desde 0, sin embargo, es más interesante hacer uso de plataformas que ya ofrecen la infraestructura necesaria para explotar las ventajas que presenta el IoT y ofrecen estrategias para mitigar los problemas que se presentan. Es por este motivo por el que usaremos FIWARE.

3.2. La plataforma de FIWARE

Según se expone en el vídeo de formación [9] de FIWARE Zone [2], FIWARE proviene de una iniciativa de la Unión Europea para impulsar el desarrollo de tecnologías y estándares

[2]FIWARE Zone es una iniciativa de la Junta de Andalucía y Telefónica que pretende dar a conocer FIWARE y ofrecer formación a nivel administrativo y técnico. El proyecto posee centros en Málaga y Sevilla desde donde

abiertos para el internet del futuro, enfocándose especialmente en aplicaciones relacionadas con ciudades inteligentes (*Smart Cities*, en inglés) e IoT. Según se comenta en el vídeo y en su web [5], aunque en realidad FIWARE es la definición de los estándares de comunicación entre los dispositivos IoT (véase figura 1), también se le llama FIWARE a la organización que la mantiene y al ecosistema de componentes software que lo componen, basados en arquitectura de código abierto que permiten la creación de aplicaciones basadas en la nube. Pretende ser un integrador de soluciones en cualquier contexto, es decir, FIWARE no está desarrollado para una lógica de negocio concreto, sino que pretende ser un sistema **abierto** e **interoperable**.

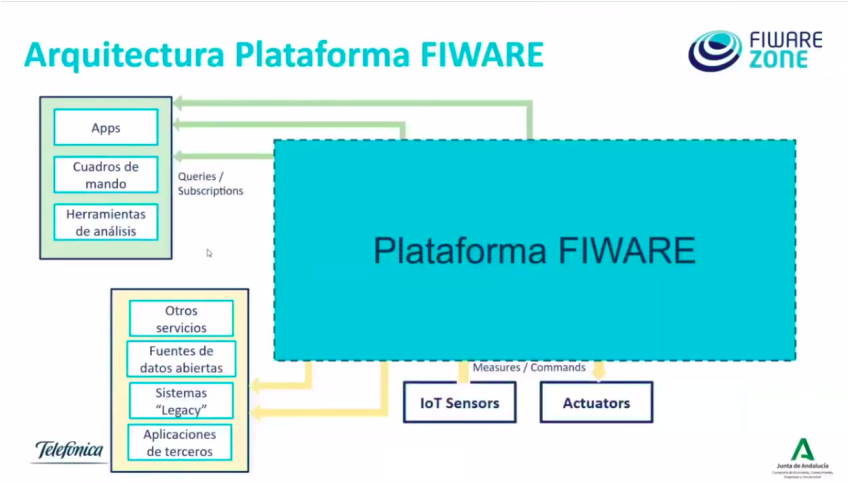


Figura 1: Ámbito de FIWARE como interfaz de comunicación. Imagen extraída de [9]

Vamos a analizar, a continuación, el interior de la caja de la Plataforma de FIWARE.

3.2.1. Catálogo de Componentes de FIWARE

Si queremos entrar en analizar los componentes que ofrece la plataforma de FIWARE, podemos hacerlo desde el punto de vista de la norma española. Si quisiéramos desarrollar una plataforma para un ciudad inteligente, en UNE 178104:2017 [17], se describe la arquitectura que se debe utilizar como referencia. Según [9], FIWARE ofrece muchos de estos componentes. En concreto, haciendo referencia a los componentes de la figura 2, FIWARE cubre la capa de interoperabilidad, la capa de conocimiento y la capa de adquisición/interconexión y algunos módulos de la capa de soporte (registro o *logs*, agenda, control de plataforma y seguridad).

ofrecen formaciones y talleres a empresas, personal de administraciones, y cualquier público interesado.

Aunque dentro de las capas hay algunos componentes que no cubre directamente. Por ejemplo, en la capa de interoperabilidad, FIWARE no dispone directamente de un portal de datos abiertos, pero sí hay integraciones nativas con portales ya existentes. Por otro lado, la semántica de ciudad tampoco está definida en FIWARE, sino que son los datos armonizados los que nos darán información sobre cómo son los datos que debe manejar el sistema.

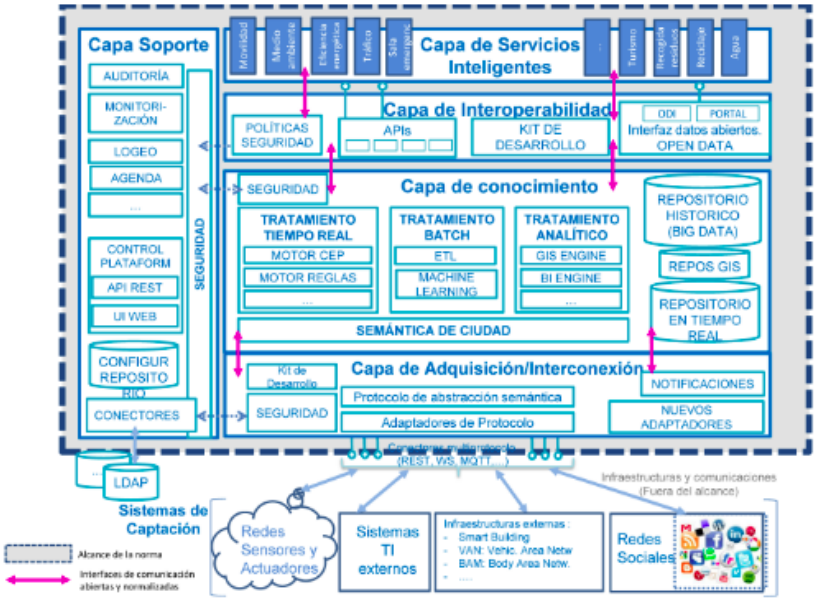


Figura 2: Modelo normalizado de capas de una plataforma de ciudad inteligente según [17]

En la práctica, los componentes de la plataforma FIWARE, también denominados `Generic Enablers`, serán una serie de microservicios, generalmente opcionales, que nos van a permitir añadir funcionalidades al sistema. A la hora de diseñar un sistema basado en FIWARE, debemos seleccionar todos aquellos que nos interesen en función de los requisitos. Los `Generic Enablers` se pueden clasificar según su funcionalidad en: gestión de contexto, interfaces IoT, robots y sistemas de terceros, procesamiento, análisis y visualización de datos de contexto, seguridad, gestión de API, publicación, monetización de datos, componentes externos de terceros. En el catálogo de FIWARE [3] se puede encontrar más información sobre los `Generic Enablers`.

Todos los microservicios que se vayan a utilizar deberán desplegarse con su debida configuración para trabajar conjuntamente y ofrecer una API con los servicios. El único `Generic`

[3] https://www.fiware.org/catalogue/

`Enabler` que es obligatorio utilizar es el de gestión de contexto, que almacena los datos y los sirve mediante la interfaz NGSI, la cual será otro elemento clave a tener en cuenta en la construcción del sistema.

3.3. Interfaces NGSI

Según [6], NGSI (en inglés, *Next Generation Service Interface*) es una interfaz propuesta por FIWARE y recogida en las normas de la ETSI. Se trata de una serie de normas abiertas, orientadas al intercambio de datos de contexto [4] donde se definen:

- un **modelo de datos** para la información de contexto, basado en un modelo de información simple que utiliza la noción de entidades de contexto [5].

- una **interfaz de datos** de contexto para intercambiar información mediante operaciones de consulta, suscripción y actualización.

- una **interfaz de disponibilidad** de contexto para intercambiar información sobre cómo obtener información de contexto.

Para cumplir el objetivo de transmitir cualquier tipo de dato, dependiendo del contexto, su modelo base debe representar cualquier dato genérico (entidad de contexto). Veremos que, dependiendo de la versión de NGSI que utilicemos, el modelo de datos limitará la potencia y el alcance de lo que se puede hacer con cada versión. Las versiones actualmente en uso son NGSI-V2 y NGSI-LD. También existe la versión NGSI-V1, que es la inicial, pero se desaconseja su uso, ya que se ha marcado como obsoleta (*deprecated*), a menos que se requieran las funcionalidades de registro o descubrimiento de proveedores de los datos, que todavía está en desarrollo en NGSI-V2. Incluso en ese caso, existe la posibilidad de coexistencia con la versión 2. Para más detalles de los modelos de NGSI-V2 y NGSI-LD, se puede consultar el apéndice B.

[4] Los datos de contexto son aquellos que aportan información sobre el contexto del verdadero dato que se va a enviar: estructura, semántica...

[5] Una entidad de contexto (o simplemente entidad) es aquella que representa una cosa (objeto físico o digital) con información para contextualizarla.

3.4. Datos armonizados

Como se ha comentado anteriormente, FIWARE pretende ser un integrador de soluciones en cualquier contexto, abierto e interoperable. Para ello, hará uso de los **datos armonizados**, que son interfaces que nos dan información sobre cómo son los datos que debe manejar el sistema. Veremos que en lugar de definir sus propios modelos de datos, ofrece herramientas más genéricas, como Linked Data y el uso de ontologías para ser flexible y permitir modelos de datos estándares. Esto nos permitirá usar un modelo de datos que ya todo el mundo conoce y hacer referencias a objetos con identificadores únicos para acceder a su información.

Estos datos armonizados se van a convertir en un elemento clave, ya que solventan el problema de interfaces de estandarización descrito en anteriores secciones, además, según [1], como la visión de cómo construir los sistemas que plantea FIWARE se basa en el paradigma de Gemelo Digital, es necesario analizar qué datos van a ser relevantes para el modelo que se quiere replicar. En este caso, los datos armonizados nos van a proporcionar unos esquemas de datos donde estará toda la información necesaria.

3.4.1. Ontología SAREF

SAREF, del inglés *Smart Applications REFerence*, es un proyecto abierto de la ETSI cuyo objetivo es definir una serie de ontologías. Se trata de modelos compartidos consensuados que facilitan la correspondencia entre los modelos de datos de diferentes sistemas existentes en el ámbito de las aplicaciones inteligentes. Gracias a NGSI-LD, podemos hacer uso de estos modelos de datos, que son interoperables con el modelo de datos armonizados.

En la web del SAREF [6] se especifican explícitamente los conceptos básicos más recurrentes en el ámbito de aplicaciones inteligentes, las relaciones entre ellos y las restricciones. Los principios fundamentales son:

- **Modularidad**: Los modelos de datos se organizan en módulos, cada uno de los cuales se centra en un dominio de aplicación específico. Por ejemplo, SAREF cuenta con un módulo de edificios, otro de energía, de ciudades, de agricultura, etc.
- **Reutilización**: Se reutilizan los conceptos y relaciones para conectar los módulos. Por ejemplo, si se define un módulo de Edificios con un tipo para edificios, o espacios, luego

[6] https://saref.etsi.org/

si se define un módulo de ciudades, se puede hacer referencia a los espacios para definir parques, o al tipo edificio para definir una relación de pertenencia entre un edificio y una ciudad, o de edificios en la misma calle.

- **Extensibilidad**: Los diferentes módulos recogen los conceptos básicos y más comunes a la hora de desarrollar aplicaciones inteligentes, pero pueden existir lógicas de negocios que necesiten ciertos conceptos más específicos. Se permite la posibilidad extender los conceptos básicos para adaptarlos a las necesidades de cada aplicación.

- **Mantenibilidad**: Los modelos de datos se mantienen en un repositorio público, lo que permite que se puedan actualizar y mejorar con el tiempo.

3.5. El núcleo de FIWARE: Context Broker

Para que una plataforma se pueda calificar como "Basado en FIWARE", tiene que hacer uso del *Context Broker* [7] de FIWARE, que será el encargado de gestionar y facilitar los datos de la plataforma. Habilita la posibilidad de gestionar la información en una organización de gran escala altamente descentralizada. Alrededor de este, se encontrarán componentes que soportarán y facilitarán la captura, procesamiento, seguridad, etc. de los datos.

En la web de FIWARE [5] podemos encontrar una serie de tutoriales que explican las bases para entender los servicios que pertenecen al ecosistema y poder desarrollar sobre ellos. El más básico empieza a hablar sobre el *Context Broker*. Nos expone la arquitectura mínima necesaria para que funcione y nos va indicando pruebas a realizar sobre el mismo para ver cómo funciona. Los despliegues del tutorial se pueden ver en la figura 3. Constan de `Orion Context Broker`, uno de los *Context Broker* que existen. Este está conectado a una base de datos `MongoDB`. Desde el puerto 1026, que es su puerto por defecto, podemos hacerle peticiones usando algún programa para ellos y probar su funcionamiento.

[7] El patrón de diseño Bróker es un patrón de arquitectura que puede ser usado para estructurar sistema de software distribuido con componentes desacoplados que interactúan mediante llamadas remotas procedurales. El bróker es el responsable de coordinar la comunicación, como redirección de peticiones o transmisión de resultados y excepciones.

Figura 3: Arquitectura mínima para que FIWARE Context Broker funcione, según el tutorial de [7]

3.5.1. Implementaciones

En la tabla 1 se muestra una comparativa de las diferentes implementaciones de *Context Brokers*. Esta comparativa nos servirá para decidir cuál de ellos será más interesante utilizar. Veremos que hay implementaciones de diferentes tamaños. Un tamaño mayor implicará despliegues que requieran más recursos, pero permitirá un tráfico mayor y soportará una densidad de datos mayor. Por otro lado un tamaño menor significará que podrá ejecutarse en servidores más ligeros, pero soportará una carga menor de datos. De hecho los más ligeros son ideales para utilizar en nodos del borde. En un contexto de computación en la niebla, estos nodos están localizados en redes cercanas a los dispositivos reales, en lugar de servidores descentralizados. Estos nodos recibirán menos carga de datos, pues su objetivo es proporcionar servicio a los dispositivos de una zona solo. Esto nos permitirá realizar parte del procesamiento en esos nodos sin saturar los servidores de la nube, lo cual mejora los tiempos de respuesta y reduce el coste energético al ser servidores más ligeros.

También es interesante conocer qué interfaz soportan cada uno de estos brókers. Tenemos casos de algunos que soportan la V2, otros que soportan la versión de Linked Data, pero también casos que soportan ambas o no terminan de soportar del todo ninguna. (Ofrece subfuncionalidades de estas API).

3.6. Los adaptadores NGSI: `IoT Agents`

El problema más importante al que se enfrenta NGSI es su uso real. En general, los dispositivos fabricados no implementan NGSI, sino que implementan otras interfaces más comunes, por lo que no se pueden adaptar a NGSI directamente. Es responsabilidad de FIWARE desarrollar herramientas que permitan establecer comunicación con estos dispositivos, por ello,

Nombre	**Descripción**	**Interfaces Soportadas**	**Tamaño**	**Entorno**
Orion	Context Broker referente en la web de FIWARE	NGSI-V2	normal	nube
Orion-LD	Alternativa a `Orion Context Broker` que utiliza la interfaz NGSI-LD	NGSI-V2 NGSI-LD	normal	nube
Cepheus Broker	Alternativa muy ligera pensada para nodos en el borde. Implementa muy pocas funcionalidades, el resto lo delega en el Context Broker de la nube.		muy ligero	borde
ThinBroker	Implementación adaptada para funcionar en un entorno Fog Flow en el borde.	NGSI-LD	ligero	FogFlow y borde
Scorpio Broker	Alternativa a `Orion Context Broker` mucho más potente y pesada, para entornos muy grandes.	NGSI-LD	muy pesado	sistemas federados y nube
Stellio	Alternativa a `Orion Context Broker` más potente.	NGSI-LD	pesado	nube

Cuadro 1: Listado de implementaciones de Context Broker

existen los adaptadores de interfaces, denominados "IoT Agent". Se puede encontrar información sobre ellos y tutoriales útiles en la web de FIWARE. Véase [8].

Los `IoT Agents` son componentes cuyo objetivo es adaptar la comunicación entre FIWARE y el resto de dispositivos pasando de NGSI a cada protocolo. Estos actuarán de *facade* del servicio de FIWARE para que los dispositivos puedan interactuar con él. Existen agentes desarrollados para protocolos como Lightweight M2M [8], LoRaWAN [9], UltraLight2.0 [10] o OneM2M, del cual ya hemos hablado en los antecedentes. Para permitir la comunicación con dispositivos que utilicen estas interfaces, será necesario levantar el agente apropiado. En caso de que no exista un agente para el protocolo que se necesite, existen plantillas para personalizar un `IoT Agent` a cualquier protocolo.

[8] https://omaspecworks.org/what-is-oma-specworks/iot/lightweight-m2m-lwm2m/

[9] https://lora-alliance.org/about-lorawan/

[10] https://fiware-iotagent-ul.readthedocs.io/en/latest/usermanual.html

3.7. Protocolos de seguridad

El `Context Broker` de FIWARE es el elemento principal de la plataforma. Existen multitud de agentes internos y externos que quieren acceder a su API para interactuar con el sistema, pero exponerlo directamente al exterior generaría multitud de problemas de seguridad y de protección de datos. El núcleo de FIWARE no dispone de mecanismos de autodefensa, sino que requiere de otros `Generic Enablers` que le proporcionen esa seguridad.

Estos `Generic Enablers`, categorizados en la sección de "seguridad", van a dotar al sistema de la protección necesaria con diferentes técnicas y protocolos, como "OAuth2.0", el uso de PDP, Proxies PEP o API Gateways.

3.7.1. OAuth2.0

OAuth2.0 [11] viene de *"Open Authorization"* en inglés o "Autorización Abierta" en español, y es un protocolo de **autorización** que permite restringir, o conceder, permisos para acceder a los recursos o realizar operaciones sobre los mismos. "OAuth2.0" permitirá registrar y autenticar al usuario en la plataforma de forma segura.

El servidor de OAuth2.0 alberga datos de usuarios, aplicaciones e información sobre los permisos que los usuarios tendrán sobre estas aplicaciones.. El cliente se encargará de establecer comunicación con el servidor de autorización para conocer las capacidades del usuario en la aplicación. El cliente se abstrae de la autenticación del usuario, es decir, no maneja nombres de usuarios, ni correos, ni contraseñas.

OAuth2.0 delega la gestión de los usuarios a un microservicio especializado, lo cual se puede considerar una ventaja, ya que se centraliza toda la lógica y mejora la seguridad, teniendo que acotar la protección de los usuario a ese microservicio. Otra ventaja podría ser el uso compartido por distintos servicios, lo que permite tener una cuenta utilizable por múltiples servicios. No será necesario que el usuario cree cuentas en los diferentes servicios que se ofrecen. Por otro lado, hay que tener en cuenta que el incremento de complejidad del sistema al tener que trabajar con multitud de microservicios puede ser una desventaja.

OAuth2.0 dispone de diferentes flujos de concesiones, es decir, el conjunto de pasos que un cliente tiene que realizar para obtener la autorización de acceso a los recursos. Estos flujos

[11] https://auth0.com/es/intro-to-iam/what-is-oauth-2

son: concesión de código de autorización, véase en la figura 4, concesión implícita, concesión de credenciales del propietario del recurso, concesión de credenciales de clientes, autorización de dispositivos y token de actualización. Cada uno de estos flujos tienen características que lo hacen más apropiados a determinados contextos, sin embargo, a rasgos generales, podemos decir que todos estos flujos realizan los siguientes pasos:

1. La **solicitud de autorización**, donde el cliente (aplicación web, móvil, ...) solicita autorización al servidor de autorización. Para ello proporciona un identificador, un secreto para identificar al *cliente*, los ámbitos (a qué se quiere acceder) y un URI de redireccionamiento para enviar el código de autorización o token de acceso, dependiendo de qué tipo de concesión se utilice.

2. La **autenticación del cliente**. En este momento, el servidor de autorización autentica al cliente y verifica los permisos para acceder a los ámbitos especificados.

3. La **interacción del propietario del recurso**. El propietario del recurso interactúa con el servidor de autorización para conceder el acceso.

4. La **respuesta del servidor de autorización**. El servidor de autorización redirige al cliente usando la URI de redireccionamiento con el código de autorización, token de acceso o un token de actualización.

5. El **acceso al recurso** usando el token de acceso. El cliente solicita acceso al recurso desde el servidor de recursos.

3.7.2. Punto de decisión política

Según se define en el estándar de XACML [12], un punto de decisión de política (PDP, en inglés *Policy Decision Point*) es un mecanismo que evalúa peticiones de acceso a recursos para decidir si concedérselo a un usuario determinado. Es importante conocer estos sistemas y su alcance ya que FIWARE proporciona herramientas PDP para proteger el acceso a los recursos y

[12] eXtensible Access Control Markup Language (XACML) es un lenguaje de marcado estándar basado en XML, diseñado para especificar las políticas de control de acceso de un sistema. Enlace a la especificación: http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html

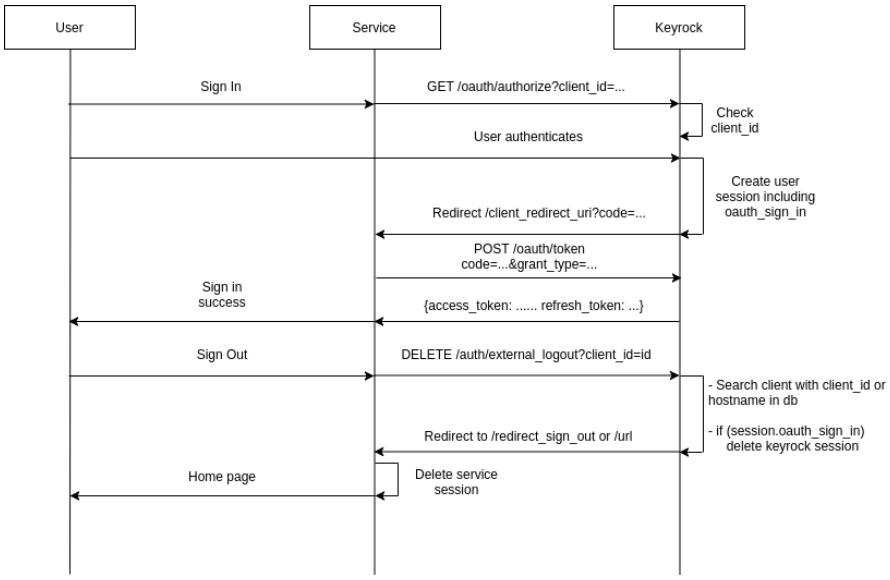


Figura 4: Flujo de OAuth2.0 por concesión de código de autorización. Extraído de los tutoriales oficiales de FIWARE de Keyrock.

dependiendo de las características necesarias, podremos saber qué configuraciones y servicios serán necesarios.

Desde el punto de vista de la granularidad, los PDP pueden tener tres niveles de de control de acceso.

- Primer nivel: Nivel de acceso autenticado. Distingue entre usuarios autenticados y usuarios anónimos.
- Segundo nivel: Nivel de autorización básica. Distingue usuarios, recursos y verbos. Es capaz de determinar si un usuario debería de poder acceder a un recurso determinado usando un verbo concreto (Por ejemplo, si yo puedo hacer un GET a /building).
- Tercer nivel: Nivel de autorización avanzada o de grano fino. Permite establecer, usando el lenguaje XACML, políticas más concretas de autorización. Por ejemplo podríamos expresar en este lenguaje que todos los usuarios que tengan el rol de Administrador pueden acceder a todos los departamentos cuya ciudad coincidan con la suya.

3.7.3. Proxy PEP

Según el estándar de XACML, los proxies PEP, del inglés, *Policy Enforcement Point*, son agentes de red diseñados para forzar la revisión de permisos de una petición y bloquearla si

procede.

Al ser un proxy, actúa de intermediario entre el Context Broker y el cliente, es decir, la IP que se proporcionaría como punto de acceso al servicio sería la del proxy PEP en lugar del Context Broker, por lo que este se podría ejecutar en una red privada mejorando la seguridad del sistema significativamente.

Este sistema, al recibir una petición entrante, interactuaría internamente con el PDP para consultar si el cliente tiene acceso, o no, al recurso solicitado. En caso positivo, establece la comunicación con el Context Broker y actúa de intermediario, y en caso de el PDP deniegue el acceso, bloquea la conexión.

3.7.4. API Gateway

Existe, como alternativa al proxy PEP, el API Gateway, que actúa como punto único de entrada al sistema. Esto permite, no solo forzar la validación contra el PDP, sino también monitorizar el tráfico de datos, poner restricciones de ratio de peticiones o tener un control más exhaustivo de las peticiones que se realizan al sistema.

3.7.5. Implementación OAuth2.0: Keyrock

Este Generic Enabler es un servidor de autenticación que utiliza el protocolo OAuth2.0. Dispone de todas las estrategias de concesión de permisos mencionadas en el apartado de anterior. En la figura 5, se puede observar un ejemplo minimalista de un sistema que utiliza Keyrock. El frontal debe autenticarse contra el microservicio para obtener sus credenciales para utilizar contra el Context Broker.

Por otra parte, la siguiente arquitectura presenta diferentes problema:

- La responsabilidad aplicar la decisión tomada por Keyrock depende del cliente. Usar Keyrock sólo no sería suficiente para añadir la seguridad deseada, ya que no se puede confiar en un tercero la responsabilidad de denegar el acceso a un recurso no permitido. Por este motivo, habría que incluir otros elementos como un Proxy PEP o un API Gateway.

- Keyrock puede alcanzar solo el primer nivel de control de acceso, o el segundo si se usara la concesión de credenciales del propietario del recurso. Si quisiéramos añadir

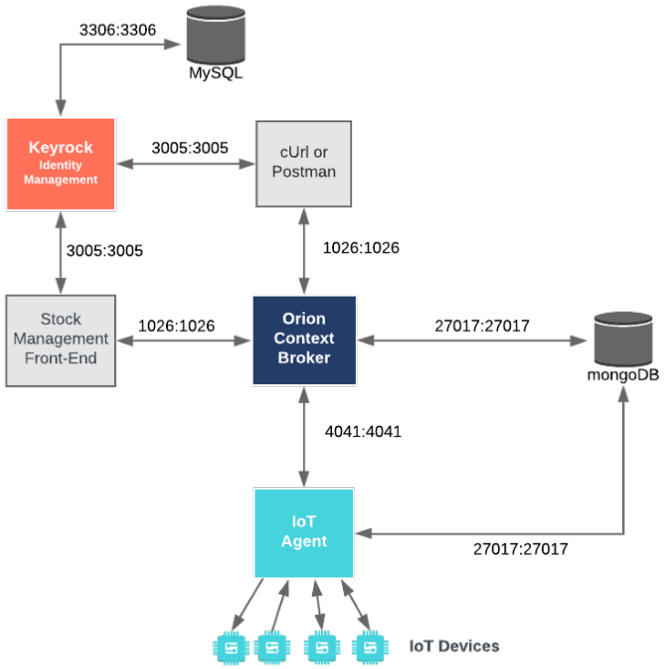


Figura 5: Componentes de un sistema mínimo basado en FIWARE integrando Keyrock, extraído de los tutoriales encontrados en el README del repositorio de Github de Keyrock.

un grado de control de acceso más alto, tendríamos que hacer uso de otros `Generic Enablers` que actúen de PDP, como `Authzforce`, que es un Generic Enabler que actúa de PDP con control de acceso hasta nivel 3, permitiendo establecer normas más específicas usando XACML. Como, establecer roles dentro de un edificio para separar datos públicos de datos privados.

3.7.6. Implementación Proxy PEP: `Wilma`

`Wilma` es un Generic Enabler de FIWARE que actúa un proxy PEP entre la API de FIWARE y el cliente. Realiza las comprobaciones necesarias para verificar si la entidad autorizadora (PDP) aprobó el acceso al recurso y actuar en consecuencia permitiendo el paso o bloqueando la respuesta. Se puede ver un ejemplo de los componentes mínimos necesarios para utilizar `Wilma` con FIWARE en la figura 6.

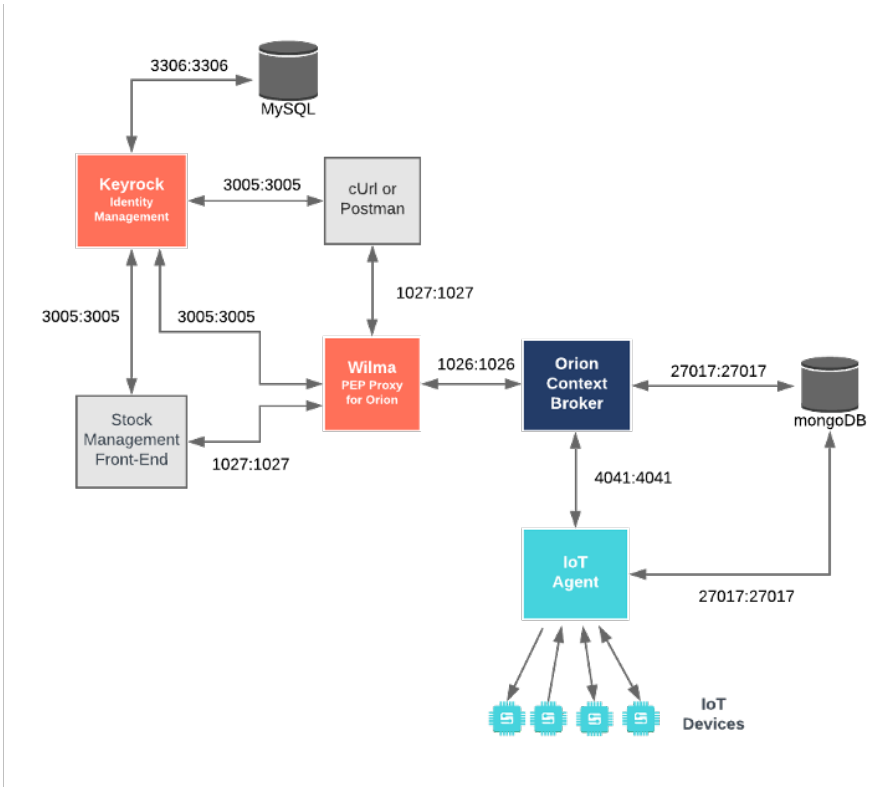


Figura 6: Componentes de un sistema mínimo basado en FIWARE integrando Keyrock y Wilma, extraído de los tutoriales encontrados en el README del repositorio de Github de Keyrock.

3.8. Computación en el borde

La computación en el borde, conocida como *Edge Computing* en inglés, se refiere a un modelo de procesamiento de datos en el que el almacenamiento y el procesamiento de la información se llevan a cabo más cerca del lugar donde se generan los datos, en contraposición a depender únicamente de centros de datos remotos o en la nube.

En lugar de enviar datos a largas distancias para su procesamiento en servidores centrales, la computación en el borde implica utilizar dispositivos más cercanos al lugar de origen de los datos, como sensores, dispositivos IoT, routers, o incluso servidores locales. Esto reduce la latencia y el tiempo de respuesta, ya que los datos se procesan más rápidamente y las acciones se ejecutan más cerca del punto de origen.

Este enfoque es especialmente útil en entornos donde la velocidad y la eficiencia son críticas, como en aplicaciones industriales, IoT, vehículos autónomos, sistemas de salud y muchas otras áreas donde se requiere un procesamiento rápido de datos en tiempo real. La compu-

tación en el borde ayuda a optimizar el rendimiento, mejorar la seguridad y reducir la carga en las redes al procesar datos de manera más localizada.

FIWARE dispone de un ecosistema de servicios para diseñar un sistema con computación en el borde. A esta conjunción de servicios se le denomina `Fog Flow`, el cual dispone de las herramientas necesarias para aplicar este paradigma al sistema desarrollado. Entre otros, dispone de:

- Un sistema para diseñar *workers* y poder desplegarlos automáticamente
- un *Task Designer* que tomará la decisión de qué `workers` desplegar en qué momentos.
- Una interfaz gráfica para diseñar los flujos de tareas y conectar los nodos.

3.9. Docker

Docker es una plataforma de código abierto diseñada para crear, desplegar y administrar aplicaciones en contenedores. Los contenedores son entornos livianos y portátiles que permiten empaquetar una aplicación junto con sus dependencias y configuraciones, lo que facilita su ejecución en diferentes entornos informáticos de manera consistente.

Con Docker, los desarrolladores pueden crear contenedores que contienen todo lo necesario para que una aplicación se ejecute, incluidas bibliotecas, herramientas, código y entorno de tiempo de ejecución. Estos contenedores pueden ejecutarse en cualquier sistema que tenga Docker instalado, lo que garantiza la consistencia y la portabilidad de las aplicaciones a lo largo del ciclo de desarrollo, desde el entorno de desarrollo hasta la producción.

Algunos beneficios clave de Docker incluyen la portabilidad, la eficiencia en el uso de recursos, la facilidad para escalar aplicaciones, la rápida implementación y la gestión simplificada de las aplicaciones en contenedores.

En resumen, Docker simplifica el proceso de desarrollo, implementación y administración de aplicaciones al permitir la creación y ejecución de entornos de aplicación estandarizados y portátiles en diferentes sistemas informáticos.

3.10. NextJS

Next.js es un popular framework de React utilizado para construir aplicaciones web modernas y potentes. Proporciona una forma eficiente y fácil de crear aplicaciones web basadas en React con características avanzadas, optimización para SEO (optimización de motores de búsqueda) y renderizado del lado del servidor (SSR, por sus siglas en inglés, *Server Side Rendering*) sin la necesidad de configuraciones complejas. Algunas de las características clave de Next.js incluyen:

- **Renderizado del lado del servidor**: Next.js permite renderizar páginas en el servidor, lo que mejora el rendimiento y la eficiencia de la carga inicial. También es capaz de generar sitios estáticos, lo que facilita la creación de páginas prerenderizadas para un mejor rendimiento y SEO.

- **Enrutamiento dinámico**: Ofrece un sistema para manejar la navegación entre páginas de manera intuitiva y dinámica. Simplemente con poner una página en la carpeta "src/ejemplo1/ejemplo2", ya se renderiza accediendo a la URL "/ejemplo1/ejemplo2".

- **Integración con React**: Next.js se basa en React y aprovecha sus ventajas, permitiendo a los desarrolladores utilizar todas las características de React junto con las funcionalidades adicionales que ofrece.

En resumen, Next.js es un framework potente y versátil que simplifica el desarrollo de aplicaciones web React al proporcionar características avanzadas, renderizado eficiente y optimizaciones que ayudan a mejorar el rendimiento, la SEO y la experiencia del usuario, además, una de las ventajas de utilizar NextJS es que, Vercel, su empresa creadora, ofrece servidores gratuitos para desplegar frontales desarrollados con esta tecnología.

3.11. Nginx

Nginx es un servidor web ligero de código abierto que destaca por su eficiencia y alto rendimiento. Originalmente diseñado para servir como un servidor HTTP y proxy inverso, Nginx ha evolucionado para abordar una variedad de roles en la infraestructura web moderna. Por otro lado, Nginx tiene una configuración muy flexible, lo que permite a los administradores

adaptar el servidor a las necesidades específicas de su infraestructura y aplicaciones utilizando unos archivos de configuración a través de un lenguaje de configuración propio. Entre sus funcionalidades principales, nos pueden resultar útiles las siguientes:

- **Proxy inverso**: Nginx actúa como un intermediario entre los clientes y otros servidores web, lo que le permite manejar solicitudes HTTP en nombre de otros servidores. Esto es crucial para distribuir la carga de trabajo, mejorar la seguridad y proporcionar escalabilidad a las aplicaciones web.

- **Servidor HTTPS**: Nginx ofrece capacidades avanzadas de cifrado y seguridad, lo que lo convierte en una opción popular para servir contenido web de forma segura a través del protocolo HTTPS. Su soporte para certificados SSL/TLS y su capacidad para configurar conexiones seguras hacen que sea una herramienta esencial para proteger la privacidad y la integridad de los datos transmitidos en línea.

En resumen, Nginx es una herramienta versátil que cumple un papel fundamental en la entrega de contenido web seguro, eficiente y escalable. Su capacidad para actuar como un proxy inverso y servidor HTTPS nos resultarán interesantes. Para más información sobre Nginx se puede consultar su página principal [13]

[13] https://nginx.org/en/

4

Descripción del Problema y Análisis de Requisitos

En este capítulo se describirá en detalle el análisis de requisitos del sistema, así como la metodología de trabajo que se ha seguido para la realización del proyecto, su estructuración y la planificación que se pretenderá seguir. Además, se explicarán las decisiones de diseño que se han tomado y la arquitectura que se va a desarrollar en el proyecto, apoyándonos en las explicaciones de las tecnologías disponible en la sección "Estado del arte".

4.1. Captura de Requisitos

En esta subsección, se documenta el análisis de requisitos del sistema. Tras describir en primer lugar las funciones principales del producto a desarrollar, y explicar brevemente las características de los usuarios del sistema, pasaremos a detallar los requisitos funcionales y los no funcionales del mismo. Se describirán también en detalle los principales casos de uso del sistema.

4.1.1. Funciones del producto

Como se menciona en los objetivos, se pretende desarrollar un sistema que sea capaz de gestionar los edificios de una ciudad que sea susceptible a ser usado por otros sistemas mediante una API.

En este sistema, cualquier usuario podrá registrarse, iniciar sesión y obtener información sobre diversos edificios, como cafeterías, librerías, bibliotecas y más. La aplicación se convier-

te en una herramienta útil para aquellos que buscan conocer la disponibilidad de servicios en tiempo real, la capacidad de los lugares y descubrir eventos y actividades en la ciudad. Con filtros personalizables, los usuarios pueden refinar sus búsquedas según sus necesidades específicas, ya sea preguntando por la disponibilidad de plazas de estacionamiento o de un producto, o por la capacidad de un lugar o nivel de saturación, como un hospital. Además, podrá acceder a información privada del edificio si el dueño del edificio le ha otorgado la capacidad para ello.

Un usuario podrá crear estos edificios rellenando un formulario con los datos necesarios que tendrán que ser revisado por un administrador del sistema. Una vez creado, se podrá gestionar su información y añadir los dispositivos IoT oportunos.

Los usuarios podrán tener roles dentro de los edificios. Estos definirán la capacidad del usuario de modificar información sobre el mismo, y la capacidad de interactuar con sus dispositivos, entendiendo interactuar como la capacidad de ver los datos que producen en caso de ser un sensor, o la capacidad de activarlo, en caso de que sea un actuador.

4.1.2. Características de los Usuarios

La aplicación de edificios inteligentes estará pensada para una amplia variedad de usuarios en una ciudad grande. Desde jóvenes hasta personas mayores, la aplicación busca ser accesible para los diferentes grupos demográficos.

La familiaridad con la tecnología puede variar, desde usuarios familiarizados con otras aplicaciones hasta aquellos con conocimientos básicos, por ello es importante seguir los principios de intuitividad que siguen las aplicaciones que son tendencia, como el uso de guías de estilos, como *Material Design*. La aplicación promete ser intuitiva y adaptarse a usuarios con diferentes niveles de educación y presupuestos, proporcionando una experiencia integral para explorar y conocer el entorno urbano de manera efectiva.

4.2. Requisitos funcionales

Algunos de los requisitos están extraídos de las operaciones CRUD de un recurso. Estas operaciones son creación (C, de *create*, en inglés), lectura (R, de *read* en inglés), modificación (U, de *update*) y borrado (D, de *delete*). En los requisitos se hará referencia a las operaciones por estas siglas.

ID	Título	Descripción
RF-EDI1	R edificios	Se pueden manejar los datos de un edificio: Cualquier usuario podrá visualizar los datos de edificios validados.
RF-EDI2	U edificios	Los editores y el usuario que envió la solicitud de creación del edificio (creador) podrá editar los datos.
RF-EDI3	D edificios	El creador del edificio o cualquier administrador de la plataforma será el único que podrá borrarlo.
RF-EDI4	Solicitud de creación de edificio	Cualquier usuario logueado puede solicitar la creación de un edificio rellenando un formulario. Al enviar los datos se insertan en la base de datos con un atributo que indique su estado, quedando solamente visibles para los administradores.
RF-EDI5	Aceptar solicitud creación de edificio	Un administrador podrá aceptar solicitudes de creación de un edificio. Una vez aceptadas cambia el estado a aprobado y pasan a ser visibles para todos los usuarios.
RF-EDI6	Denegar solicitud creación de edificio	Un administrador podrá denegar solicitudes de creación de un edificio. Cuando se rechacen, estos se eliminarán de la base de datos.
RF-USU1	R de Usuarios	Los usuarios logueados podrán visualizar los perfiles de otras personas, y sus propios datos.
RF-USU2	U de Usuarios	Los usuarios logueados podrán actualizar únicamente los datos de su perfil.
RF-USU3	D de Usuarios	Los usuarios logueados podrán darse de baja en cualquier momento.
RF-USU4	Registro en la aplicación	Cualquier usuario podrá registrarse en el sistema.
RF-USU5	Validación del acceso	Cualquier usuario registrado podrá iniciar sesión en la aplicación usando las credenciales proporcionadas en el registro.
RF-USU6	Cerrar sesión	Cualquier usuario logueado podrá cerrar sesión.
RF-ROL1	Otorgar rol de edificio	El creador de un edificio podrá otorgar roles en su edificio a cualquier usuario registrado en la aplicación. El criterio para otorgar los roles queda a cargo del creador.

RF-INI1	Acceso a la página de inicio	Cualquier usuario puede acceder a la página principal.
RF-DIS1	C dispositivos	El creador o los editores de un edificio podrán añadir dispositivos a un edificios. Se deberá indicar su visibilidad: Público o privado.
RF-DIS2	R dispositivos	Dependiendo de la visibilidad de los datos: ▪ público: Cualquier usuario que acceda al edificio podrá interactuar con el dispositivo. ▪ privado: Solo los usuarios que tengan uno de los siguientes roles: creador, editor o miembro, podrán interactuar con el dispositivo.
RF-DIS3	U dispositivos	El creador o los editores de un edificio podrán editar los datos de los dispositivos del mismo.
RF-DIS4	D dispositivos	El creador o los editores de un edificio podrán eliminar cualquier dispositivo perteneciente al edificio.
RF-FIW1	Control y gestión de dispositivos	Debe ser capaz de controlar y gestionar dispositivos y sistemas conectados en el edificio, como sistemas de iluminación, HVAC, puertas, persianas, entre otros. Esto debe ser posible desde una única plataforma central.

4.3. Requisitos no funcionales

ID	Título	Descripción
RNF-USU1	Servicio de Autenticación de FIWARE	Se utilizarán los microservicios de autenticación de `Identity Manager` para la autenticación de los usuarios y de los dispositivos en el sistema por compatibilidad con FIWARE.
RNF-USU2	Acceso anónimo	Un usuario no registrado podrá entrar en la aplicación de forma anónima. Solo podrán visualizar datos públicos de los edificios.
RNF-USU3	Token de sesión	Se utilizará un token de sesión JWT (del inglés, *Json Web Token*) que expirará cada 24 horas y que la aplicación guardará en el *Session Storage*.

RNF-USU4	Almacenamiento de la contraseña	Se almacenarán las contraseñas de los usuarios en una base de datos dedicada a las credenciales gestionada por `Identity Manager`.
RNF-EDI1	Ordenación en el listado de edificios	Los edificios se ordenarán por cercanía a la ubicación de la persona si se pudiera utilizar. En caso de no estar disponible, se ordenará por orden alfabético.
RNF-EDI2	Filtrado en el listado de ordenación de edificios	Los edificios se podrán filtrar por nombre, tipo de edificio, distancia o si tiene algún rol especial en ese edificio.
RNF-EDI3	Mapa en los detalles de un edificio	En la vista de los detalles, se incluirá un mapa mostrando la ubicación del edificio. Utilizará la API de Google Maps.
RNF-ROL1	Maestro roles de la aplicación	Hay 3 roles en la aplicación: administrador, usuario y anónimo.
RNF-ROL2	Maestro roles de un edificio	Cualquier usuario registrado podrá tener un rol de los siguientes en cada edificio: creador, editor, miembro, público.
RNF-FIW1	Capacidad de recopilación de datos	Debe ser capaz de recopilar datos de los sensores y dispositivos ubicados en diferentes áreas del edificio y en tiempo real. Además, debe ser capaz de manejar grandes volúmenes de datos.
RNF-FIW2	Integración con sistemas de terceros	Debe ser compatible con sistemas de terceros para garantizar la interoperabilidad. Debe poder integrarse sin problemas con sistemas existentes en el edificio.
RNF-FIW3	Análisis de datos	Debe ser capaz de analizar los datos recopilados de los sensores y dispositivos para proporcionar información valiosa.
RNF-FIW4	Seguridad y privacidad de los datos	Los datos sensibles se tratarán de manera especial para aumentar su seguridad. Por ejemplo, cifrar los datos o aplicarle un *hash* según sea necesario, como matrículas de coches o contraseñas. Estos datos deben estar protegidos contra la intrusión en el sistema.
RNF-FIW5	Escalabilidad	Debe ser escalable para que pueda manejar el crecimiento de edificios inteligentes. Debe ser capaz de adaptarse a la creciente cantidad de sensores y dispositivos conectados y proporcionar una capacidad de procesamiento adecuada para manejar los grandes volúmenes de datos.

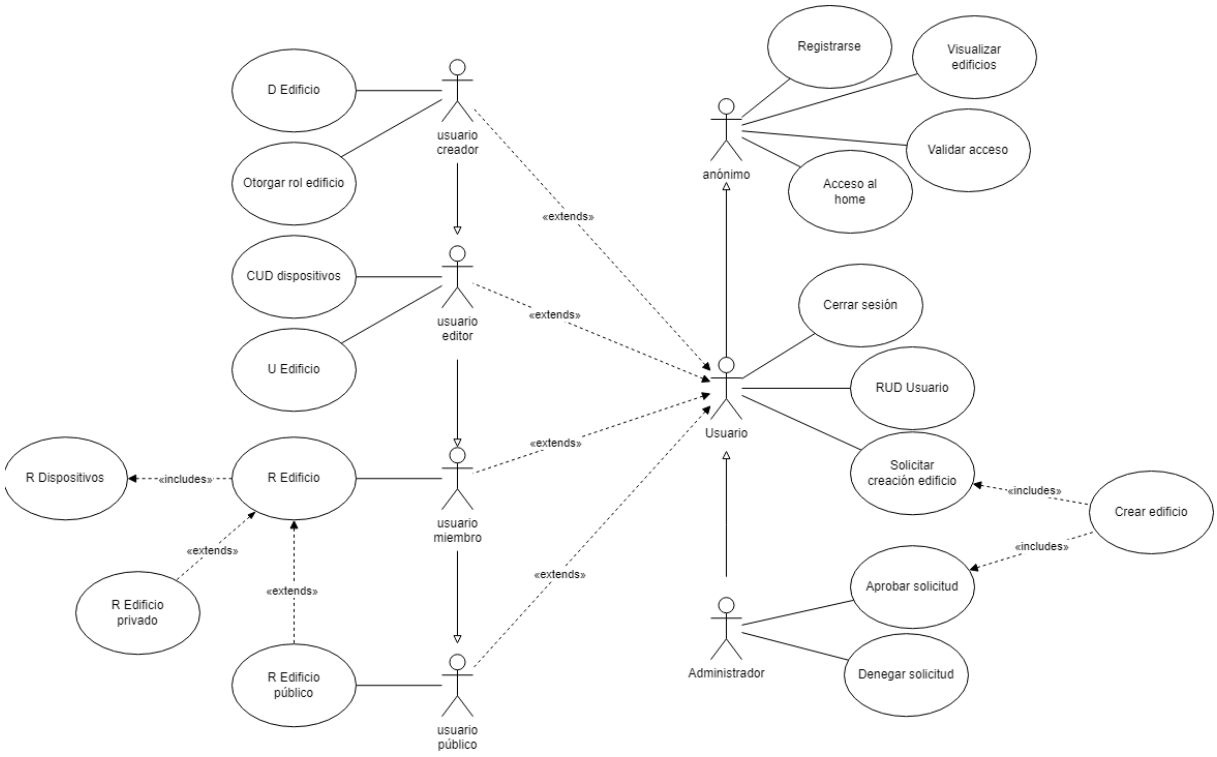


Figura 7: Diagrama de casos de uso

RNF-FIW6	Integración con tecnologías de vanguardia	Debe ser compatible con tecnologías emergentes, como inteligencia artificial o el aprendizaje automático, para poder ofrecer soluciones de IoT avanzadas.

4.4. Casos de uso

A continuación se describirán los casos de uso del sistema. En la figura 7 se observa un diagrama que relaciona los casos de uso del sistema.

CU-01	Registrarse
Versión	1.0
Dependencias	▪ RF-USU4 Registro en la aplicación
Precondición	Ninguna
Descripción	Cualquier usuario podrá registrarse en la aplicación.

Secuencia normal	1. El usuario accede al *Home* 2. El usuario selecciona el botón de registrarse para navegar al formulario de registro. 3. El usuario rellena el formulario de registro. 4. El usuario selecciona el botón de enviar formulario.
Postcondición	El usuario consigue una cuenta con la que podrá acceder a funcionalidades adicionales.
Excepciones	1. Las credenciales de autenticación ya existen. *a*) El sistema notificará al usuario con un mensaje. *b*) Se cancela el caso de uso. 2. La contraseña no cumple las políticas de seguridad. *a*) El sistema notificará al usuario con un mensaje. *b*) Se cancela el caso de uso.
Comentarios	Ninguno

CU-02	Visualizar listado de edificios
Versión	1.0
Dependencias	▪ RF-EDI1 R edificios ▪ RF-EDI7 Ordenación en el listado de edificios ▪ RF-EDI8 Filtrado en el listado de ordenación de edificios
Precondición	Ninguna
Descripción	Cualquier usuario registrado o no podrá visualizar un listado de los edificios.

Secuencia normal	1. El usuario accede al *Home*. 2. El usuario puede ver el listado de edificios en el *Home*. 3. El usuario puede filtrar el listado rellenando un formulario de filtros para buscar el edificio deseado.
Postcondición	El usuario obtiene un listado de edificios filtrado bajo su criterio.
Excepciones	1. No existe ningún edificio con el filtro seleccionado. *a*) El sistema notificará al usuario con un mensaje. *b*) Se cancela el caso de uso.
Comentarios	Ninguno

CU-03	Validar acceso
Versión	1.0
Dependencias	▪ RF-USU5 Validación del acceso
Precondición	Estar registrado.
Descripción	Cualquier usuario previamente registrado se podrá loguear en la aplicación usando las credenciales previamente proporcionadas en el registro.

Secuencia normal	1. El usuario accede al *Home*. 2. El usuario selecciona en Iniciar sesión. 3. El usuario navega a un formulario para rellenar con los datos para autenticarse. 4. El usuario introduce sus credenciales en el formulario. 5. El usuario envía el formulario. 6. El usuario regresa al *Home* con su sesión validada.
Postcondición	El usuario obtiene un token de usuario validado por el sistema.
Excepciones	1. Las credenciales proporcionadas no son correctas. *a*) El sistema notificará al usuario con un mensaje. *b*) Se muestra de nuevo el formulario. Permitiendo al usuario modificar los datos para reintentarlo hasta 3 veces más. *c*) Si sigue fallando, se muestra una opción de recuperar contraseña.
Comentarios	Ninguno

CU-04	Acceso al *Home*
Versión	1.0
Dependencias	▪ RF-INI1 Acceso a la página de inicio.
Precondición	Ninguna.
Descripción	Cualquier usuario, logueado o no, puede acceder a la página principal (*Home*).

Secuencia normal	1. Cuando el usuario accede a la web, accede al *Home* directamente. 2. Desde cualquier sitio de la web se puede acceder al *Home* seleccionando el logo ubicado en la barra superior.
Postcondición	Ninguna
Excepciones	Ninguna
Comentarios	Ninguno

CU-05	Cerrar sesión
Versión	1.0
Dependencias	▪ RF-USU6 Cerrar sesión.
Precondición	Haber iniciado sesión previamente y que esté activa.
Descripción	Cualquier usuario logueado podrá cerrar sesión.
Secuencia normal	1. El usuario selecciona el botón de cerrar sesión en la barra superior, visible desde cualquier parte de la web. 2. Se invalida la sesión del usuario.
Postcondición	La sesión del usuario se cierra y queda invalidada.
Excepciones	1. La sesión del usuario que se intenta cerrar no existe o ya está cerrada. *a*) El sistema notificará al usuario con un mensaje.
Comentarios	Ninguno

CU-06	RUD usuario
Versión	1.0

Dependencias	▪ RF-USU1 R de Usuarios. ▪ RF-USU2 U de Usuarios. ▪ RF-USU3 D de Usuarios.
Precondición	Tener un usuario autenticado en la aplicación.
Descripción	Cualquier usuario logueado podrá visualizar sus datos, modificarlos o borrar su cuenta. También podrá buscar los perfiles de otras personas.
Secuencia normal	1. El usuario se loguea con su cuenta. 2. El usuario selecciona la sección de datos del perfil. 3. El usuario puede ver su información. 4. El usuario puede seleccionar .[ed]itar datos"para modificar alguno. 5. El usuario puede seleccionar "borrar perfil"para eliminar su perfil.
Postcondición	El usuario ha modificado su o lo ha eliminado.
Excepciones	Ninguna
Comentarios	Ninguno

CU-07	Solicitar creación edificio
Versión	1.0
Dependencias	▪ RF-EDI4 Solicitud de creación de edificio
Precondición	Tener un usuario autenticado en la aplicación.

Descripción	Los datos de los edificios deben ser verificados para evitar la inserción de datos falsos o inapropiados. Para ello, primero un usuario solicita la creación del edificio.
Secuencia normal	1. El usuario selecciona el botón de Crear nuevo Edificio. 2. El usuario navega a un formulario con la información necesaria para el edificio. 3. El usuario rellena el formulario. 4. El usuario envía el formulario.
Postcondición	Se ha registrado la solicitud de creación de edificio.
Excepciones	Ninguna
Comentarios	Ninguno

CU-08	Crear edificio
Versión	1.0
Dependencias	1. Solicitar la creación de edificio. 2. Ser aprobada por un administrador.
Precondición	Para poder crear un edificio en el sistema se tiene que: 1. Solicitar la creación de edificio. 2. Ser aprobada por un administrador.
Descripción	Para crear un edificio en el sistema primero debe ser validada por un administrador, por lo que se debe solicitar su creación previamente.

Secuencia normal	1. El usuario solicita la creación del edificio con sus respectivos datos. 2. Un administrador debe revisar la solicitud previamente y aprobarla si está de acuerdo con los datos.
Postcondición	Se registra en el sistema los datos del edificio.
Excepciones	1. El administrador deniega la creación del edificio. *a*) El edificio se elimina del sistema.
Comentarios	Ninguno

CU-09	Aprobar solicitud
Versión	1.0
Dependencias	▪ RF-EDI5 Aceptar solicitud creación de edificio
Precondición	Debe existir una solicitud de edificio y el usuario debe ser administrador de la aplicación.
Descripción	La creación de edificios debe estar regulada por motivos de seguridad. Un administrador debe revisar las solicitudes que se realicen y ser validadas antes de ser visible por el resto de usuario de la aplicación.

Secuencia normal	1. El administrador accede al menú de administración. 2. El administrador accede a la sección de solicitudes de edificios pendientes. 3. El administrador selecciona de la lista de solicitudes pendientes la que desee revisar. 4. El administrador revisa los datos proporcionados para ese edificio. 5. El administrador está de acuerdo con los datos del edificio y selecciona en el botón para aprobarla. 6. El sistema notifica al administrador que la solicitud se ha realizado con éxito 7. El administrador regresa automáticamente a la lista de solicitudes pendientes.
Postcondición	Los datos del edificio se han marcado como válidos y los usuario pueden tener acceso a él.
Excepciones	Ninguna
Comentarios	Ninguno

CU-10	Denegar solicitud
Versión	1.0
Dependencias	▪ RF-EDI6 Denegar solicitud creación de edificio.
Precondición	Debe existir una solicitud de edificio y el usuario debe ser administrador de la aplicación.

Descripción	La creación de edificios debe estar regulada por motivos de seguridad. Un administrador debe revisar las solicitudes que se realicen y ser validadas antes de ser visible por el resto de usuario de la aplicación.
Secuencia normal	1. El administrador accede al menú de administración. 2. El administrador accede a la sección de solicitudes de edificios pendientes. 3. El administrador selecciona de la lista de solicitudes pendientes la que desee revisar. 4. El administrador revisa los datos proporcionados para ese edificio. 5. El administrador no está de acuerdo con los datos del edificio y selecciona en el botón para denegarla. 6. El sistema notifica al administrador que la solicitud se ha realizado con éxito. 7. El administrador regresa automáticamente a la lista de solicitudes pendientes.
Postcondición	Los datos del edificio se han marcado como válidos. Los usuarios no tendrán acceso a él.
Excepciones	Ninguna
Comentarios	Ninguno

CU-11	D edificio
Versión	1.0
Dependencias	▪ RF-EDI3 D edificios.

Precondición	Tener un edificio creado. Y ser el creador del mismo.
Descripción	Caso de uso relacionado con el borrado de un edificio.
Secuencia normal	1. El usuario creador del edificio accede a los datos del edificio. 2. El usuario accede a la sección de modificar datos. 3. El usuario busca la acción borrar edificio. 4. El sistema notifica al usuario que va a realizar una acción que no se puede deshacer. 5. El usuario confirma que quiere borrar los datos. 6. Se borran los datos del edificio. 7. El usuario regresa al *Home*.
Postcondición	Los datos del edificio se han borrado.
Excepciones	1. El usuario no es el creador del edificio. *a*) El sistema notifica al usuario que no tiene permiso para realizar esa acción. 2. El edificio no existe. *a*) El sistema notifica al usuario que el edificio no existe.
Comentarios	Ninguno

CU-12	Otorgar rol edificio
Versión	1.0
Dependencias	▪ RF-ROL1 Otorgar rol de edificio.
Precondición	Ser el creador de un edificio.

Descripción	Caso de uso relacionado con la concesión de roles dentro de un edificio.
Secuencia normal	1. El creador del edificio accede a los datos del edificio. 2. El creador accede a la sección de modificar el edificio. 3. El creador accede la sección de roles. 4. El creador accede a modificar el listado de roles. 5. El creador selecciona añadir nuevo usuario. 6. El creador indica el usuario y el rol que desea otorgarle. 7. El creador confirma la elección.
Postcondición	Se le ha otorgado un rol especial dentro de un edificio a un usuario.
Excepciones	Ninguna
Comentarios	Ninguno

CU-13	CUD dispositivos
Versión	1.0
Dependencias	▪ RF-DIS1 C dispositivos. ▪ RF-DIS3 U dispositivos. ▪ RF-DIS4 D dispositivos.
Precondición	Ninguna
Descripción	Dentro de un edificio se podrán incluir dispositivos IoT que interactúen con otros usuarios, entendiendo por interacción la lectura de datos o activarlos para que realicen alguna función.

Secuencia normal	1. El creador del edificio accede a los datos del edificio. 2. El creador del edificio accede a los sensores del edificio. 3. El creador selecciona el botón registrar nuevo dispositivo. 4. El creador rellena el formulario con la información del dispositivo. 5. El creador confirma los datos del dispositivo.
Postcondición	Ninguna
Excepciones	Ninguna
Comentarios	Ninguno

CU-14	U Edificio
Versión	1.0
Dependencias	▪ RF-EDI2 U edificios.
Precondición	Pertenecer a un edificio con el rol de editor o creador.
Descripción	Modificación de los datos de un edificio.
Secuencia normal	1. El creador del edificio accede a los datos del edificio. 2. El creador modifica los datos correspondientes del edificio. 3. El creador confirma la modificación pulsado el botón de guardar los cambios.
Postcondición	Ninguna

Excepciones	1. El usuario no es editor o creador del edificio. *a*) El sistema notifica al usuario que no tiene permiso para realizar esa acción. 2. El edificio no existe. *a*) El sistema notifica al usuario que el edificio no existe.
Comentarios	Ninguno

CU-15	R Edificio
Versión	1.0
Dependencias	▪ RF-EDI1 R edificios.
Precondición	Ninguna
Descripción	Se refiere al caso de uso relacionado con ver los detalles de un edificio, incluidos sus dispositivos.
Secuencia normal	1. El usuario selecciona el edificio del que desea conocer más información. 2. El usuario viaja a una vista donde se ven todos los detalles públicos del edificio.
Postcondición	Ninguna
Excepciones	Ninguna
Comentarios	Ninguno

CU-16	R dispositivos
Versión	1.0
Dependencias	▪ RF-DIS2 R dispositivos.

Precondición	Ninguna
Descripción	Se refiere al caso de uso relacionado con visualizar los detalles de un dispositivo.
Secuencia normal	1. El usuario accede a los detalles de un edificio. 2. El usuario visualiza el listado de dispositivos. 3. El usuario puede seleccionar un dispositivo y ver los detalles del mismo.
Postcondición	Ninguna
Excepciones	Ninguna
Comentarios	Ninguno

CU-17	R edificio privado
Versión	1.0
Dependencias	▪ RF-EDI1 R edificios.
Precondición	El usuario debe tener rol de miembro en un edificio para poder visualizar los datos privados del mismo.
Descripción	Datos privados de un edificio.
Secuencia normal	1. El usuario accede a los datos del edificio. 2. El usuario accede a la pestaña de datos privados del edificio.
Postcondición	Ninguna
Excepciones	Ninguna
Comentarios	Ninguno

CU-18	R Edificio público
Versión	1.0

Dependencias	▪ RF-EDI1 R edificios.
Precondición	Ninguna
Descripción	Datos privados de un edificio.
Secuencia normal	1. El usuario accede a los datos del edificio.
Postcondición	Ninguna
Excepciones	Ninguna
Comentarios	Ninguno

4.5. Metodología de trabajo

En esta sección vamos a hablar sobre la metodología de trabajo que se va a seguir para la realización del TFG.

4.5.1. Adaptación de *Scrum*

El desarrollo de este TFG se llevó a cabo utilizando una adaptación de la metodología ágil conocida como *Scrum*. La elección de esta metodología se fundamentó en su capacidad para ofrecer un marco de trabajo flexible y adaptable, permitiendo una gestión eficiente de los procesos involucrados en la investigación y desarrollo del proyecto.

Fundamentos de *Scrum* *Scrum* es un enfoque iterativo e incremental que prioriza la colaboración del equipo, la adaptación continua y la entrega de valor de forma iterativa. Se basa en roles definidos, eventos y artefactos que facilitan la planificación, ejecución y revisión del trabajo. Como esta metodología está orientada al desarrollo en equipo, no se aplicará de manera estricta, sino adaptándola a las necesidades que se vayan detectando durante el desarrollo del proyecto. En realidad, el propio *Scrum* incentiva la realización de reuniones de retrospectivas orientado a adaptar a la propia metodología a las necesidades del equipo y del proyecto.

Implementación en el TFG Durante el desarrollo de este trabajo, se aplicaron los principios de *Scrum* de la siguiente manera:

- ***Sprints* y Planificación**: El proyecto se dividió en iteraciones llamadas *sprints*, cada uno con una duración fija de un mes. Antes de cada *sprint*, se dedicó un tiempo de planificación en la que se identificaron las tareas a realizar y se establecieron los objetivos a cumplir al finalizar el periodo.

- **Revisión y Retrospectiva**: Al finalizar cada *sprint*, se llevó a cabo una revisión del trabajo completado y se recopilaron los comentarios para futuras mejoras. Asimismo, se realizó una retrospectiva para evaluar el proceso y determinar qué aspectos podían ser optimizados en los siguientes *sprints*. Por ejemplo, inicialmente, los *sprint* fueron pensados para 2 semanas, pero tras la segunda retrospectiva, se identificó que para este proyecto podría ser más interesante realizar el *sprint* mensualmente.

El uso de *Scrum* ha facilitado una gestión eficaz del tiempo, una mayor visibilidad del progreso y una capacidad para responder ágilmente a las necesidades del proyecto. Además, una entrega iterativa de resultados, lo que ha contribuido significativamente al éxito y documentación de este TFG.

4.5.2. Herramientas de organización

Como herramienta de organización se ha decidido utilizar Jira Software. Se trata de una herramienta de organización destinada al desarrollo de sistemas software que permite organizar y relacionar los objetivos, casos de usos, y tareas en un sistema de tarjetas.

Jira dispone de un sistema de tarjetas organizadas con una jerarquía de tareas, subtareas e historias agrupadas en épicas que permiten tener una visión global del proyecto. Dentro de cada tarjeta se pueden poner descripciones, estimaciones y comentarios oportunos para llevar un registro y poder ver la cuantificar el coste y el avance del trabajo, como se puede observar en la figura 8.

En la figura 9 podemos observar un subconjunto de las tareas definidas para el proyecto, organizadas por épica, sin embargo, existen otras vistas para visualizar el listado, como el diagrama de Gantt, como se muestra en la figura 12 o el *backlog*, donde se pueden visualizar las tareas sin hacer. Desde la figura 10, se pueden crear los *sprints* y organizar las tareas dentro de los mismos. Al finalizar cada *sprint* se pueden ver estadísticas para analizar el avance, ver la cantidad de puntos de *sprint* que puedo abarcar por *sprint*, reflexionar sobre las dificultades

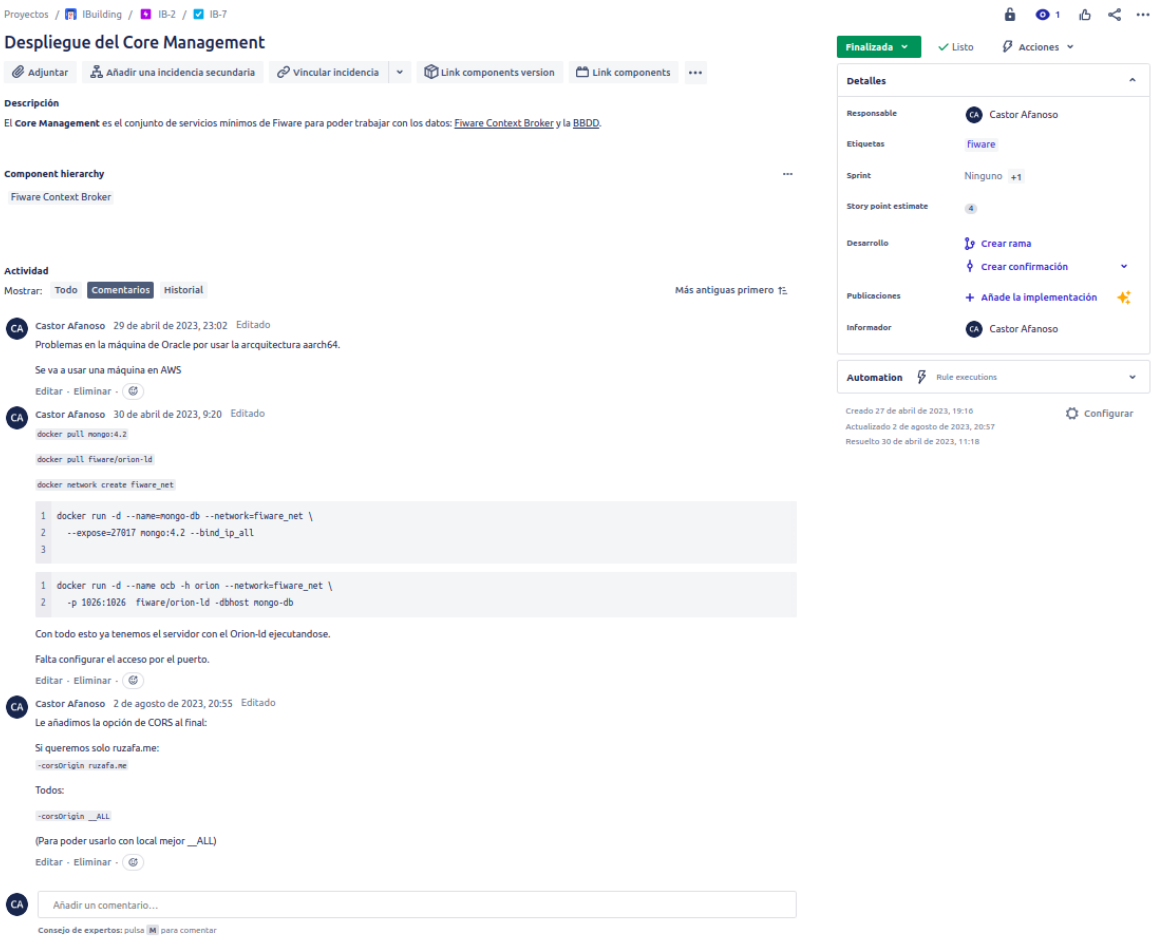


Figura 8: Descripción de una tarea

del *sprint*, posibles mejoras etc. En la figura 11, se puede observar un ejemplo de una iteración del proyecto.

4.5.3. Estimaciones de tareas

Los casos de uso han sido analizados y separados en tareas pequeñas e independientes que han sido descritas, categorizadas y puntuadas con puntos de *sprint*, en ingles, *sprint points*: un sistema de puntuación basado en la complejidad de la tarea. Una mayor puntuación significa que la tarea será más complicada de abordar, conllevará más pasos, mayor inversión de tiempo y tendrá una mayor incertidumbre acerca del tiempo de ejecución de la misma y su alcance.

No existe una relación directa entre tiempo estimado y puntos de *sprint*, sin embargo si se va haciendo un estudio de la relación entre tiempo empleado para la realización de cada tarea y los puntos asignados, se puede sacar una media para determinar cuánto tiempo tardará una

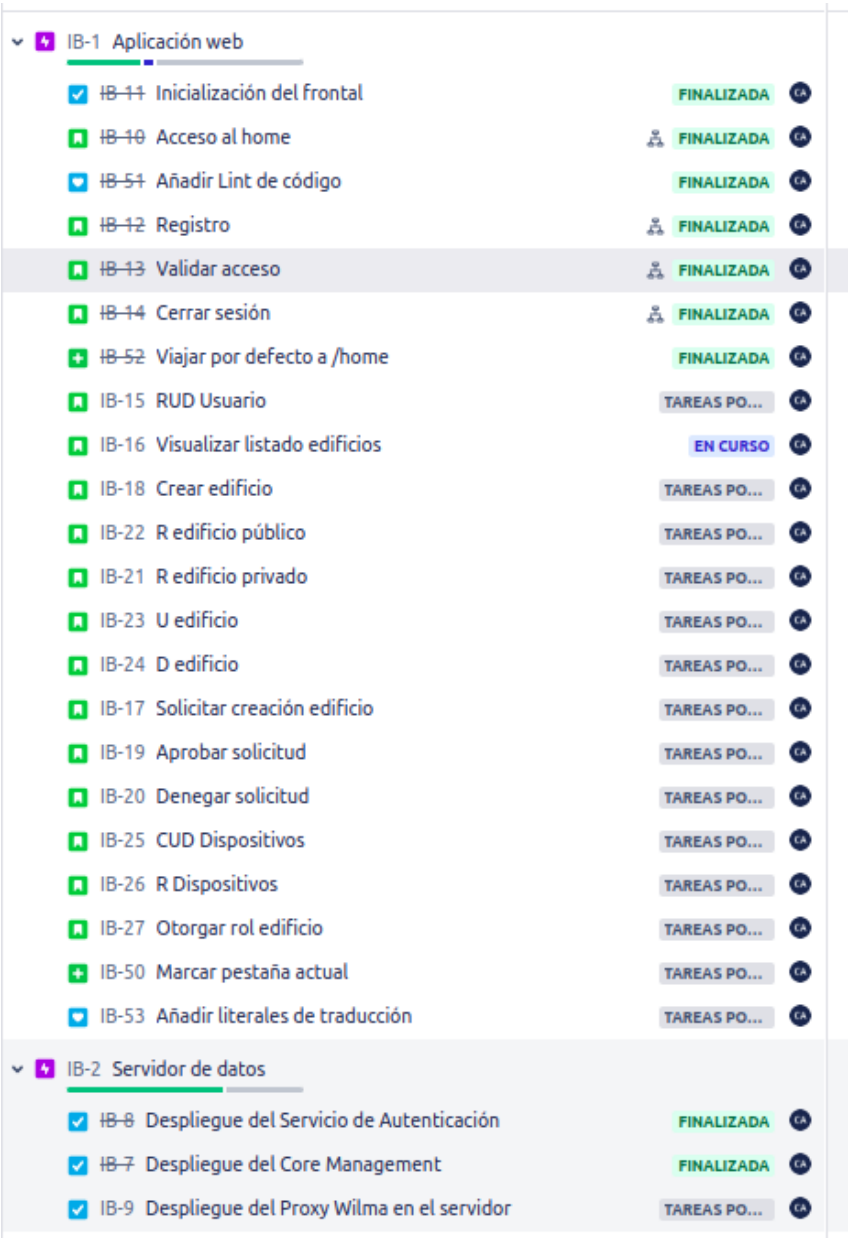


Figura 9: Listado de tareas

persona en realizar una tarea. Esta media será individual por cada persona y por cada proyecto.

Una vez finalizado una cantidad de *sprints*, se puede determinar cuantos puntos es capaz de abarcar una persona por *sprint*, esta información resulto útil para determinar cuántas tareas asignar en los *sprints* posteriores.

Si una tarea recibe una cantidad de puntos demasiado elevada, significa que hay mucha incertidumbre sobre la misma y abarca demasiado, es un indicativo de que esa tarea se puede subdividir en tareas más pequeñas, que son las que deben puntuarse y ejecutarse.

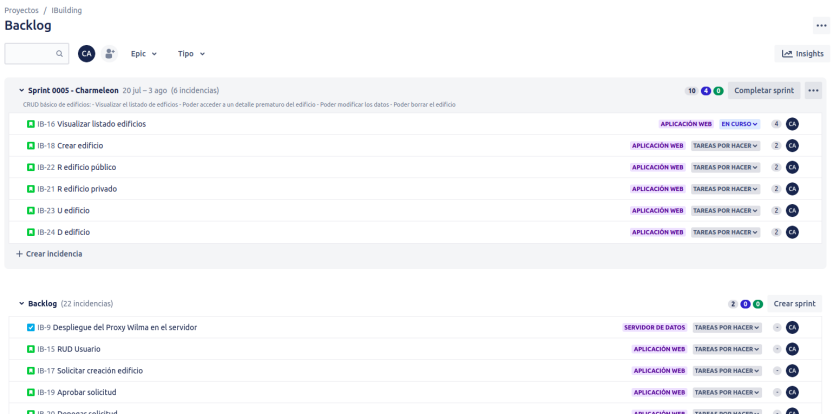


Figura 10: Backlog del proyecto con tareas volcadas en un *sprint*

Figura 11: Resultados de un *sprint*

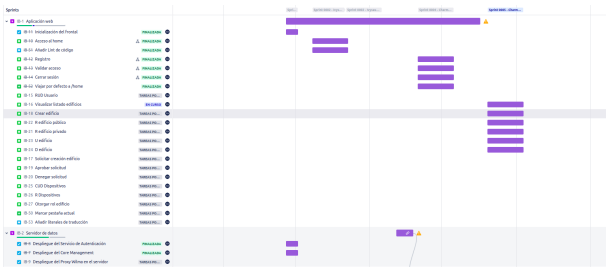


Figura 12: Diagrama de Gantt

Existen diversas formas de asignar estos puntos, por ejemplo, utilizando la sucesión de Fibonacci: 1, 2, 3, 5, 8, etc. O por potencias de 2: 1, 2, 4, 8, etcétera, que fue la que se utilizó en este proyecto. Si una tarea recibía una estimación de 8 puntos, se debía de intentar separar en tareas más pequeñas.

4.6. Planificación de *sprints*

En esta sección se van a enumerar los *sprints* que se planificaron para la realización de este TFG.

- **Primer *sprint***: Inicialización del frontal y despliegues de los servicios Orion-LD e `Identity Manager`.
- **Segundo *sprint***: Acceso al *Home* y enrutamiento del frontal.
- **Tercer *sprint***: Implementación del protocolo OAuth2.0 con el frontal y Orion.
- **Cuarto *sprint***: Desarrollo del frontal para operar con la API de Orion-LD y mostrar la información otorgada por el mismo usando el modelo de datos de SAREF en el frontal: Edificios, Espacios de edificios y sensores.
- **Quinto *sprint***: Desarrollo de un sensor en Arduino para generar medidas para almacenar en el sistema.

4.7. Decisiones de diseño

En esta sección se comentarán las decisiones de diseño principales, explicando porqué se eligen unas soluciones frente a otras, comparándolas o porqué no se descartará de la solución final.

4.7.1. Uso de SAREF

Aunque FIWARE recomienda el uso de modelos de datos armonizados, hemos preferido hacer uso de SAREF, al ser un modelo más completo y mejor documentado. Al estar más completo no será necesario extender el modelo de datos, por lo que simplifica la tarea por este punto.

4.7.2. Uso de la API NGSI-LD y Orion-LD

Como se explica en la sección de "Estado del arte", usar los datos armonizados, o la ontología SAREF, es una práctica recomendada si queremos resolver el problema de estandarización,

ya que usamos un modelo de datos reconocido internacionalmente y que, además, es interoperable con otros modelos de datos existentes. Pero para que esto sea efectivo, hay que aplicar el Linked Data, por lo que, si se estudian ambas versiones, vemos que la API que nos resulta más apropiado en este caso, sería Orion-LD, al implementar el Linked Data.

Por otro lado, si queremos que el servicio esté "Basado en FIWARE", deberemos elegir un Context Broker, al ser el Generic Enabler obligatorio. De las alternativas expuestas en la tabla 1, podemos restringirnos a la que más se adapte a las necesidades:

- El tamaño no debe ser demasiado pesado, ya que no vamos a necesitar gestionar una cantidad muy grande de datos. Además de no tener los recursos necesarios para desplegar servicios demasiados grandes.
- Como queremos trabajar con NGSI-LD, las implementaciones que no soporten esta API, no nos van a interesar.
- Para el ámbito de pruebas en el que nos encontramos, no será necesario ninguna implementación en el borde.

Teniendo en cuenta estos puntos, Orion-LD Context Broker es el que mejor se adapta a nuestras especificaciones para un nodo en la Nube. Si fuéramos a implementar una solución aprovechando la computación en el borde, sería interesante desplegar en los nodos del borde el `ThinBroker`.

4.7.3. Seguridad

Para proporcionar seguridad a los servicios utilizaremos el Generic Enabler Keyrock [14] que implementa el protocolo de OAuth2.0, detallado anteriormente en la sección de tecnologías. El flujo que se utilizará será el de concesión de código de autorización, ya que es el flujo tradicional y se adapta muy bien a la casuística.

Acompañado de Keyrock, sería altamente recomendable desplegar Wilma, ya que, como se menciona anteriormente, OAuth2.0 dictamina la autorización, pero no la aplica, y no es seguro delegar en el cliente la responsabilidad de aplicar la sentencia del PDP. `AuthZForce` también sería interesante para, por ejemplo, implementar el caso de uso RF-DIS2, que nos

[14] https://fiware-idm.readthedocs.io/en/latest/index.html

permite controlar el acceso a nivel de rol dentro de un edificio, cosa que Keyrock no permite, al no tener una granularidad de control tan fina.

En la solución que se va a implementar no se desplegarán ni Wilma ni AuthZForce debido a que la complejidad de los despliegues y configuraciones se salen del ámbito de este TFG, además a la hora de desplegar los sistemas, no tenemos suficientes recursos para levantarlo. Sí se mencionan y estudian para plantear una solución final y más realista, ya que el no usarlo conlleva graves consecuencias a nivel de seguridad: No podremos verificar si el dispositivo es real o fraudulento y no podremos validar contra ningún PDP si el usuario tiene acceso al recurso solicitado.

Otra solución alternativa, y más moderna, podría ser el uso de API-Gateways, como Kong, el cual actúa como punto único de entrada al sistema. De esta forma, Kong podría actuar como PEP, permitir aplicar políticas de seguridad, como ajustar el ratio de llamadas, o añadir monitorización de tráfico, incluyendo, incluso, *dashboards*.

4.7.4. Computación en el borde

Con Fog Flow se podría añadir una solución para la implementación en la niebla. Para ello habría que añadir dos nodos: El nodo del borde y el nodo del servidor, ambos con una serie de microservicios orientados a gestionar la demanda de dispositivos.

La implementación de este sistema es complejo, así que no se llevará acabo, pues se necesitarían dos servidores, con diversos servicios levantados en Docker, más capacidad extra para levantar los workers y los operadores, que no son más que contenedores que realizan una operación, sin embargo, sabemos la utilidad a nivel de escalabilidad y eficiencia ecológica que suponen, por lo que es una alternativa digna de contemplar para una solución realista.

4.7.5. IoT Agent

Los IoT Agents permiten conectar dispositivos que utilizan interfaces diferentes a NGSI-LD. En nuestro caso, para facilitar el desarrollo del dispositivos físico, decidimos utilizar UltraLight2.0 como interfaz de comunicación, ya que es una interfaz sencilla y ligera, sin embargo, por como funciona el IoT Agent para UltraLight, ha sido imposible implementarlo en la solución real, ya que, como veremos en la implementación, va actualizando el valor del sensor en una entidad que hace referencia al sensor y la ontología SAREF nos dice que cuando re-

gistramos una medida nueva, se debe crear una entidad de tipo “Measurement” con el valor y asociarlo con la entidad del sensor.

Como estos dos aspectos no son compatibles, había que decidir entre usar SAREF, usar IoT Agent UltraLight, implementar un IoT Agent personalizado con las características que necesitamos, o implementar el sensor usando otro protocolo.

Finalmente, por simplificar, la opción elegida fue implementar el sensor utilizando NGSI-LD directamente, pero sabemos que en una solución realista deberíamos trabajar con un IoT Agent personalizado.

4.8. Arquitectura general

Debido a la complejidad de la solución, se hablarán de dos implementaciones: Una realista, la cual podemos observar en la figura 14, y una simplificada, en la figura 13. La solución realista dispone de muchos aspectos que quedan fuera del ámbito del TFG, por lo que es preferible realizar una implementación más sencilla, practicable y controlada y exponer de forma teórica aquellos aspectos que no se abordarán pero que son importantes tener en cuenta aunque no se hable de ellos en la práctica. En todo momento se aclarará qué apartados se implementaron realmente y cuales se mencionaron de manera teórica.

¿Porqué se expone una implementación real y una simplificada? A la hora de intentar desarrollar todos los requisitos y los casos de uso, algunos de ellos requerían el despliegue de microservicios adicionales y la complejidad del sistema requería muchas configuraciones extras, tiempo de aprendizaje y pruebas extras, cambios en la arquitectura, en el código del frontal... Por lo que decidimos dejarlo fuera del ámbito del proyecto, pero sin dejar de mencionarlo. Se comenta y estudian los elementos de una “solución realista” porque podría cubrir todos los requisitos descritos, cosa que la solución implementada, en la práctica, no lo puede hacer.

Ambas arquitecturas están organizadas en torno a tres grandes bloques: La infraestructura de datos, denominada `IBuilding`, el consumo de la API con una interfaz gráfica denominada `Frontal Building` y la alimentación del sistema, con dispositivos físicos como `Person Counter`.

4.9. Solución Implementada

Si observamos la figura de izquierda a derecha, el primer bloque es `IBuilding`. Este bloque estará compuesto por todos los componentes de FIWARE que habrá que configurar y desplegar. Para esta solución solo hemos incluido el `Context Broker`, en concreto `Orion-LD` con una base de datos en `MongoDB`, ya que son los componentes mínimos necesarios para que se pueda considerar que está impulsado por FIWARE. Además, de un `Generic Enabler` para gestionar la seguridad: `Keyrock`, que nos ofrece el servicio de `OAuth2.0`. Este servicio irá acompañado de su base de datos en `MySQL`. Con todos estos componentes debidamente configurados y desplegados, ya tendremos disponible una API con la interfaz `NGSI-LD` que podrán ser usados por el resto de bloques.

El segundo bloque que se observa en la figura 13, es el de `Frontal Building`. Será una aplicación web que ofrecerá una interfaz gráfica para visualizar e interactuar con los datos de FIWARE. Este frontal lo desarrollaremos utilizando `NextJS`.

El último bloque de nuestra solución es el de "Devices". Este bloque representa los componentes IoT. En nuestro caso, implementaremos un sensor con Arduino que alimentará a la API. Este sensor será un contador para determinar cuantas personas hay en un área. Este dispositivo se conectará a la API usando `NGSI-LD`, pero en la solución realista, se proponen alternativas para poder conectar dispositivos que utilicen otro tipo de interfaces.

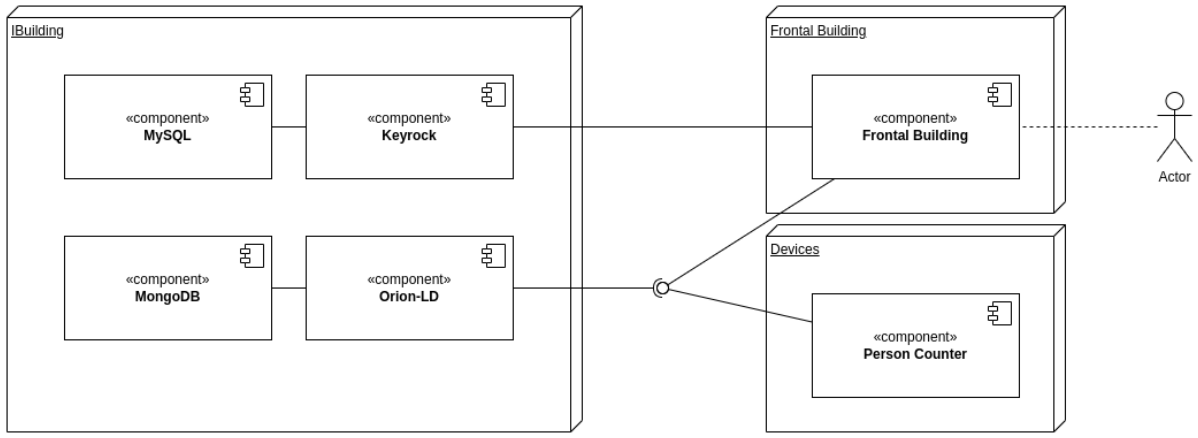


Figura 13: Diagrama de componentes del sistema implementado

4.10. Solución realista

En la figura 14 podemos observar los mismos tres bloques descritos anteriormente. Esta solución añade a la implementada otros componentes que por su complejidad no han podido ser implementados en la solución final.

Por un lado, en el bloque de `IBuilding`, además de los componentes de la solución simple (`Orion-LD` y `Keyrock`), tenemos los `IoT Agents` que actúan de adaptadores de protocolo. Existen adaptadores prefabricados que solo es necesario desplegar para empezar a utilizarlo. También se puede descargar el código fuente de unas plantillas para desarrollar uno personalizado. Como los prefabricados no se adaptaban a los requerimientos de la ontología SAREF, optamos por no usarlos, ya que no entra en los objetivos del TFG.

También podemos encontrar en este bloque `Wilma`, que actúa como proxy para mejorar la seguridad del sistema. Si nuestra API no va a estar expuesta a internet directamente, (es decir, solo en redes privadas para uso interno), no sería estrictamente necesario añadir esta capa de seguridad, pues controlamos todos los clientes que acceden y podemos realizar las comprobaciones de seguridad en estos, pero si se expone a internet para el uso de terceros, no se puede delegar en ellos la verificación de los permisos. Como podemos observar, `Wilma` actúa como proxy de `Orion-LD`, pero también actúa como proxy para los `IoT Agents`, aunque la una versión para los IoT Agents es diferente. En esta solución cualquier punto de entrada está protegido por una instancia de `Wilma`, se le conoce como securización por el *South Port*. En su lugar, también se podría asegurar por el *North Port* conectándolo con el mismo `Wilma` que se usa para acceder a `Orion-LD` directamente. Sin embargo, en este caso, por simplificar, ya que el objetivo principal no es la seguridad, se decidió no hacer uso de `Wilma` y revisar la seguridad en el frontal.

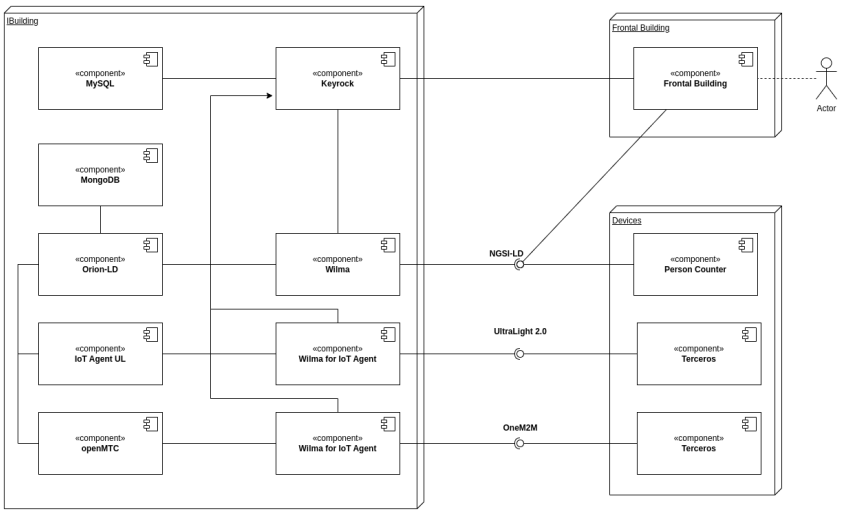


Figura 14: Diagrama de componentes de una propuesta más realista

5

Desarrollo del Proyecto

El objetivo de esta sección es explicar los pasos que se han seguido para desarrollar la solución propuesta en el apartado "diseño de la solución". Se mencionarán los aspectos más destacables, los problemas que se presentaron y de qué forma se abordaron. La estructura de esta sección está organizada en torno a las partes principales del TFG: La infraestructura de FIWARE, el desarrollo de un frontal y el desarrollo de un dispositivo IoT.

5.1. Despliegues de los servicios FIWARE

En la primera iteración se ha puesto el foco en los despliegues del *Context Broker* y el servicio de autenticación *Keyrock*. Para ello, la primera opción era hacer uso de FIWARE Lab, un servicio de FIWARE que facilita entornos de prueba con servicios ya desplegados. Sin embargo, era necesario solicitar acceso a FIWARE para poder acceder a estos entornos. La Universidad de Málaga poseía unas credenciales las cuales me fueron facilitadas para acceder al servicio, no obstante, estas credenciales estaban desfasadas y tuve que volver a solicitar acceso a estos entornos, sin éxito. Por lo tanto, hubo que hacer uso de otra alternativa: Realizar los despliegues en un servidor propio. Las opciones viables conocidas eran:

- `Oracle Cloud`: Ofrece 4 instancias de máquinas virtuales de manera gratuita.

- `Amazon Web Services` (AWS): La capa gratuita de AWS ofrece el servicio EC2 (del inglés, *Amazon Elastic Compute Cloud*) que ofrece capacidad de computación en la nube a través de máquinas virtuales.

5.1.1. Configuración de `Orion-LD` con Oracle

Desde la web de Oracle, una vez que nos hemos registrado, se puede acceder a la sección de instancias, como se muestra en figura 15 y, desde ahí, controlar las instancias que tengamos disponibles o crear nuevas, para lo cual habrá que rellenar un formulario con las características que deseemos de nuestra máquina virtual. Entre otras cosas, como se muestra en la figura 16, podemos configurar la ubicación de la máquina virtual, la imagen, claves de seguridad...

Figura 15: Menú de instancias de la nube de Oracle

Figura 16: Configuraciones de máquinas virtuales en `Oracle Cloud`

De entre todas las opciones de tipo de instancia, las opciones gratuitas eran:

- VM.Standard.A1.Flex que se ejecuta en aarch64 [15].

[15] también conocido como ARM

- VM.Standard.E2.1.Micro que se ejecuta en x86.

Aunque la arquitectura no fue relevante a la hora de elegir el tipo de instancia, al intentar levantar el Context Broker dio bastantes problemas, ya que FIWARE no es compatible con la arquitectura aarch64.

Haciendo pruebas de diversos tipos con Docker Compose, manualmente con Docker, compilando directamente del código fuente..., el error obtenido siempre era el mismo que se puede observar en la figura 17. Fueron varios intentos hasta llegar a la conclusión de que el problema se encontraba la arquitectura seleccionada.

```
orion-ld_1  | exec /usr/bin/orionld: exec format error
```

Figura 17: Error debido a la ejecución de FIWARE en una arquitectura aarch64

Con VM.Standard.E2.1.Micro se podría haber ejecutado sin problemas, ya que funciona sobre la arquitectura x86, sin embargo, por falta de información sobre el factor de las arquitecturas, la decisión en ese momento fue probar directamente con Amazon Web Services.

5.1.2. Configuración de Orion-LD con Amazon

Una vez registrado en la web de Amazon Web Services, se puede acceder a la creación de una instancia con EC2. Como se puede ver en la figura 18, la elección del sistema operativo fue Amazon Linux ya que está optimizado para las instancias de Amazon. Por otro lado, existe la posibilidad de seleccionar como arquitectura ARM o x86, que, debido al problema de compilación de FIWARE observado en el punto anterior, sabemos que hay que usar la arquitectura x86.

En el formulario de creación de la máquina virtual se pueden generar unas claves público-privada. Las clave privada generada se descarga codificada en PEM para enviar por SSH desde una terminal:

```
1 ssh -i "fiware-keys.pem" ec2-user@dominio
```

Por otro lado, también se puede configurar una lista de IP permitidas mediante los grupos de seguridad. Estos monitorizan el tráfico de la máquina virtual y rechaza los paquetes que no cumplan las políticas de seguridad establecidas de tal forma de que la máquina virtual no

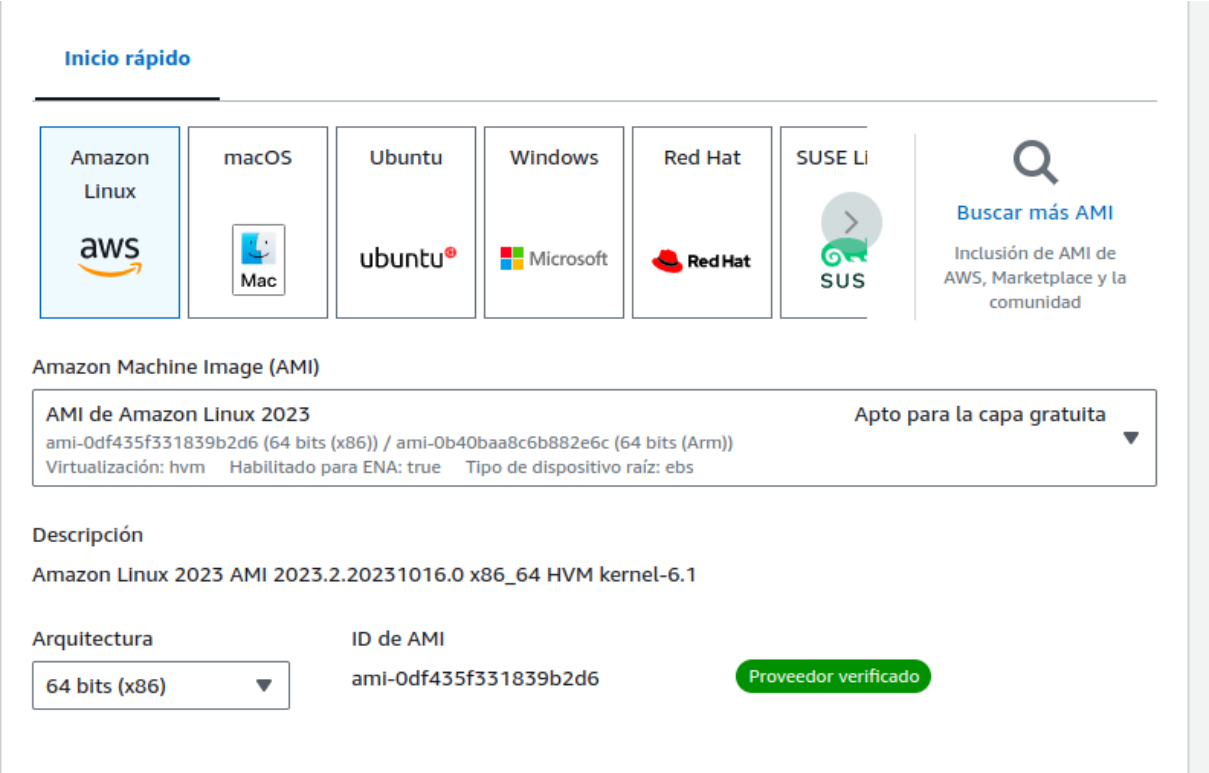


Figura 18: Pantalla de configuración de la instancia en AWS

llega a recibir esas peticiones calificadas como inseguras. En nuestro caso debemos permitir la conexión por SSH y los puertos para HTTP y HTTPS (80 y 443) para todos los usuarios.

Una vez creada la máquina tendremos información sobre ella en una vista con el listado de todas las instancias de EC2 como se observa en la figura 19. Desde esta vista podremos obtener la IP de la instancia para acceder por SSH, aunque también es posible acceder desde una terminal web, como se ve en la figura 20, que ofrece AWS de manera segura.

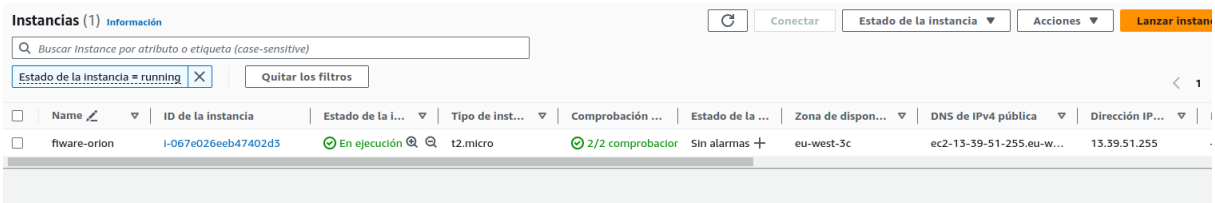


Figura 19: Listado de instancias EC2 disponibles en AWS

Una vez se ha accedido a la máquina virtual, es necesario realizar las labores de mantenimiento recomendadas, como actualizar las librerías del sistema. Como Amazon Linux está basado en CentOS hay que usar dnf como gestor de paquetes para actualizar e instalar las

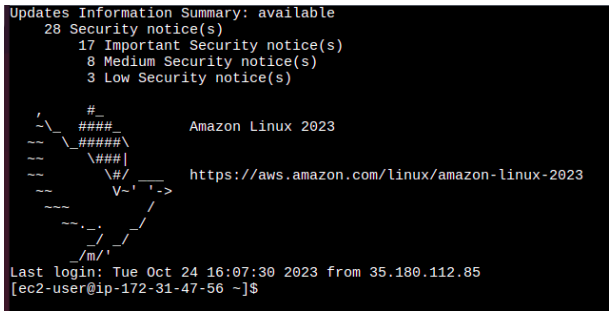


Figura 20: Terminal disponible desde la web de AWS.

librerías necesarias.

```
sudo dnf update
sudo dnf install docker
```

A continuación iniciamos el servicio de Docker y añadimos nuestro usuario al grupo de usuarios que pueden usar Docker.

```
sudo service docker start
sudo usermod -aG docker ec2-user
```

Una vez instalado Docker, procederemos a instalar Docker Compose. Para ello nos lo descargamos desde el repositorio oficial usando curl. Colocaremos la descargar en el path y le daremos permisos de ejecución. Una vez hecho esto podremos comprobar si se ha instalado correctamente, preguntando la versión de Docker Compose que tenemos.

```
sudo curl -L https://github.com/docker/compose/releases/download/1.22.0/
    docker-compose-$(uname -s)-$(uname -m) -o /usr/local/bin/docker-compose
sudo chmod +x /usr/local/bin/docker-compose
docker-compose version
```

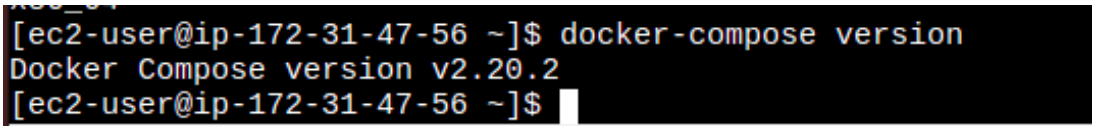


Figura 21: Versión de Docker Compose tras instalarla en la instancia de EC2

Con todo esto ya podemos ejecutar el archivo YML de `Orion-LD` que se puede ver en el Anexo A.

5.1.3. Configuración de dominio

Lo siguiente a configurar es un dominio para acceder a Orion sin usar la IP directamente. En este caso se ha utilizado Namecheap como proveedor de dominios. Para registrar un dominio con Namecheap hay que buscar el nombre deseado para comprobar la disponibilidad y precios. En este caso había disponible una oferta de un dominio terminado en ".me" gratuito durante un año. Para este proyecto se ha escogido "ruzafa.me".

Una vez seleccionado y activado el dominio, se pueden configurar subdominios para redirigir a la IP deseada. En nuestro caso, al tener una máquina con IP estática, se puede crear un subdominio para traducir a esta IP y así no es necesario hacer referencia a una IP fija en ninguna prueba. Una de las ventajas de esto es que si a futuro cambio la IP por el motivo que sea solo será necesario reconfigurar los registros de la DNS.

En la figura 22 se puede ver la entrada configurada con una IP de ejemplo, en la sección "DNS Avanzado" de Namecheap.

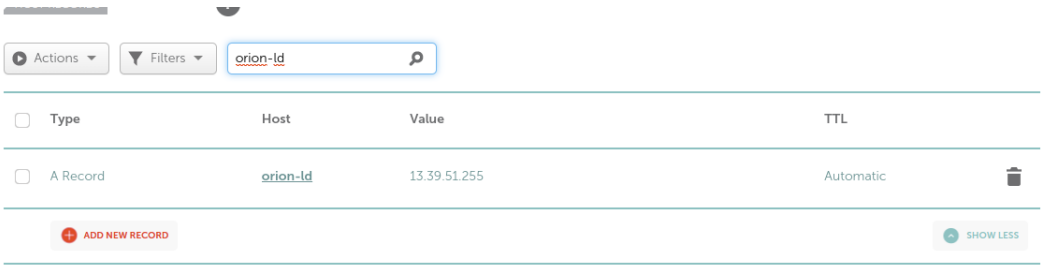


Figura 22: Entrada en los registros de la DNS de Namecheap para el subdominio del Orion-LD

Esta configuración no está completa para un despliegue real, ya que, en realidad, si quiero acceder al servicio tendría que utilizar la siguiente URL: http://orion.ruzafa.me:1026. Hemos expuesto el servicio de manera directa usando su puerto, lo cual conlleva diversos problemas de seguridad, además, utilizar HTTP sin ningún tipo de protección es peligroso.

Posteriormente, configuraremos un proxy inverso con Nginx. De esta forma podemos tener una capa más de seguridad sobre el tráfico. Podremos mandar todo el tráfico por el puerto 443 para que vaya por HTTPS y controlar el tráfico con el Nginx, actuando como intermediario entre el usuario y el servicio real en lugar de exponer cada servicio con su puerto directamente a internet. Solo estarán accesibles los puertos 443 y 80, que redirigirá al 443. De esta forma tendremos una URL de este estilo https://fiware.ruzafa.me/orion-ld, para acceder al Orion.

Así, para añadir más servicios en la misma instancia, solo hay que añadir la configuración necesaria en el Nginx con una nueva subruta para cada servicio, por ejemplo el servidor de autenticación Keyrock: https://fiware.ruzafa.me/keyrock.

5.1.4. Configuración de Keyrock

En el Anexo A se puede ver el archivo de Docker Compose utilizado para configurar el servicio de identidad para FIWARE.

A la hora de desplegarlo en la misma instancia de AWS que Orion, hubo problemas de recursos. El espacio de almacenamiento no era suficiente para albergar ambos servicios, por lo que hubo que desplegar Orion y Keyrock en máquinas diferentes, cada uno con su servidor de Nginx y su subdominio personalizado: "orion.ruzafa.me" y "keyrock.ruzafa.me".

Adicionalmente, para permitir la característica de enviar correos de confirmación, hubo que configurar un servidor SMTP Relay [16], como Brevo, que es una plataforma de marketing orientada al envío de correos masivos mediante plantillas que posee una API, accesible mediante claves de integración, para actuar de Relay y enviar correos automáticamente desde otros servicios. Para permitir a Keyrock mandar correos desde Brevo, se utilizó "mailer", un servicio desplegado con Docker que actúa como proxy entre el servidor externo de SMTP Relay y Keyrock. En la configuración de Docker Compose de mailer se encuentran las credenciales generadas en Brevo para el uso de la API. Véase la figura 23.

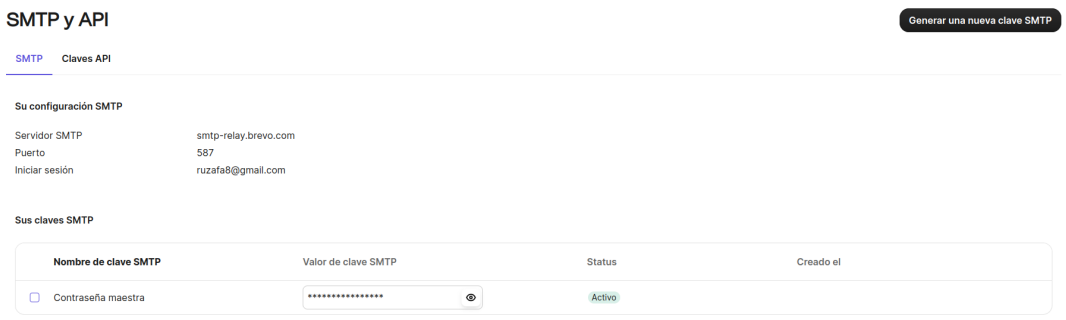


Figura 23: Configuración de la API para el envío de correos automáticos en Brevo

Para configurar los dominios desde los cuales enviar los correos, hay que acceder a confi-

[16] Un SMTP Relay es un tipo de servidor que se encarga del envío y recepción de correos electrónicos. Dentro del protocolo SMTP es una pieza importante, ya que son los servidores que mueven los correos entre los distintos dominios.

guración de Brevo, como se observa en la figura 24. Hay que incluir unas entradas en la DNS del dominio siguiendo las instrucciones que proporciona Brevo para verificar la posesión del dominio. En la figura 24 se pueden observar estas instrucciones de verificación. Los valores están ocultos por motivos de seguridad. También se pueden configurar los nombres de los remitentes, como se observa en la figura 25

Figura 24: Configuración de la DNS para verificar el dominio desde el servidor externo de SMTP Relay

5.1.5. Configuración de Nginx y certificados HTTPS

En esta sección describimos los pasos para instalar el servidor Nginx en la máquina virtual de AWS y cómo configurarlo para securizar los servicios de Keyrock y Orion-LD utilizando HTTPS.

Es conveniente proteger estos servicios con HTTPS por diferentes motivos:

- El servidor del frontal (desplegado en Vercel) se integrará con estos servicios, y por lo tanto hará llamadas para obtener los datos. Como Vercel ya nos ofrece los servicios

Figura 25: Configuración de nombre del remitente

mediante HTTPS con certificados generados automáticamente, hacer llamadas a servicios que utilicen HTTP provocaría el problema del contenido mezclado, conocido como *mixed context* en inglés, el cual nos dice que si desde un servicio que ofrece HTTPS llamamos a uno que utiliza HTTP estamos perdiendo toda la seguridad que HTTPS nos ofrecía. Como la seguridad se degrada, los navegadores rechazan este tipo de respuestas.

- Autentica al servidor, lo que asegura al cliente que la información sensible se envía al destinatario correcto y no a un servidor fraudulento que podría utilizarla de manera indebida.
- Proporciona confidencialidad, ya que los datos se envían cifrados y solo pueden ser accedidos por el servidor adecuado, evitando que agentes externos puedan interceptar o visualizar la información, incluyendo datos sensibles.
- Asegura la integridad de la información transmitida, ya que cualquier modificación o alteración realizada por agentes externos es detectada.

Let's Encrypt nos permite generar de una manera fácil certificados mediante una serie de comandos en la terminal que ejecutaremos tras instalar Nginx.

Otra problemática que nos encontramos es el de las CORS. FIWARE, en sus variables de entorno y flags de configuración, provee herramientas para configurar las cabeceras de las CORS con los hosts permitidos. El problema es que no funciona para la API NGSI-LD, aunque esto no se advierte en ninguna documentación de FIWARE. Se considera una característica todavía sin implementar en su repositorio de git. Para arreglarlo probamos a verificar que

estaba usando la versión estable más reciente, ya que periódicamente van sacando arreglos, pero el más reciente hasta la fecha no cuenta con este problema solucionado. Así que la otra alternativa es aprovechar el Nginx, que actúa de intermediario, para inyectar las cabeceras necesarias en las peticiones.

Para instalar Nginx fueron usados los siguientes comandos:

```
sudo dnf install -y nginx
sudo systemctl enable nginx.service
```

En los archivos de Nginx, en /etc/nginx/, hay que añadir la configuración necesaria para que actúe de proxy reverso y, por otro lado, las peticiones que envía Orion no tiene las cabeceras Access-Control-Allow-Origin, ni la de Access-Control-Allow-Headers que son las que informan al navegador de qué origen deben controlar que lleguen peticiones y qué cabeceras están permitidas.

Con proxy_pass indicamos a qué servicio debe redirigir el proxy inverso. En este caso, a un servicio en el mismo host que esté escuchando en el puerto 3000.

Para añadir cabeceras, se utiliza add_header. Como queremos más de un origen (dispondremos de dos entornos de desarrollo que se explicarán más adelante), vamos a configurarlo de forma dinámica, ya que el navegador solo admite uno. Con el map podemos hacer una expresión regular que evalúe el origen de la petición y si coincide con uno de los entornos, se asigna ese origen a la variable, que luego se asigna a la cabecera. Si llama cualquier otro origen, no cumplirá la expresión regular y tomará el valor por defecto, vacío, entonces la cabecera no tomará ningún valor y el navegador lo rechazará por defecto.

```
map $http_origin $allow_origin {
    ~^https?://(pre-)?building.ruzafa.me?$ $http_origin;
    # NGINX won't set empty string headers, so if no match, header is unset
        .
    default "";
}

location / {
    proxy_pass http://localhost:3000/;

    add_header Access-Control-Allow-Origin $allow_origin;
    add_header Access-Control-Allow-Headers *;
```

```
12
13      proxy_redirect      off;
14      proxy_set_header    Host            $host;
15      proxy_set_header    X-Real-IP       $remote_addr;
16      proxy_set_header    X-Forwarded-For
            $proxy_add_x_forwarded_forrelacionadas;
17      client_max_body_size        10m;
18      client_body_buffer_size     128k;
19      proxy_connect_timeout       90;
20      proxy_send_timeout          90;
21      proxy_read_timeout          90;
22      proxy_buffer_size           4k;
23      proxy_buffers               4 32k;
24      proxy_busy_buffers_size     64k;
25      proxy_temp_file_write_size 64k;
26  }
```

A continuación instalaremos el certificado con Let's Encrypt. Para ello utilizaremos el instalador de paquetes de Python, pip. Una vez instalado, situaremos el ejecutable del programa en el path del sistema para poder ejecutarlo con el comando "certbot", que iniciará un proceso en sus servidores para verificar nuestro Nginx como servidor.

```
pip install certbot certbot-nginx
sudo ln -s /opt/certbot/bin/certbot /usr/bin/certbot
sudo certbot --nginx
```

5.1.6. Administración de Keyrock

Una vez desplegado todo, podemos ver la interfaz gráfica de Keyrock e iniciar sesión con la cuenta de administración si queremos hacer labores de mantenimiento o registrarnos, recibiendo nuestro correo de confirmación.

Todos los estilos se pueden modificar y adaptar: El logo, los contenidos de los mensajes, correos, etc. No tocaremos nada sobre este tema, ya que no será relevante.

```
[ec2-user@ip-172-31-38-1 ~]$ sudo certbot --nginx
Saving debug log to /var/log/letsencrypt/letsencrypt.log
Enter email address (used for urgent renewal and security notices)
 (Enter 'c' to cancel): ruzafa8@gmail.com

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Please read the Terms of Service at
https://letsencrypt.org/documents/LE-SA-v1.3-September-21-2022.pdf. You must
agree in order to register with the ACME server. Do you agree?
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
(Y)es/(N)o: y

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Would you be willing, once your first certificate is successfully issued, to
share your email address with the Electronic Frontier Foundation, a founding
partner of the Let's Encrypt project and the non-profit organization that
develops Certbot? We'd like to send you email about our work encrypting the web,
EFF news, campaigns, and ways to support digital freedom.
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
(Y)es/(N)o: n
Account registered.

Which names would you like to activate HTTPS for?
We recommend selecting either all domains, or all domains in a VirtualHost/server block.
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
1: keyrock.ruzafa.me
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Select the appropriate numbers separated by commas and/or spaces, or leave input
blank to select all options shown (Enter 'c' to cancel):
Requesting a certificate for keyrock.ruzafa.me

Successfully received certificate.
```

Figura 26: Instalación del certificado con Let's Encrypt

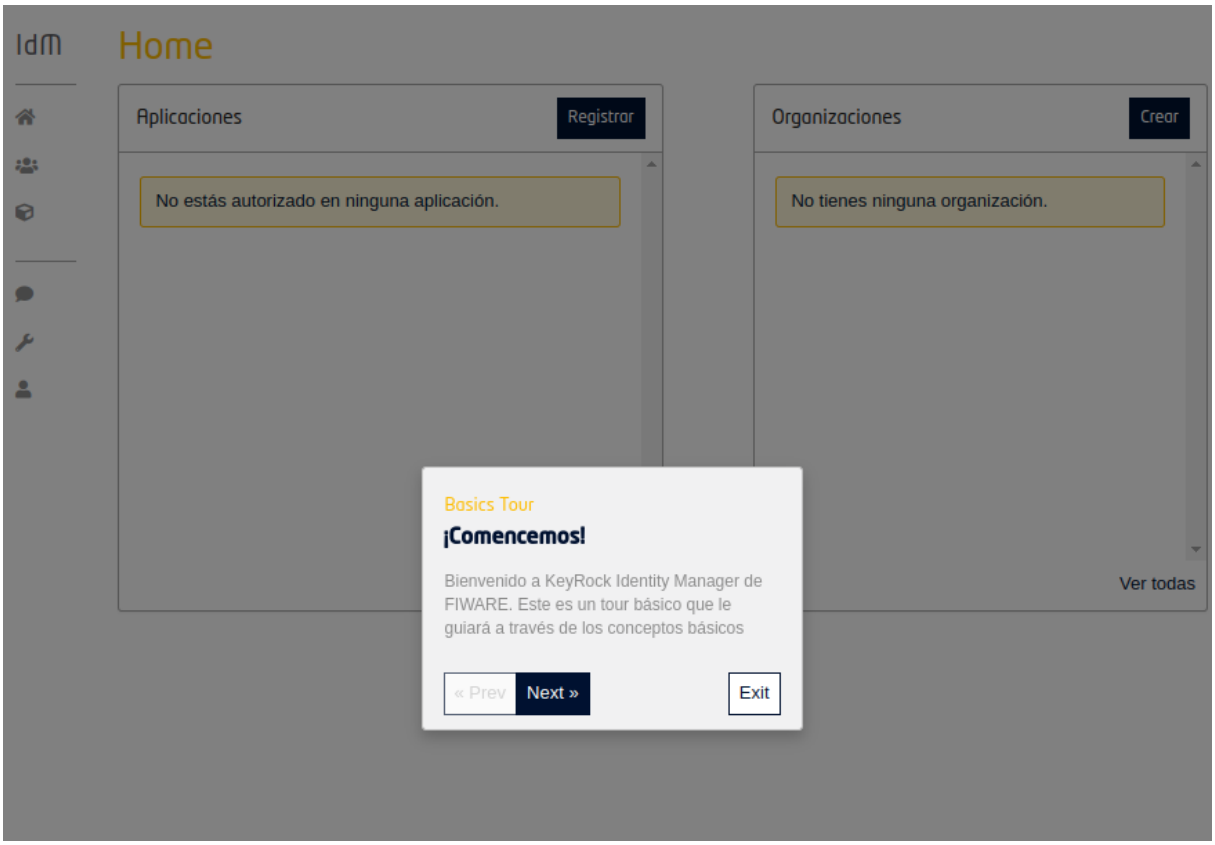


Figura 27: Panel de administración de Keyrock

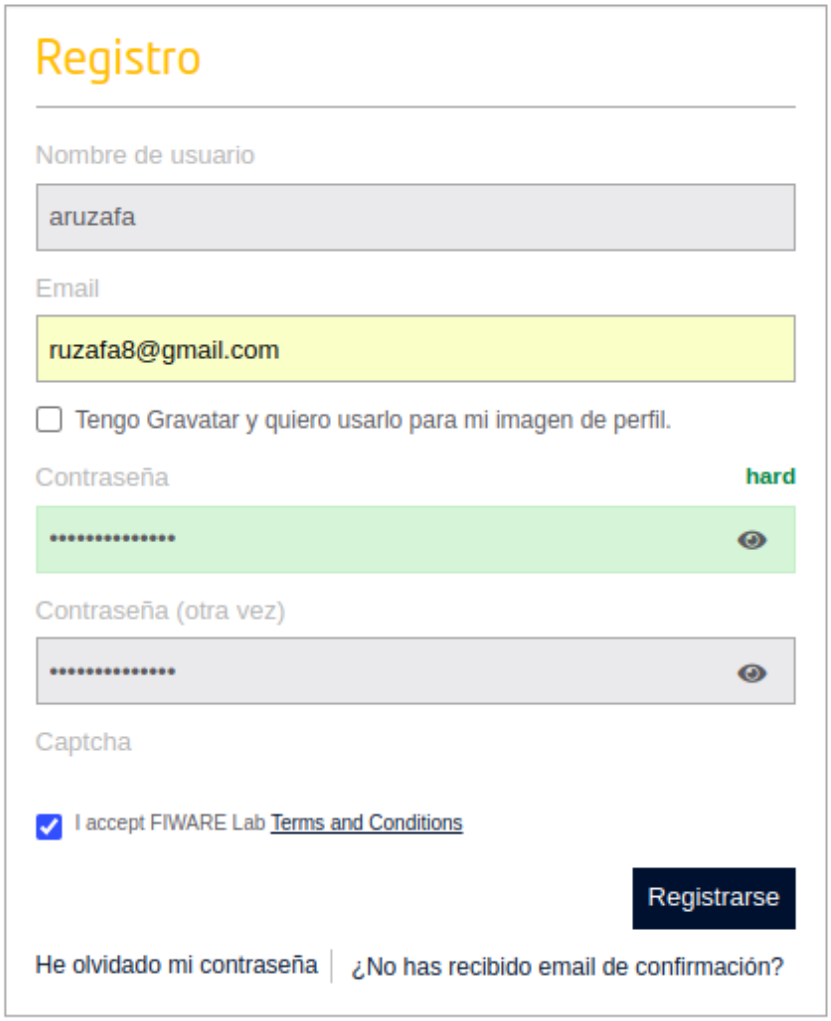


Figura 28: Formulario de registro con Keyrock

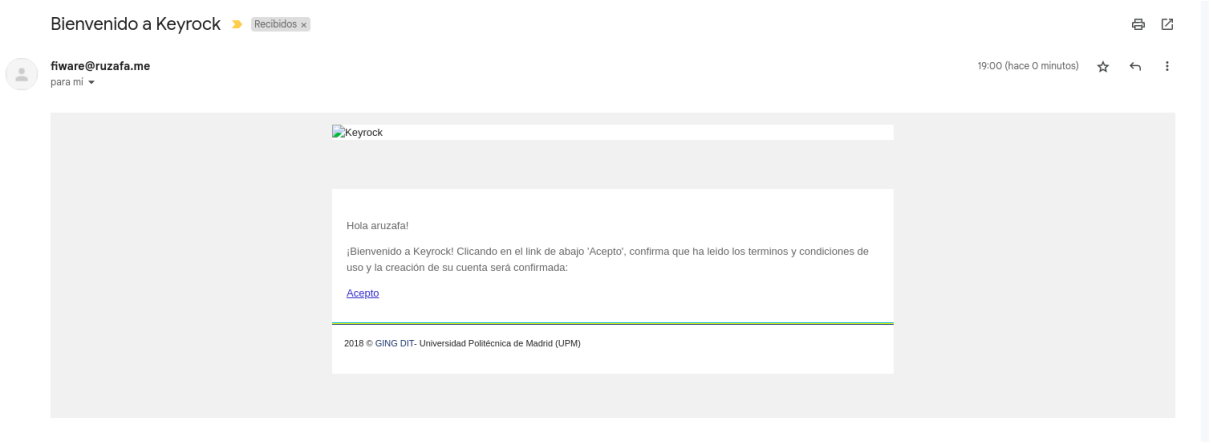


Figura 29: correo electrónico de confirmación recibido tras rellenar el formulario de registro

5.2. Desarrollos del frontal

Una vez desplegados los servicios necesarios, empezamos a desarrollar el código para un frontal donde poder visualizar los datos de FIWARE. Para ello, a partir de la segunda iteración, se utilizó NextJS como tecnología para implementarlo. A continuación, se detallarán los detalles más relevantes y las configuraciones que han sido necesarias para desarrollar este frontal.

5.2.1. Estructura del proyecto

La estructura del proyecto ha venido condicionada por el uso de NextJS, ya que al ser un framework, es él el que da las pautas de estructuración y el que controla el flujo de la aplicación. Por ejemplo, todo lo que se coloque dentro de las carpetas app o src se enrutarán automáticamente y será accesible desde una URL.

- **api**: Lógica de las conexiones a las API de FIWARE u otras externas. Las conexiones se realizarán utilizando la función fetch, disponible en javascript vanilla.

- **app**: Carpeta donde se sitúan las pantallas de la aplicación. El nombre de esta carpeta es importante, ya que NextJS lo usará para enrutar estos componentes.

- **components**: Aquí se almacenan componentes fabricados con React, como los formularios o tablas y que se reutilizarán en la aplicación.

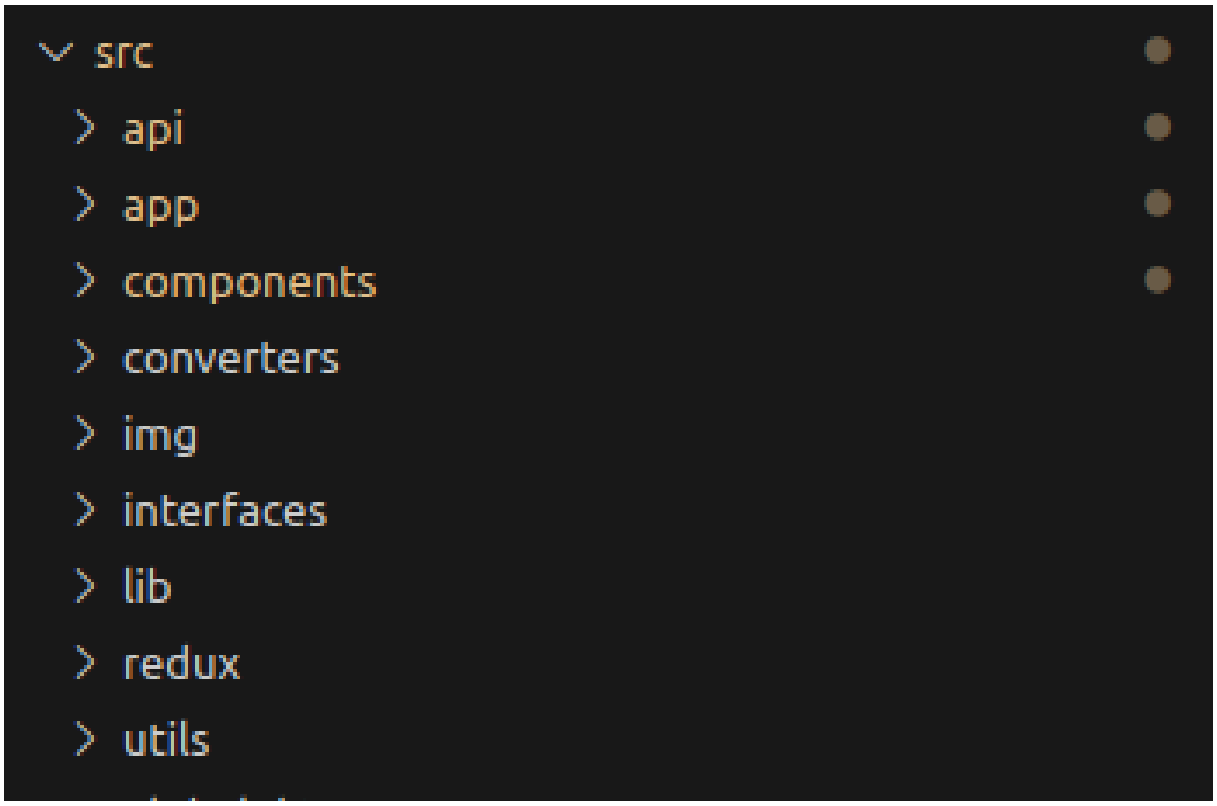


Figura 30: Estructura de la carpeta src del proyecto de `IBuilding`

- **converters**: Funciones para transformar los datos de las interfaces NGSI a los objetos estándar con los que se trabaja en la aplicación (DTO).
- **img**: Imágenes estáticas.
- **interfaces**: Contiene las definiciones de interfaces en typescript con los tipos de datos que se usan en la aplicación (usuario, edificio, dispositivo, ...)
- **lib**: Archivos con hooks de React u otros componentes especiales, como Providers.
- **redux**: Archivos de código para la configuración del Redux [17] de la aplicación.
- **utils**: Archivo con funciones reutilizables en varias partes del código.

5.2.2. Flujo de trabajo

El flujo de trabajo utilizado para organizar las ramas en git está basado en "gitflow". Cada tarea de Jira dispone de una rama en el repositorio donde se desarrolla la misma. Una vez

[17] Redux es un patrón de arquitectura de datos que permite manejar estados en una aplicación.

finalizado, se mezcla en la rama “beta”, donde se van añadiendo todos los desarrollos durante el sprint. Previamente a terminar el sprint, se revisan posibles errores que se pueden resolver directamente añadiendo *commits* en esta rama “beta”, para proceder a mezclar esta rama en *main*. Este flujo, que se puede consultar visualmente en la figura 31, ha permitido trabajar en diversas tareas concurrentemente, ya que está orientado a llevar un control de cada tarea, teniendo una única rama y una tarjeta en Jira con comentarios y otra información relevante, y tener un registro por iteración de qué está pasando.

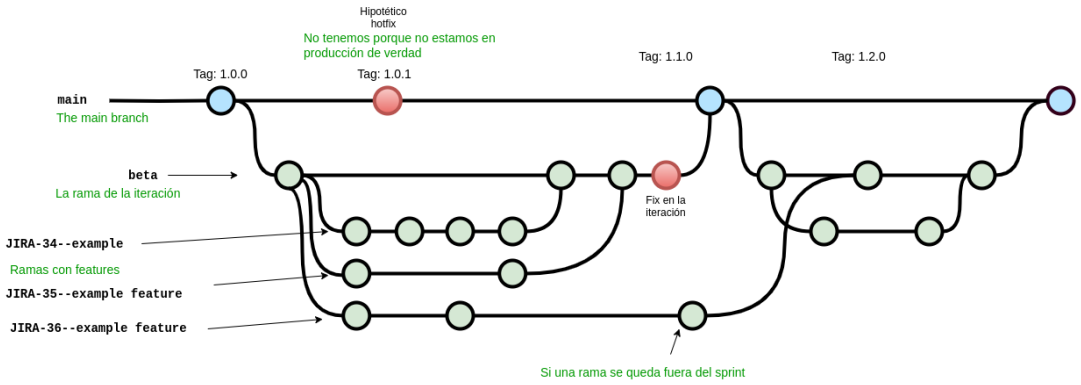


Figura 31: Flujo de git seguido

Como buena prácticas se ha utilizado ESLint, para un formateo y legibilidad de código buena. De esta forma, todas las tareas que se mezclaran el “beta” debían de pasar la validación de ESLint. También se usó *semantic release*, que permitió tener un control versiones del código automáticamente mediante *tags*, comprobando el contenido de los *commits*.

5.2.3. Despliegue en Vercel

Vercel ofrece servidores gratuitos con un dominio para los usuarios que deseen desplegar código desarrollado con NextJS, además de un sistema de entrega continua (CD, del inglés *Continous Delivery*) para desplegar los cambios en la web automáticamente. Desde la web se puede configurar cuál es el repositorio fuente de git, como se ve en la figura 32. De esta forma, cualquier cambio que se haga en la rama *main* se podía ver en la web automáticamente.

Además del dominio, Vercel también permite añadir configuraciones para añadir dominios propios. En este caso, al disponer de un dominio propio, se configuraron dos subdominios para disponer de dos entornos de desarrollo. Como la configuración de entornos de desarrollo en Vercel se hace por ramas, y disponemos de dos ramas principales debido al flujo de desarrollo,

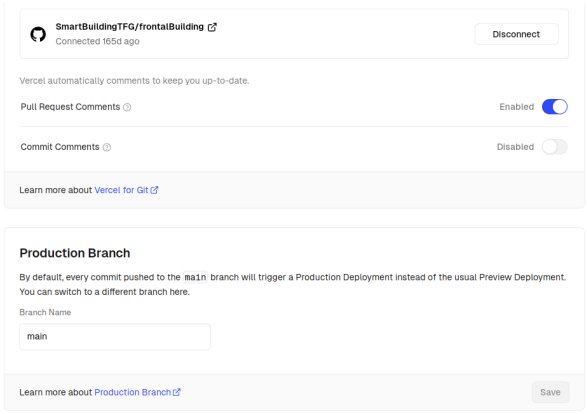


Figura 32: Configuración del repositorio de git en Vercel para los despliegues.

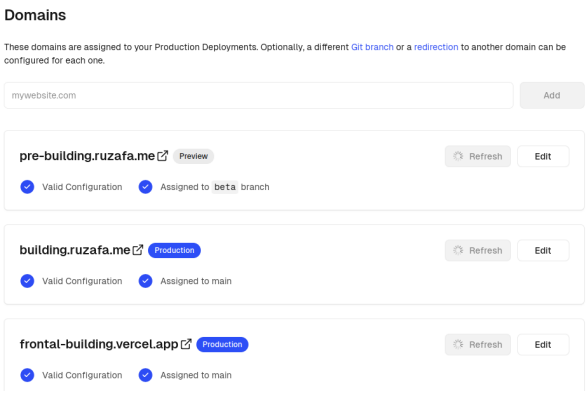


Figura 33: Configuración de los entornos de desarrollo por ramas en Vercel.

main y beta, resulta natural tener dos entornos de desarrollo: Uno asociado a la rama main, que podríamos denominar producción, y otro asociado a "beta", pre-producción. En la figura 33 se puede observar la configuración de los entornos y cómo se relacionan con las ramas.

Tener un entorno de pre-producción ha resultado útil por varios motivos:

- Para realizar pruebas, ya que era posible consultar el estado de ambos entornos, producción y pre-producción y comparar los cambios que se estaban aplicando en la rama "beta" para determinar si un cierto error provenía de una tarea subida en el sprint.

- Configuraciones más realistas: Se ha podido hacer un uso más realista de las variables de entornos, la configuración del Nginx también era más avanzada, como se ha mencionado anteriormente, para soportar múltiples orígenes, etc.

- La configuración desde Vercel era completamente trivial: Solo hubo que configurar los

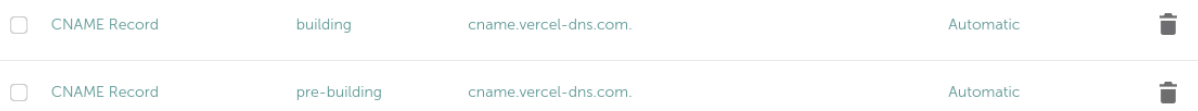


Figura 34: Registros de los subdominios en Namecheap para apuntar a la DNS de Vercel.

subdominios desde Namecheap, para que apuntaran a la DNS de Vercel, como se puede observar en la figura 34.

5.2.4. Integración del flujo de OAuth2.0

Para la implementación de OAuth2.0, se implementó el flujo de concesión de código de autorización. Para ello se registró la aplicación en Keyrock, generando un ID de cliente y un secreto que se almacenaron en variables de entorno. Los pasos para el registro de la aplicación están documentados en las figuras 35, donde se observa el Home de Keyrock. Desde aquí se pueden registrar las aplicaciones pulsando el botón de "Registrar", la figura 36, con un formulario a rellenar con la información de la aplicación, como la URL o la URL de retorno y la figura 37, donde se puede configurar el tipo de token, que en este caso lo queremos tipo JWT, o consultar el ID y secreto del cliente, que será necesario proporcionar al frontal. Con esto, se puede construir la URL que redirige al servidor de OAuth para que el usuario pueda autenticarse:

```
const authorizeOauth = (): void => {
  const redirectUri = encodeURIComponent(REDIRECT_URI)
  window.location.href = `${API_URL}/oauth2/authorize?
    response_type=code&client_id=${CLIENT_ID}
    &redirect_uri=${redirectUri}&scope=jwt`
}
```

La redirect URI, que corresponde con "URL de retorno" en Keyrock, es la URL a la que volverá el cliente si la autenticación es correcta. Es una URL del frontal, que estará esperando, como parámetro, un código generado en ese momento que deberá enviar de nuevo al servidor, junto con su secreto, para obtener el token del usuario. Véase la función que se envía al servidor para obtener el token:

```
const oauthLogin = async (code: string): Promise<LoginResponse> => {
  return await fetch(`${API_URL}/oauth2/token`, {
```

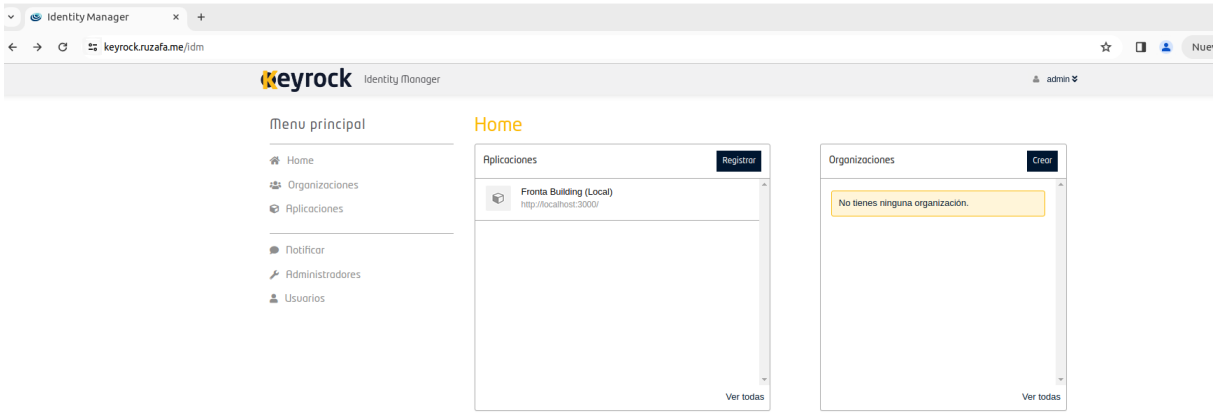


Figura 35: Home de Keyrock

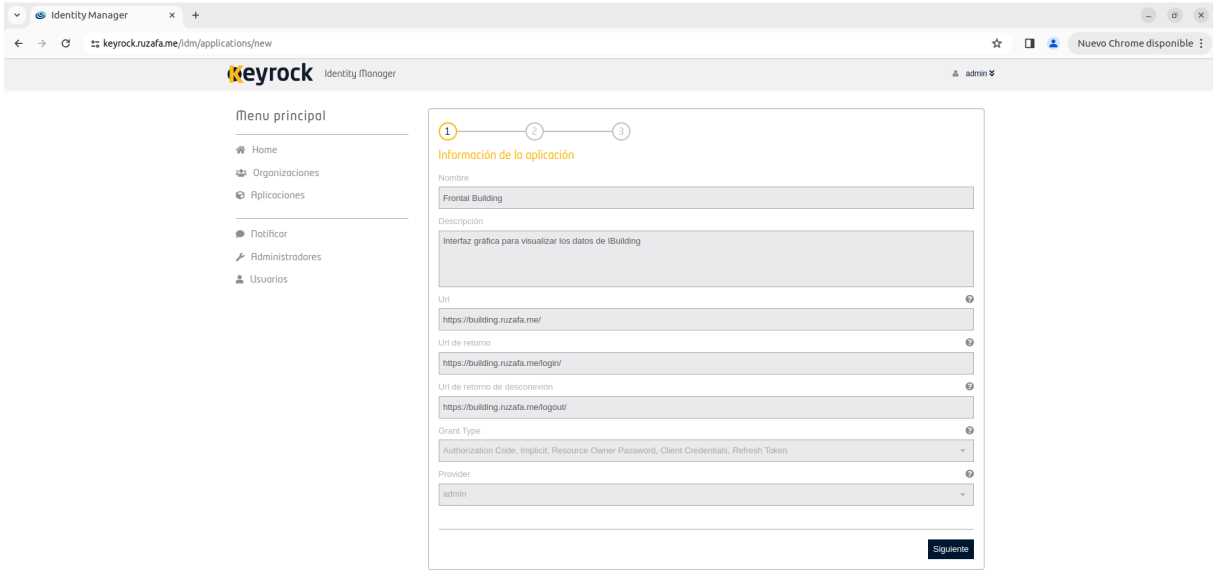


Figura 36: Formulario de registro de aplicación de Keyrock

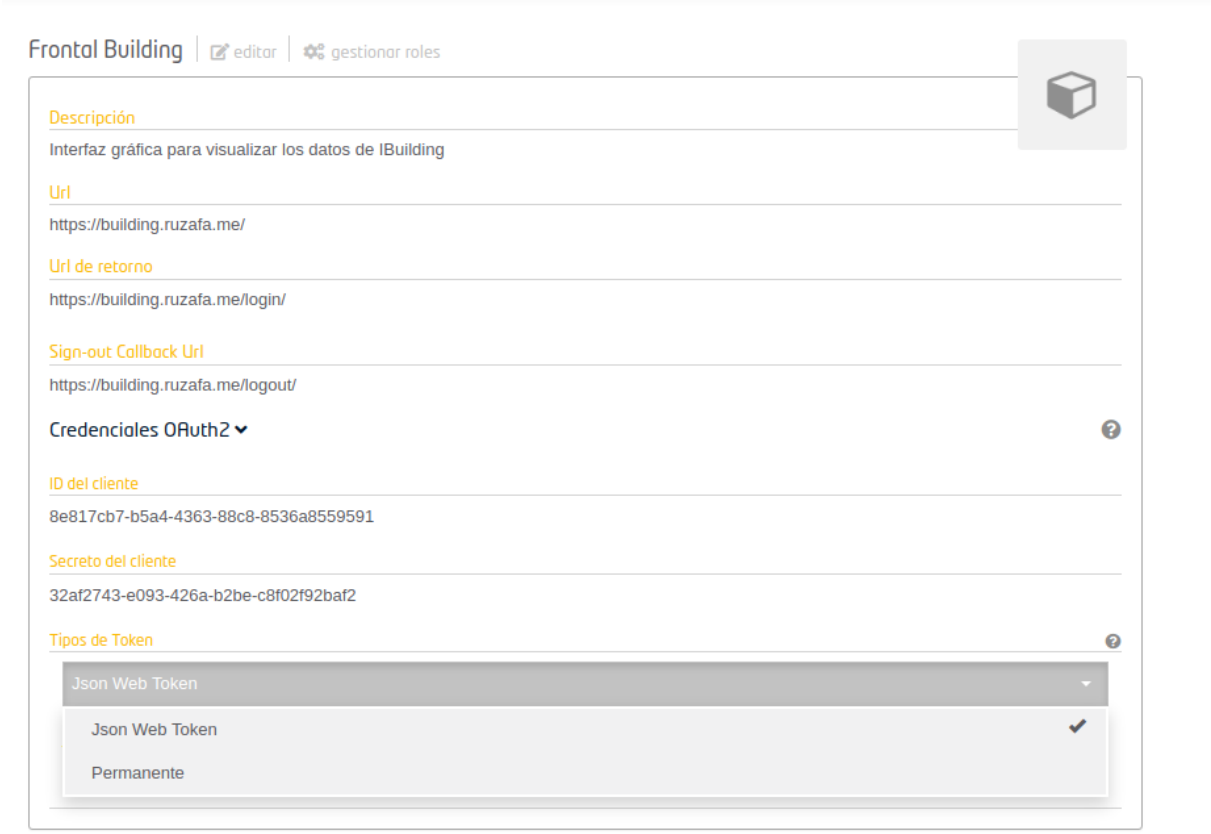


Figura 37: Detalles de la configuración de la aplicación

```
    method: 'POST',
    headers: {
      Authorization: `Basic ${Buffer.from(
        `${CLIENT_ID}:${CLIENT_SECRET}`
      ).toString('base64')}`,
    },
    body: new URLSearchParams({
        code,
        grant_type: 'authorization_code',
        redirect_uri: REDIRECT_URI, scope: 'jwt'
    }),
  })
```

Este token traerá información del usuario en formato JWT, así que, una vez obtenido, se almacena en el local storage para mantener la sesión iniciada y se decodifica para extraer el ID del usuario. Con este ID, consultamos el sistema de Orion, para ver si ese usuario existe y obtener toda la información relevante para la aplicación y, sino, crearlo. Una vez obtenida la información del usuario, esta es asimilada por el Redux para utilizarla por toda la aplicación. A continuación mostramos la sección del código implementa este algoritmo. Esta sección está en la URL de callback que se llama una vez el usuario se ha autenticado en el servidor de Keyrock.

```
const code = searchParams.get('code')
IdentityManager.oauthLogin(code).then((res: LoginResponse) => {
  localStorage.setItem('token', res.access_token)
  localStorage.setItem('refresh_token', res.refresh_token)
  const idmUser: IDMUser = jwtDecode<IDMUser>(res.access_token)
  getUserFromIDMIdentifier(idmUser.id)
    .then((user: User | null) => {
      if (user === null) {
        createUserFromIDM(idmUser)
          .then((newUser: User) => dispatch(setUser(newUser)))
          .catch(console.error)
      } else {
        dispatch(setUser(user))
      }
    }).catch(console.error)
  push('/home')
```

```
}
```

5.2.5. Implementación de los modelos SAREF

Las especificaciones de SAREF nos detallan información sobre cuáles son las entidades y cuáles son las relaciones principales, pero no nos especifican qué atributos tienen estas entidades. Para ello se pueden hacer extensiones del modelo adaptadas a la lógica del negocio, por ejemplo, si necesitamos un atributo que sea la dirección, podemos añadirlo y tiparlo como resulte más conveniente, teniendo en cuenta de que existen ya tipos definidos por organismos como schema.org.

FIWARE participa en un programa llamado Smart Data Models [18] que también estandariza tipos y relaciones. Tiene sus propias ontologías. Una de ella extiende del mismo modelo que utilizamos, SAREF4BLDG, añadiendo atributos que consideran apropiados. En su repositorio de github [19] podemos encontrar estas especificaciones.

Para trabajar con las entidades que FIWARE nos devuelve o que queremos registrar, hemos decidido utilizar DTO [20]. Gracias a este patrón podemos unificar los objetos de FIWARE, ya que las peticiones de creación requerirán un formato concreto y las peticiones que nos devuelvan la información, lo harán en otro formato.

Concretamente, a la hora de crear un edificio, según nos exige NGSI-LD debemos indicar el tipo de la entidad, por ejemplo, en caso de un Building Space, sería un *s4bldg:BuildingSpace*, o los atributos de las relaciones, por ejemplo la relación entre Building y BuildingSpace deben llevar su nombre completo también (*s4bldg:isSpaceOf*), además de que los atributos que tengan algún tipo especial, como Relación o PostalAddress, deben indicarse también. De la misma forma, cuando se recupera un edificio, recibimos los atributos con los nombres completos, lo cual es impráctico a la hora de acceder a ellos mediante código. Véase un ejemplo en la figura 38

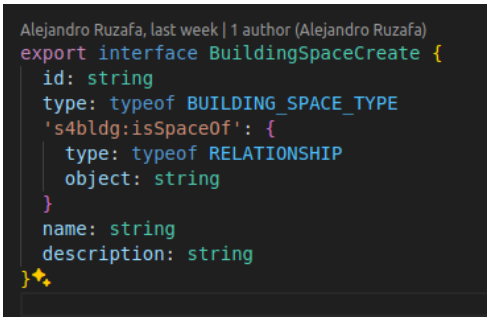
Para su unificación en los DTO, se utilizan unas funciones que transformarán los objetos entre estos tipos. Véase 39. De esta forma, las funciones que estén en la capa de conexión

[18] https://smartdatamodels.org/

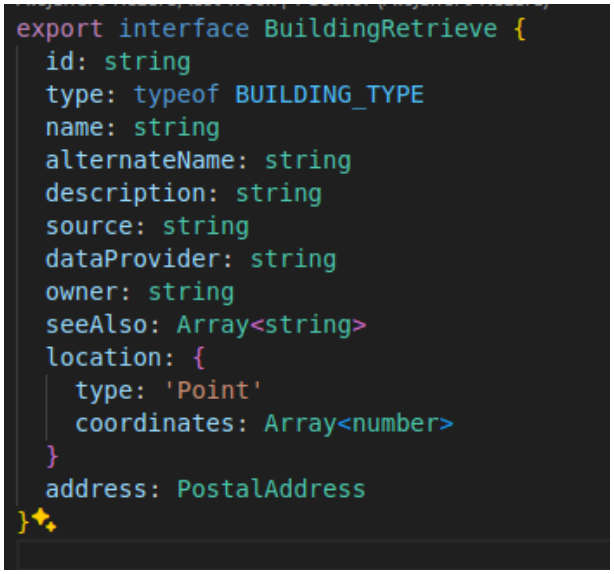
[19] https://github.com/smart-data-models/dataModel.S4BLDG/

[20] Objeto de trasferencia de datos (DTO, del inglés *Data Transfer Object*) es un patrón que nos permite encapsular los datos en un formato apropiado para transferirlo entre procesos de las aplicaciones.

con la API solo recibirán o devolverán objetos del DTO, que serán transformados usando las funciones conversoras, como en la figura 40, donde observamos cómo hacemos la llamada para obtener las medidas de un sensor, ordenado por fecha de creación y poniendo uno como límite, ya que solo nos interesa la última.



(a) Tipo para almacenar un Building Space



(b) Tipo para recuperar un Building

Figura 38: Interfaces desarrolladas en el código siguiendo las ontologías SAREF4BLDG y Smart Data Models

5.2.6. Crear/visualizar el listado de edificios edificio

Cuando entramos en la web accedemos por defecto al Home (figura 42), donde podemos visualizar el listado de edificios con "scroll" infinito para que los resultados vengan paginados. También podemos crear edificios, pero solo lo podrán hacer aquellos usuarios que hayan iniciado sesión. Véase 41.

5.2.7. Detalles del edificio

Si seleccionamos un edificio del listado, accedemos al detalle del mismo. En la figura 43, accede el usuario que creó el edificio, por lo que puede actualizar los campos de los datos generales, pero en caso de que no haya iniciado sesión o sea otro usuario, no podré editar la información del mismo, como se puede apreciar en la figura 44.

```
export function buildingRetrieveToBuildingDTO(buildingRetrieve: BuildingRetrieve): BuildingDTO {
  const { id, name, alternateName, description, source, dataProvider, owner, seeAlso, location, address } = buildingRetrieve
  return {
    id,
    name,
    alternateName,
    description,
    source,
    dataProvider,
    owner,
    seeAlso,
    location: { coordinates: location.coordinates },
    address,
  }
}

export function buildingDTOToBuildingCreate(buildingDTO: BuildingDTO): BuildingCreate {
  const { id, name, alternateName, description, source, dataProvider, owner, seeAlso, location, address } = buildingDTO
  return {
    id,
    type: BUILDING_TYPE,
    name,
    alternateName,
    description,
    source,
    dataProvider,
    owner,
    seeAlso,
    location: { type: 'Point', coordinates: location.coordinates },
    address: { value: address },
  }
}
```

Figura 39: Funciones conversoras entre los tipos de la API y los DTO

```
import { type MeasurementRetrieve } from '@/interfaces/measurement/measurement-retrieve.interface'
import { Option, type Filters } from './orion-ld'
import { type MeasurementDTO } from '@/interfaces/measurement/measurement-dto.interface'
import { OrionLD } from '.'
import { MEASUREMENT_TYPE } from '@/interfaces/other/types.interface'
import { measureRetrieveToMeasurementDTO } from '@/converters/measure.converter'

async function retrieveMeasurement<T>(filters: Filters, options = OrionLD.DEFAULT_OPTION): Promise<Array<MeasurementDTO<T>>> {
  return await OrionLD.getAllObjects<MeasurementRetrieve>({
    ...filters,
    type: MEASUREMENT_TYPE,
  }, options).then(measurements => measurements.map(measureRetrieveToMeasurementDTO))
}

export async function getLastMeasurementsBySensor(sensorId: string): Promise<Nullable<MeasurementDTO>> {
  return await retrieveMeasurement<number>({
    q: `s4bldg:measurementMadeBy=="${sensorId}"`,
    orderBy: '!dateCreated',
    limit: '1',
  }, [Option.concise, Option.sysAttrs]).then(measurements => measurements.length > 0 ? measurements[0] : null)
}
```

Figura 40: Llamadas a la API de NGSI-LD para obtener la medida de un sensor

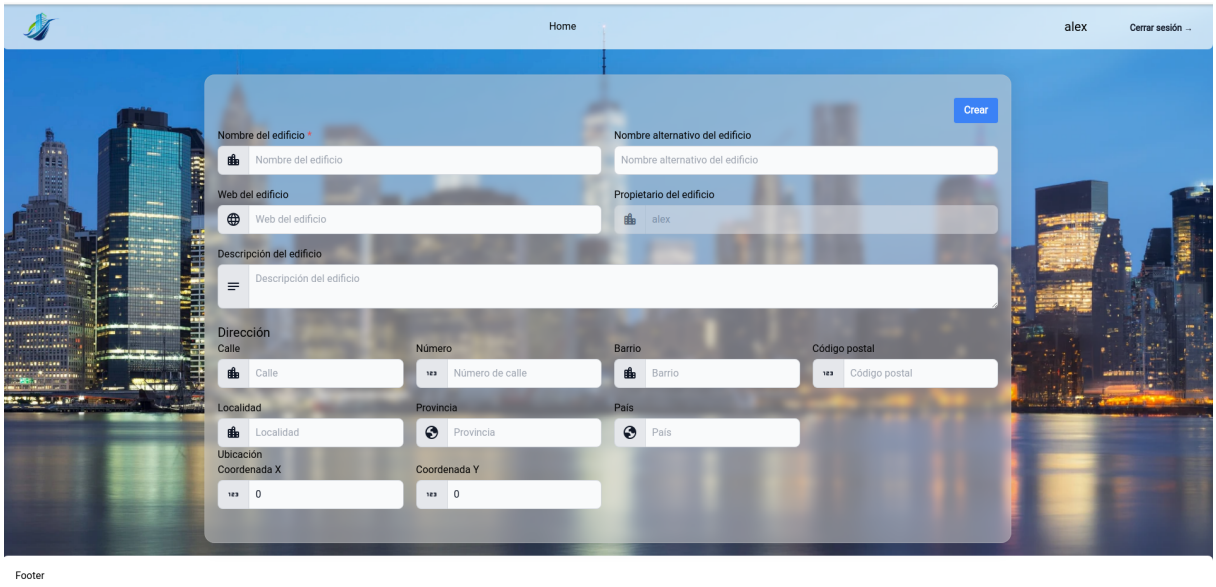


Figura 41: Formulario de creación de edificios

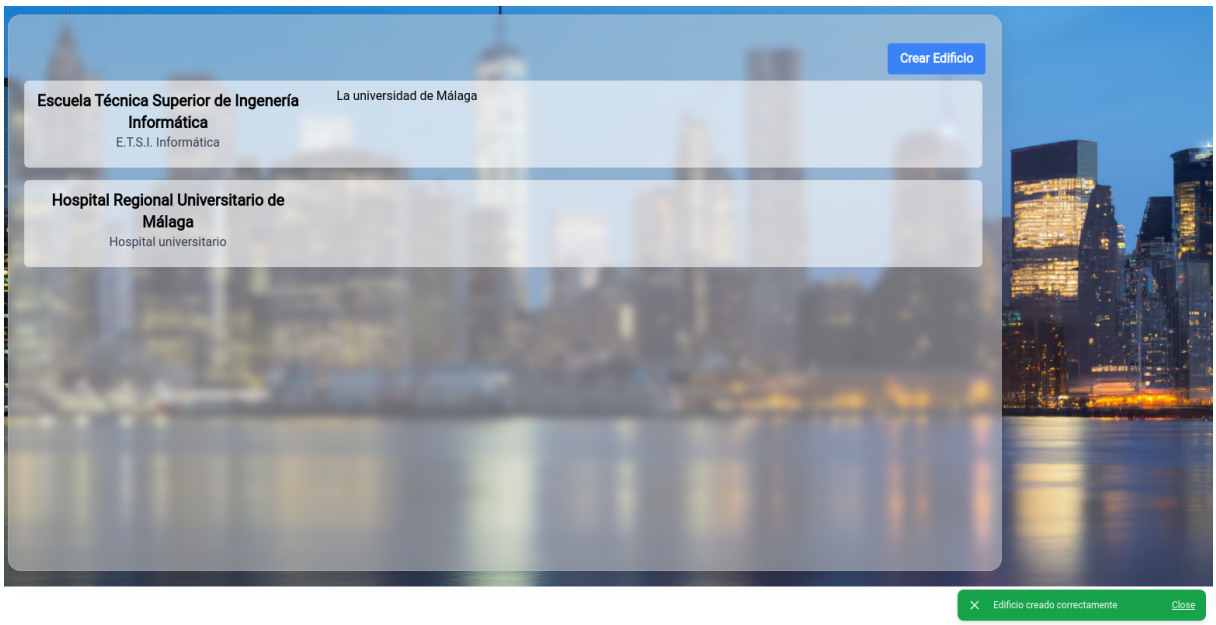


Figura 42: Listado de edificios

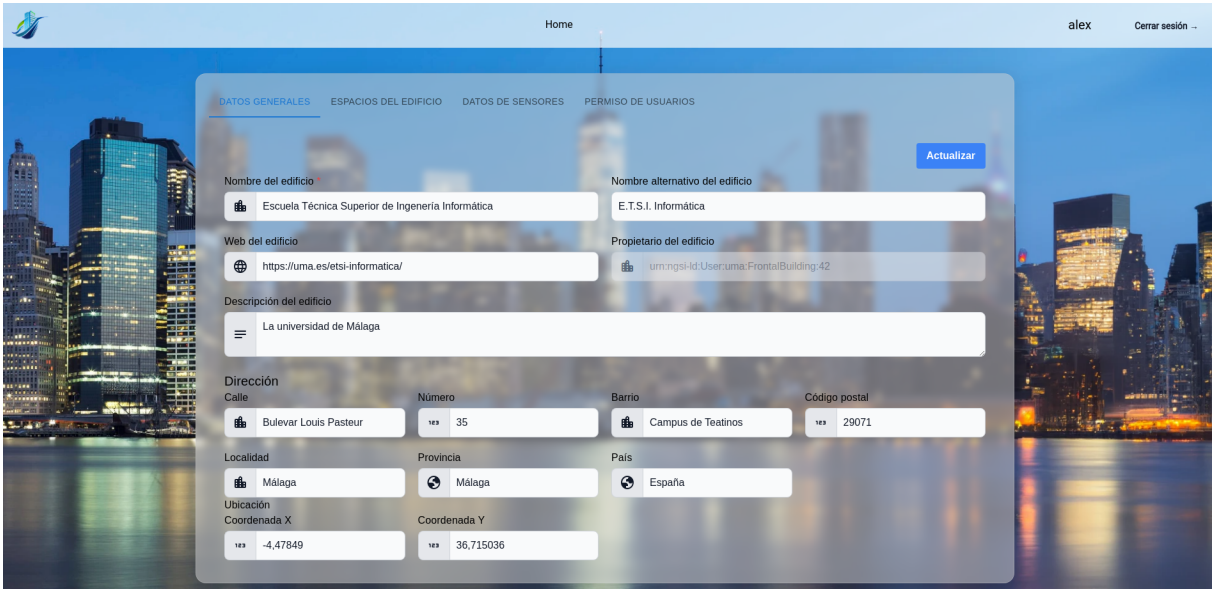


Figura 43: Detalles de un edificio

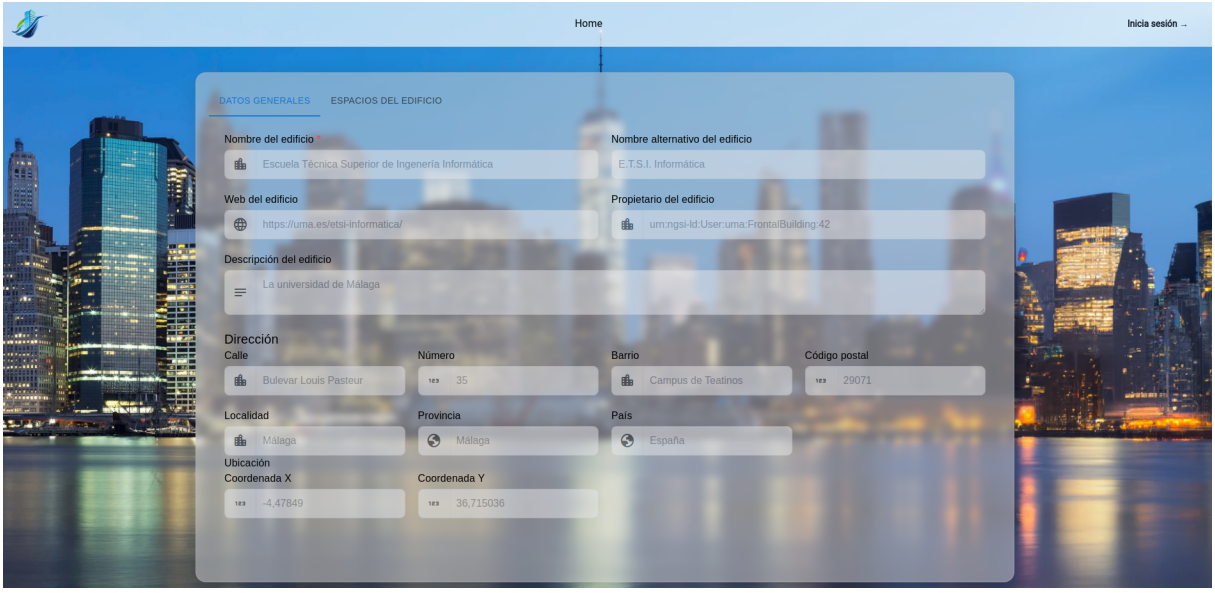


Figura 44: Detalles de un edificio sin iniciar sesión

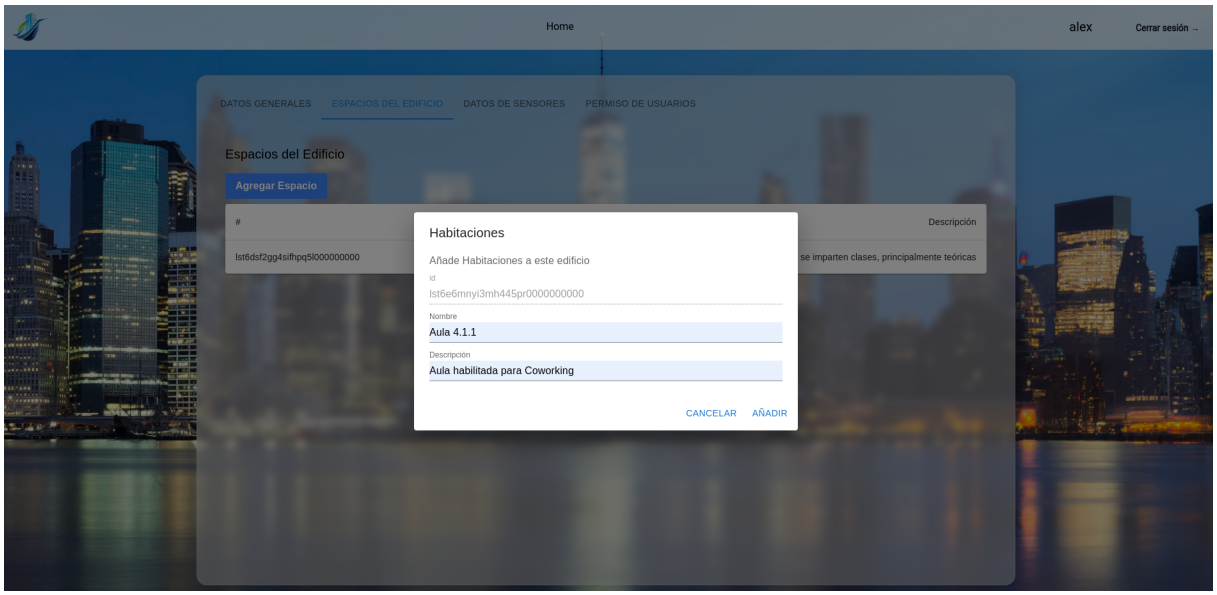


Figura 45: Formulario de creación de espacios de edificios

5.2.8. Creación de sensores

Para crear sensores, según la ontología SAREF, es necesario asociarlos a un espacio del edificio, por lo que previamente hay que acceder a la sección de “Espacios del edificio” en los detalles del edificio, como se observa en la figura 45, para poder crear un espacio y posteriormente asignárselo al sensor, como se puede observar en la figura 46. Así pues, en figura 47 se puede observar el listado de sensores del edificio y el valor de su última medición.

5.3. Desarrollos en Arduino

Para la realización de este apartado, originalmente, se decidió utilizar IoTAgent-UL, que permite conectar dispositivos que utilicen la interfaz UltraLight, ya que es un protocolo sencillo y ligero. Sin embargo, una vez desplegado y realizado pruebas, no fue posible compatibilizarlo con el modelo SAREF.

El modelo SAREF indica que las medidas de las entidades de tipo “Device”, deben almacenarse como entidades de tipo “Measurement” y asociarlas al “Device” mediante la relación “isMeasurementOf”, sin embargo, el IoT Agent, una vez que recibe la medida del sensor, la escribe en un atributo de “Device”, por lo que, en vez de crear nuevas instancias, estamos sobrescribiendo continuamente un atributo.

Existen diversas alternativas para solucionar esto. Por un lado se podría abandonar total

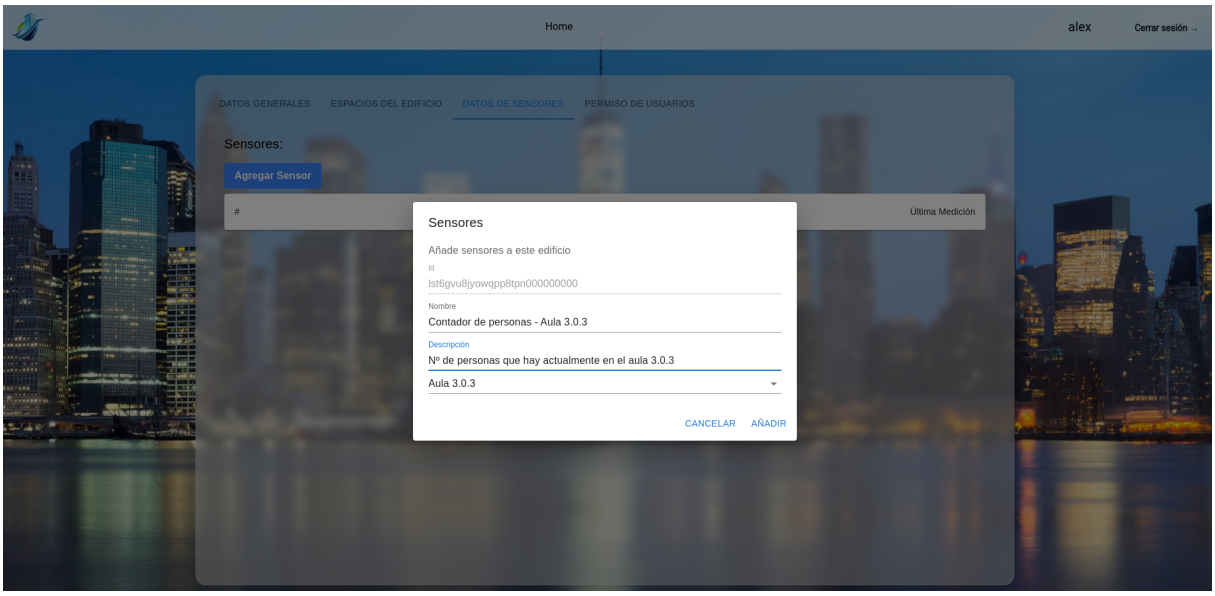


Figura 46: Formulario de creación de sensores

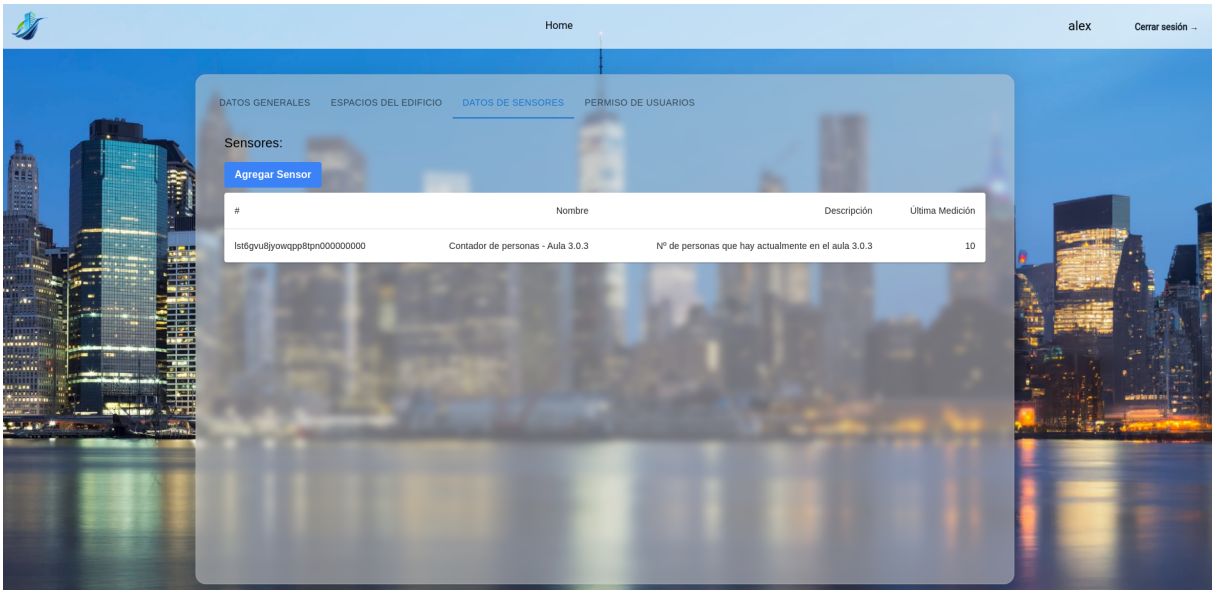


Figura 47: Listado de los sensores

o parcialmente el uso de la ontología SAREF, o, por otro lado, también se podría crear un IoT Agent personalizado, ya que FIWARE nos proporciona las herramientas para personalizarlo a un protocolo específico o un comportamiento deseado.

Sin embargo, en este caso hemos optado por abandonar el uso de IoT Agent y programar en el sensor que use el API NGSI-LD directamente, ya que abandonar la ontología SAREF iría en contra de las ideas de estandarización planteada en el TFG y construir un IoT Agent personalizado es una tarea compleja y se sale del ámbito del proyecto.

5.3.1. IoT Agent

Antes de detectar el problema de compatibilidad con SAREF se desplegó este microservicio añadiendo la siguiente configuración al docker-compose.yml de orion.

```
iot-agent:
  image: fiware/iotagent-ul
  hostname: iot-agent
  container_name: fiware-iot-agent
  depends_on:
    - mongo-db
  expose:
    - "4061"
    - "7896"
  ports:
    - "4061:4061"
    - "7896:7896"
  environment:
    - "IOTA_CB_HOST=orion-ld"
    - "IOTA_CB_PORT=1026"
    - "IOTA_NORTH_PORT=4061"
    - "IOTA_REGISTRY_TYPE=mongodb"
    - "IOTA_MONGO_HOST=mongo-db"
    - "IOTA_MONGO_PORT=27017"
    - "IOTA_MONGO_DB=iotagent-ul"
    - "IOTA_HTTP_PORT=7896"
    - "IOTA_PROVIDER_URL=http://iot-agent:4061"
```

Para que lo servicios estén organizados, es necesario configurar un grupo de servicios. Los

servicios en los IoT Agents nos permitirán guardar la información en bases de datos de MongoDB separadas. Cada servicio agrupa dispositivos de un tipo específico en la aplicación. En nuestro caso agrupamos los dispositivos IoT de los edificios en el servicio de "iotbuildings". Se podrían crear otro tipos de servicios como parkings, parques, transporte, ... Por simplificación trabajaremos con uno solo.

La siguiente petición, lanzada al North Port de IoT Agent, crea un grupo de servicios llamado "iotbuildings" que almacena objetos del tipo Sensor. Los sensores que quieran enviar datos, deberán comunicarse con el adaptador accediendo al recurso indicado en el cuerpo de la petición ("/iot/d" en este caso), deberán indicar el nombre del mismo en la cabecera "fiware-service", el api key e indicar cuál es el service-path en el que actúan.

```
curl -iX POST \
  'http://localhost:4061/iot/services' \
  -H 'Content-Type: application/json' \
  -H 'fiware-service: iotbuildings' \
  -H 'fiware-servicepath: /' \
  -d '{
 "services": [
   {
     "apikey": "1234",
     "cbroker": "https://orion.ruzafa.me",
     "entity_type": "Sensor",
     "resource": "/iot/d"
   }
 ]
}'
```

Una vez registrado el servicio, podemos lanzar una petición al North Port de IoT Agent para registrar un dispositivo dentro del servicio. Como se puede observar en la siguiente petición, podemos indicar dos tipos de atributos: los "attributes" y los "static_attributes", o atributos estáticos.

Los atributos estáticos son aquellos que no van a cambiar normalmente de valor a diferencia de los "attributes" que el IoT Agent actualizará cada vez que el sensor emita un dato.

```
curl -iX POST \
  'http://localhost:4061/iot/devices' \
```

```
  -H 'Content-Type: application/json' \
  -H 'fiware-service: iotbuildings' \
  -H 'fiware-servicepath: /' \
  -d '{
    "devices": [
        {
            "device_id":    "motion001",
            "entity_name": "urn:ngsi-ld:Motion:001",
            "entity_type": "Motion",
            "timezone":    "Europe/Berlin",
            "attributes": [{ "object_id": "c", "name": "count", "type": "
               Integer" }],
            "static_attributes": [{ "name":"refBuilding", "type": "
               Relationship", "value": "urn:ngsi-ld:Building:001" }]
        }
    ]
}'
```

Nótese que en ninguna de las peticiones se ha hecho uso de JSON-LD, ni se ha proporcionado ningún contexto, ya que originalmente IoT Agent se desarrollo para NGSI-V2. La versión Linked Data es una adaptación para que puedan funcionar los casos de uso, pero no cumple la funcionalidad Linked Data completamente.

De hecho, si en vez de enviar un contexto con los tipos simplificados, intentamos enviar el nombre del tipo completo, con su URI, obtenemos error debido a que las URI tienen caracteres inválidos como la barra "/".

5.3.2. Dispositivos físicos

La placa que usaremos es una NodeMCU [21] de la marca Amica. Se trata de una placa de desarrollo que dispone de un chip WiFi de la familia ESP. Esta placa dispone de un conjunto de pines con diferentes funcionalidades: Pines que proporcionan voltaje, pines que van a tierra y pines que pueden servir de entrada o salida de datos. Estos pines se pueden conectar a pines de diferentes dispositivos físicos para activar una medición y recuperar el resultado o para que

[21] NodeMCU es una tarjeta de desarrollo similar a Arduino, especialmente orientada al Internet de las cosas (IoT).

realicen una función.

En nuestro caso, para el desarrollo del dispositivo físico, se ha decidido programar un contador de personas utilizando un sensor ultrasónico HC-SR04, que es capaz de emitir una onda de ultrasonido la cual, posteriormente, vuelve a detectar tras rebotar contra un obstáculo.

Como conocemos el momento en el que sensor lanzó la señal y en el que lo detectó de vuelta, sabemos el tiempo que ha tardado en volver a detectar esta señal ultrasónica, y si, además, conocemos la velocidad a la que ha viajado la onda, que es la velocidad del sonido, podemos determinar la distancia en la que se encontraba el obstáculo contra el que rebotó la onda, por lo tanto, con unos ajustes en el código, podemos tener un sensor de proximidad.

El dispositivo dispone de 4 pines: VCC para recibir los voltios, GND para la toma a tierra, TRIG, por donde recibe una señal para indicarle que realice una medición lanzando un ultrasonido, y ECHO, por donde comunica que ha detectado el "eco" del ultrasonido.

Los pines del sensor deben conectarse apropiadamente a la placa de desarrollo y desde el código se debe indicar la configuración de los pines para que funcione correctamente. En la imagen 48, se puede ver los detalles de la conexión entre el sensor de proximidad y la placa de desarrollo y en la figura 49 el montaje del dispositivo físico sobre una protoboard. Los pines D2 y D3 se conectan con ECHO y TRIG respectivamente, por lo que en el código debemos configurar estos pines correctamente (ECHO como pin de entrada y TRIG como pin de salida).

5.3.3. Implementación del código en Arduino

Utilizando la función "digitalWrite" podemos enviar una señal por TRIG para activar el sensor. A continuación, usando la función "pulseIn" podemos esperar a que ECHO envíe la señal de detección. Esta función nos devolverá el tiempo, en microsegundos, que ha tardado en activarse el pin ECHO o, en otras palabras, el tiempo que ha tardado el sensor en detectar la señal que emitió.

Sabiendo que $v = \frac{s}{t}$, donde v es velocidad, s es espacio y t es el tiempo, podemos despejar el espacio: $v * t = s$. La velocidad del sonido a 20 ºC es de 343 m/s, sin embargo, teniendo en cuenta que el tiempo lo tenemos en milisegundos y que las distancias que trabajaremos estarán en el orden de los centímetros, hay que aplicar un factor de conversión para adaptar

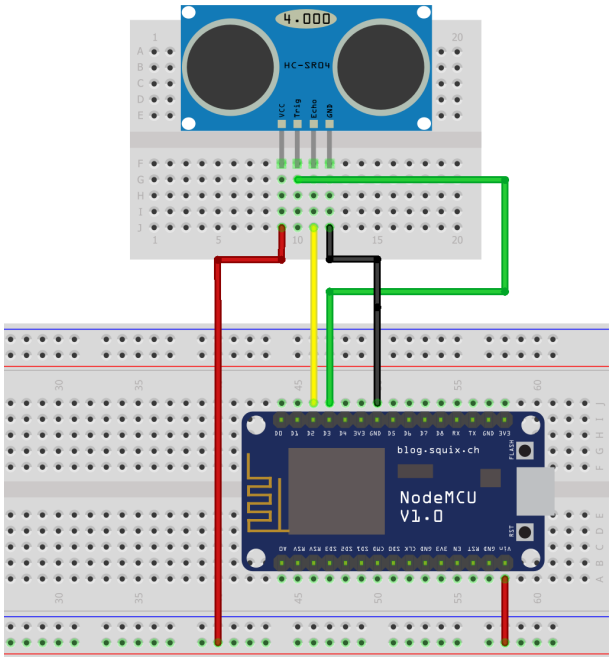


Figura 48: Esquema de conexión entre el sensor de proximidad y la placa de NodeMCU elaborado con Fritzing

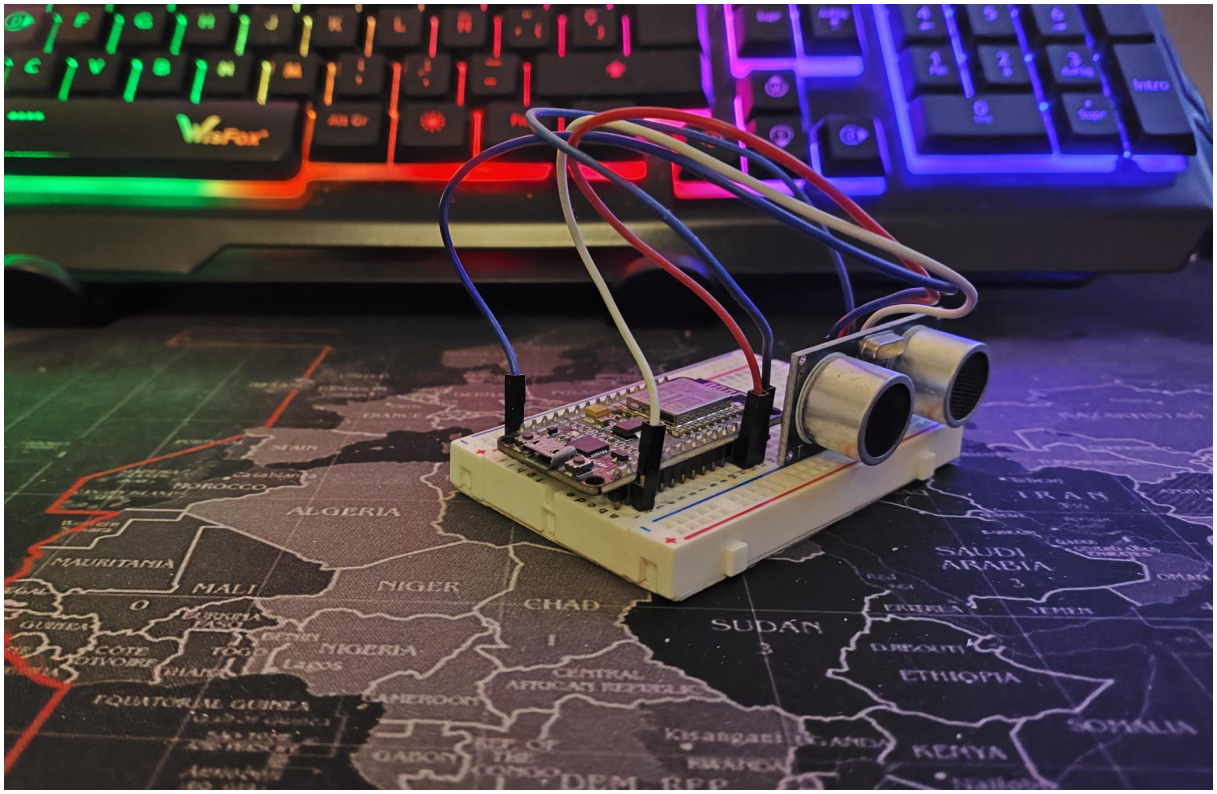


Figura 49: Montaje del dispositivo físico

las medidas:

$$343\frac{m}{s} * \frac{100cm}{1m} * \frac{1s}{1000000\mu s} = 0,0343\frac{cm}{\mu s}$$

A demás, tenemos que tener en cuenta de que la distancia que estamos calculando es desde que la onda sale del sensor hasta que es detectado de nuevo, es decir, 2 veces la distancia al objetivo, por lo que:

$$v * t = s \longrightarrow v * t = 2d \longrightarrow 0,0343t = 2d \longrightarrow \frac{0,343}{2}t = d \longrightarrow 0,1715t = d$$

```
long measureDistance() {
  long  t;
  long  d;

  // Mandar senal de 10 microsegundos para activar el sensor
  digitalWrite(TRIG, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIG, LOW);

  // Esperar la deteccion del sensor
  t = pulseIn(ECHO, HIGH);

  // Formula para determinar la distancia
  d = t*0.1715;
  return (d);
}
```

Con esta función podemos construir el detector de personas. Vamos realizando medidas cada 100 milisegundos, por encima de los 60 milisegundos que el fabricante recomienda dejar de tiempo para el correcto funcionamiento del sensor.

Como criterio para determinar si alguien está pasando, se ha utilizado un umbral. Si la distancia detectada es menor que ese umbral, significa que alguien ha pasado por delante. Por supuesto, mientras esa persona está pasando pueden seguir ocurriendo muchas medidas, por lo que mientras esa persona está delante, tenemos que evitar que lo contabilice como otra persona que está pasando, por ello se utiliza la variable “someoneInTheMiddle”, para determinar cuando ha terminado de pasar esa persona.

```
void loop() {
  long m;

  m = measureDistance();
  if (m < THRESHOLD && !someoneInTheMiddle) {
    numberOfPerson++;
    notifyDetection(numberOfPerson);
    someoneInTheMiddle = true;
  } else if (m >= THRESHOLD && someoneInTheMiddle) {
    someoneInTheMiddle = false;
  }
  delay(100);
}
```

5.3.4. Registro de la medida en Orion

Este sección del código será muy relevante, ya que es la que dota al dispositivo de esa conexión a internet y alimentará al sistema con sus datos. Para establecer conexión con internet, se ha utilizado la librería "ESP8266WiFi" la cual permitirá establecer la conexión con el router para que el dispositivo tenga acceso a internet.

```
void setupWiFi() {
  WiFi.persistent(false);
  WiFi.begin(WIFI_SSID, WIFI_PSWD);
  while (WiFi.status() != WL_CONNECTED) {
    delay(WIFI_DELAY);
    Serial.print(".");
  }
  Serial.println();
  Serial.println("Conexion establecida con exito!");
  Serial.println("Direccion IP = " + WiFi.localIP().toString());
}
```

La librería "ESP8266HTTPClient" permite mandar peticiones por HTTP/HTTPS a servidores. Con esto podemos preparar una petición para registrar en FIWARE los datos que el sensor emite:

```
  ORION_LD_ENDPOINT = "https://orion.ruzafa.me/"
```

```
url = ORION_LD_ENDPOINT + "/ngsi-ld/v1/entities/";
http.begin(client, url);
http.addHeader("Content-Type", "application/json");
http.addHeader("Link",
 "<https://saref.etsi.org/saref4bldg/v1.1.2/saref4bldg.jsonld>;
 rel=\"http://www.w3.org/ns/json-ld#context\";
 type=\"application/ld+json\"");
responseCode = http.POST(body);

if (responseCode > 0) {
  Serial.print("Solicitud POST enviada, codigo de respuesta: ");
} else {
  Serial.print("Error al enviar la solicitud, codigo de error: ");
}
Serial.println(responseCode);
http.end();
```

Según SAREF el sensor debe registrar las medidas en una entidad de tipo Measurement, este debe tener el atributo *saref:hasValue* con el valor de la medida y puede estar referenciado a una entidad Device (incluidas herencias) a través de la relación *saref:isMeasurementOf*. Gracias a la librería "ArduinoJson.h", podemos construir el JSON que SAREF nos indica:

```
StaticJsonDocument<400> json;
String body;
...

json["id"] = "urn:ngsi-ld:Measurement:uma:FrontalBuilding:"
 + generateRandomUniqueId();
json["type"] = "s4bldg:Measurement";
json["s4bldg:hasValue"] = value;
JsonObject measurementNode = json.createNestedObject(
  "s4bldg:measurementMadeBy");
measurementNode["type"] = "Relationship";
measurementNode["object"] = DEVICE_NAME;
serializeJson(json, body);
...
```

6

Conclusiones y líneas de trabajo futuras

A lo largo del desarrollo del proyecto se ha hablado de las ventajas y desventajas sobre el IoT y se han ido planteando incógnitas que se han ido estudiando a lo largo del proyecto. Gracias al trabajo realizado con FIWARE hemos visto cómo esta herramienta hace uso de las ventajas de IoT y qué herramientas ofrece para solventar los problemas. En esta sección enumeraremos y resumiremos esas características de FIWARE y veremos qué otras problemáticas plantea.

6.1. Plataforma de FIWARE

FIWARE es una solución de IoT y como tal debe tener en cuenta las características de este ámbito para explotar las ventajas y mitigar los problemas. En las primeras secciones ya hemos visto que encuentra solución para uno de los problemas más importantes de IoT: **la estandarización**. Dedicaremos secciones a hablar de los datos abiertos y de la API NGSI que recomiendan y apoyan como interfaz estándar varias entidades importantes: la ETSI, GSMA, la Comisión Europea o el programa nación de Smart City de India entre otros.

- Datos abiertos: Estandarizaciones reconocidas de manera internacional orientadas a normalizar los datos en muchos campos del munsdo *smart*, que, además, disponen de ontologías a otros modelos también estándares y reconocidos.

- Interfaz NGSI: Resuelve diferentes aspectos:

- Compatibilidad con la semántica web, lo cual permite estructurar los datos sistemáticamente.
- Ofrece herramientas para obtener los datos filtrados y ordenados de muchas formas.

- `IoT Agents`: Estos microservicios son adaptadores de protocolos que permiten traducir la comunicación entre NGSI y muchos otros protocolos, permitiendo ampliar la comunicación a dispositivos de terceros que no estaban pensados para conectarlos con FIWARE.

Otro problemas que se han planteado a lo largo del desarrollo del proyecto es el de la seguridad. Al tener una red distribuida de dispositivos es más difícil tener el control de la autenticación de los datos y la protección del mismo. FIWARE está preparado para solucionar estos problemas gracias a una familia de `Generic Enablers` cuyo objetivo es asegurar el entorno. Existen multitud de microservicios a elegir con diferentes características que hacen que el sistema se adapte muy bien a las necesidades: Wilma, AuthZForce, o Kong.

Por otro lado también podemos ver cómo con FIWARE podemos hacer uso de todas esas ventajas descritas de IoT: Computación en el borde, orquestación de dispositivos, análisis de datos o gemelos digitales, por lo que podemos concluir que FIWARE es una herramienta acertada si queremos trabajar con el IoT, ya que todos sus componentes están orientados a solucionar estos problemas y hacer eso de sus características.

6.2. Problemas encontrados durante el desarrollo

A lo largo del desarrollo del proyecto han tenido lugar una serie de problemas inesperados que han provocado un replanteamiento de partes del proyecto o la búsqueda de una solución alternativa.

Por ejemplo, el despliegue de la plataforma de FIWARE ha sido uno de los principales problemas a la hora de realizar el proyecto:

- La incompatibilidad con algunas arquitecturas.
- Documentación muy buena para los apartados introductorios con tutoriales muy claros, pero muy pobres en las secciones más avanzadas.

- Existen muchos errores en producción. Características que no funcionan como se describe en la documentación o características que son anti-intuitivas y no están documentadas. Por ejemplo, en los filtros en NGSI, el orden de las coordenadas está invertido. En el proyecto en GitHub se pueden encontrar muchos problemas documentados que se planea arreglar en futuras versiones.

- La arquitectura de servicios basada en microservicios bastante compleja. Muchos microservicios tienen configuraciones complejas y poco documentadas, además de ser más complicado de analizar los errores.

- Inconsistencia entre los IoT Agent y la ontología SAREF: IoT Agent tiene una entidad Device por cada dispositivo donde va actualizando los atributos, pero SAREF nos dice que debemos crear una entidad Measurement por cada medida. Esta inconsistencia hace que o bien tengamos que prescindir de parte del modelo de SAREF o tengamos que hacer uso de otro tipos de adaptadores, por ejemplo, uno personalizado.

- Tras empezar con el desarrollo de los requisitos, la adaptación del entorno de FIWARE a ciertos requisitos es más avanzada de lo esperado, así que se decidió desarrollar un subconjunto de requisitos, lo necesario para demostrar con una demo el funcionamiento básico de la plataforma desde diferentes puntos del proyecto: producción de datos y consumo de la API.

Por otro lado, además de los problemas de FIWARE, también nos hemos encontrado con problemáticas típicas de muchos desarrollos:

- Las CORS para el desarrollo en local y en los servidores.

- Mezclado de webs HTTPS y HTTP

- Recursos muy limitados. La máquina no permitía tener todos los microservicios: autenticación y Orion ejecutándose a la vez.

- Problemas de memoria de los microservicios.

Además, las configuraciones de los despliegues del frontal también resultaron problemáticas. Hubo problemas con el Server Side Rendering y las configuraciones del proyecto a la hora de compilar relacionados con el tipo de compilación

6.3. Conclusiones del uso FIWARE

Hemos visto cómo por un lado FIWARE es capaz de encontrar solución para muchos problemas de los planteados sobre IoT, sin embargo por otro lado existen muchas problemáticas que hace el desarrollo muy complejo. El problema principal es que se trata de una arquitectura gigantesca, una multitud de microservicios que están en continuo desarrollo, con una documentación a veces escasa que pretenden resolver muchos problemas e intentan abstraerse de lógicas de negocio para ser utilizable por cualquier modelo. Hacer algo tan genérico para resolver unos problemas tan complejos es prácticamente imposible sin que la solución sea compleja, sin embargo también hemos visto que es una herramienta muy poderosa y escalable.

De hecho hemos visto que nuestra solución implementada no cubre todos los requisitos, los relacionados con seguridad, o protección de los recursos. Ya que, aunque el frontal sí valida el acceso a los datos, desde el API cualquiera puede hacer una petición y FIWARE devolverá todos los resultados sin hacer ninguna consulta a ningún PDP.

Seria recomendable usar FIWARE si necesitamos una solución grande y muy escalable y disponemos de muchos recursos para desplegarlo todo, además de los conocimientos necesarios para ampliar, desarrollar y mantener la arquitectura.

6.4. Lineas de trabajo futuras

A lo largo del desarrollo del TFG se ha hablado de diversos puntos que no se han decidido implementar por complejidad, pero que para una solución realista son importantes a tener en cuenta. Todos estos puntos pueden ser interesantes de abordar como futuras líneas de trabajo. A continuación los recopilamos:

- El sistema actual posee una granularidad de tipo 1, se podría configurar y hacer uso de un PDP con una autorización de grano fino.
- Aunque existían ciertas validaciones de acceso a los datos, estas se estaban haciendo en el frontal. La API no contaba con ninguna estrategia para validar el acceso a los datos. Se podría añadir la configuración un proxy PEP para forzar la comprobación de los permisos a nivel de API.
- Una alterativa a un proxy PEP más moderna sería utiliza un microservicio que actúe

de API Gateway. Que, además del servicio de proxy PEP, ofrece otras ventajas como monitorización y control del tráfico. Se podrían explotar estas ventajas.

- Extender el modelo de datos de SAREF para que se adapte mejor a las necesidades de la plataforma.
- Utilizar más clases de SAREF para representar más información de los edificios.
- Se pueden añadir el módulo de computación en el borde con Fog Flow.
- Desarrollar más dispositivos físicos para conectarlos y quizás usando otras interfaces.
- Integrarlos con sistemas reales, por ejemplo, una bombilla IoT que ya se comercialice con una API propio.

Referencias

[1] Alberto Abella et al. «FIWARE for Digital Twins». En: *FIWARE Foundation e.V.* (jun. de 2021). Ed. por Juanjo Hierro.

[2] Álvaro de Cózar et al. *Smart Building using oneM2M protocol*. Nov. de 2021. URL: https://www.hackster.io/wsw/smart-building-using-onem2m-protocol-c2e7a9 (visitado 30-08-2023).

[3] Ayman Elnashar y Mohamed El-saidny. «IoT evolution towards a super-connected world». En: *Practical Guide to LTE-A, VoLTE and IoT: Paving the way towards 5G*. Wiley, 2018. Cap. 7, págs. 310-381. ISBN: 9781119063407. DOI: 10.1002/9781119063407.ch7. URL: https://doi.org/10.1002/9781119063407.ch7.

[4] Elsa Estevez, Theresa Pardo y Jochen Scholl. *Smart cities and smart governance: towards the 22nd century sustainable city*. Springer, 2021. ISBN: 3-030-61033-0.

[5] FIWARE Foundation. *FIWARE - Open APIs for Open Minds*. 2023. URL: https://www.fiware.org/ (visitado 30-08-2023).

[6] FIWARE Foundation. *Getting Started With NGSI-V2*. 2023. URL: https://fiware-orion.readthedocs.io/en/3.10.1/orion-api.html (visitado 30-08-2023).

[7] FIWARE Foundation. *Getting Started With NGSI-V2*. 2023. URL: https://fiware-tutorials.readthedocs.io/en/latest/getting-started.html (visitado 30-08-2023).

[8] FIWARE Foundation. *IoT Agents & Robots*. 2023. URL: https://fiware-tutorials.readthedocs.io/en/latest/iot-agent.html (visitado 03-09-2023).

[9] Juan Marcelo Gaitan, Daniel Villalba y Miguel Ángel Pedraza. *FIWARE ZONE - Introducción a FIWARE*. Youtube. Jul. de 2022. URL: https://www.youtube.com/watch?v=qeR0BbVKwKM&ab_channel=FIWAREZone.

[10] Daniel García Martínez. «Gemelo Digital». En: *UEM STEAM Essentials* (2022). URL: https://universidadeuropea.com/resources/media/documents/23_STEAM_GEMELODIGITAL_RZ2023.pdf.

[11] Phu Nguyen et al. «Advances in Deployment and Orchestration Approaches for IoT - A Systematic Review». En: *2019 IEEE International Congress on Internet of Things (ICIOT)*. 2019, págs. 53-60. DOI: 10.1109/ICIOT.2019.00021.

[12] oneM2M. *oneM2M REST RESOURCES*. 2023. URL: https://onem2m.org/using-onem2m/developers/rest-resources (visitado 30-08-2023).

[13] oneM2M Partners. *Benefits of oneM2M*. 2023. URL: https://onem2m.org/using-onem2m/what-is-onem2m?jjj=1707771954468 (visitado 12-01-2024).

[14] Alejandro José Ruzafa Casás y Andrés Morilla Morilla. *HACKATON 2021 IOT MALAGA SWS GROUP "IOT ELEVATOR"*. Youtube. Abr. de 2021. URL: https://www.youtube.com/watch?v=zSkWcmKkTvA.

[15] Paulo Shakarian. «Stuxnet: Revolución de Ciberguerra en los Asuntos Militares». En: *Air and Space Power Journal* (ene. de 2012).

[16] Amjad Hameed Shehab y Sufyan Al-Janabi. «Edge Computing: Review and Future Directions (Computación de Borde: Revisión y Direcciones Futuras)». En: *REVISTA AUS Journal* (sep. de 2019), págs. 368-380. URL: https://ssrn.com/abstract=3459649.

[17] UNE. «Sistemas Integrales de Gestión de la Ciudad Inteligente. Requisitos de interoperabilidad para una Plataforma de Ciudad Inteligente». En: *Advanced technical ceramics. Monolithic ceramics. Gerneral and textural properties.* (dic. de 2017), págs. 1-16.

Apéndice A
Configuraciones de Docker

Docker Compose es una herramienta que permite definir y gestionar aplicaciones compuestas por múltiples contenedores de Docker. Facilita la configuración y el despliegue de aplicaciones que requieren la ejecución simultánea de varios servicios o componentes interrelacionados.

En lugar de gestionar manualmente cada contenedor por separado utilizando comandos de Docker, Docker Compose utiliza un archivo YAML para definir la configuración de los servicios, como imágenes de contenedores, variables de entorno, volúmenes, puertos expuestos y redes.

Mediante la definición en este archivo YAML, los desarrolladores pueden describir la estructura de la aplicación y cómo sus diferentes componentes interactúan entre sí. Con un simple comando, Docker Compose puede crear, iniciar y detener todos los contenedores definidos en el archivo de configuración, lo que simplifica significativamente el proceso de administración de aplicaciones complejas con múltiples servicios.

Además, Docker Compose facilita la escalabilidad y la replicación de entornos de desarrollo, ya que los desarrolladores pueden compartir fácilmente archivos de configuración de Compose para garantizar la coherencia en diferentes máquinas.

En resumen, Docker Compose es una herramienta útil para definir, configurar y gestionar aplicaciones compuestas por varios contenedores de Docker, simplificando su administración y despliegue.

A.1. Configuración de Orion-LD

Se puede encontrar un ejemplo de Docker Compose en el repositorio de git de fiware-orion. A parte de las configuraciones por defecto, añadí lo siguiente:

- Un volumen para persistir los datos de la base de datos de forma externa a los contene-

dores, por si al hacer pruebas hay que volver a desplegar contenedores, y no se pierden los datos.

- Un parámetro de depuración para mostrar información útil durante las pruebas. Esto está especificado en "command", que es un comando que se ejecutará automáticamente al crear la instancia del contenedor.

```
version: "3.5"
services:
  orion-ld:
    image: fiware/orion-ld:1.4.0
    hostname: orion-ld
    ports:
      - "1026:1026"
    depends_on:
      - mongo-db
    command: -dbhost mongo-db-vm -logLevel DEBUG

  mongo-db:
    image: mongo:4.2
    hostname: mongo-db-vm
    volumes:
      - ~/data:/data
```

En la entrada "ports" se observa el mapeo del puerto de la instancia a la de su máquina host. En este caso, estamos diciendo que todos los paquetes que lleguen al puerto 1026 en la máquina real, se redireccionen al puerto 1026 en la instancia del contenedor. De esta forma, y como habíamos configurado anteriormente la máquina virtual para tener acceso al puerto 1026, deberíamos de poder obtener una respuesta de orion si hacemos una petición al puerto 1026.

Un apunte de seguridad. En este caso, como son servidores de pruebas, estoy usando los puertos por defecto, pero en una situación real, sería más adecuado utilizar un puerto diferente. Por simplificar las pruebas, decidí usar el puerto por defecto ya que en todas las configuraciones que necesite, no será necesario especificar el puerto, ya que por omisión, está establecido que se utilice este.

A.2. Configuración de Keyrock

Para la configuración de base de datos, existen opciones para configurar por las variables de entorno un usuario para usarlo, pero daba muchos problemas de permisos. Intentamos conectarnos a la instancia de base de datos para dar los permisos oportunos pero seguíamos teniendo problemas con el permiso del host. Finalmente decidimos usar el usuario de root, ya que solucionar el problema llevó demasiado y, al la instancia de la base de datos, tendríamos que volver a realizar manualmente cada vez esos cambios. De hecho al ejecutar el Docker da problemas porque no existe la base de datos.

```
version: '3.5'
services:
    keyrock:
        image: fiware/idm:8.4.0
        container_name: fiware-keyrock
        hostname: keyrock
        networks: {default: {ipv4_address: 172.19.1.5}}
        depends_on: [mysql-db]
        ports: ['3000:3000']
        environment:
            - DEBUG=idm:*
            - IDM_DB_HOST=mysql-db
            - IDM_HOST=http://localhost:3000
            - IDM_PORT=3000
            # Development use only
            # Use Docker Secrets for Sensitive Data
            - IDM_DB_PASS=...
            - IDM_DB_USER=root
            - IDM_ADMIN_USER=skywalker
            - IDM_ADMIN_EMAIL=skywalker@ruzafa.me
            - IDM_ADMIN_PASS=...
            # If sending eMails point to any STMP server
            - IDM_EMAIL_TRANSPORT=smtp
            - IDM_EMAIL_HOST=mailer
            - IDM_EMAIL_PORT=25
            - IDM_EMAIL_ADDRESS=fiware@ruzafa.me
            # IDM Cors
```

```
28              - IDM_CORS_ENABLED=true
29      mysql-db:
30          restart: always
31          image: mysql:8.0
32          hostname: mysql-db
33          container_name: db-mysql
34          expose: ['3306']
35          ports: ['3306:3306']
36          networks: {default: {ipv4_address: 172.19.1.6}}
37          environment:
38              # Development use only
39              # Use Docker Secrets for Sensitive Data
40            MYSQL_ROOT_PASSWORD: ...
41            MYSQL_ROOT_HOST: '172.19.1.5'
42            MYSQL_DATABASE: idm
43          volumes:
44            - mysql-db:/var/lib/mysql
45
46      mailer:
47          restart: always
48          image: mazdermind/docker-mail-relay
49          hostname: mailer
50          container_name: mailer
51          ports: ['25:25']
52          environment:
53              - RELAY_HOST_NAME=mailer
54              - SMTP_LOGIN=...
55              - SMTP_PASSWORD=...
56              - EXT_RELAY_HOST=...
57              - EXT_RELAY_PORT=...
58              - ACCEPTED_NETWORKS=172.19.1.0/24
59              - USE_TLS=yes
60  networks: {default: {ipam: {config: [subnet: 172.19.1.0/24]}}}
61  volumes: {mysql-db: ~}
```

Apéndice B
Interfaces NGSI-LD

En este apartado vamos a entrar un poco en detalle en las interfaces NGSI-V2 y NGSI-LD para ver sus diferencias fundamentales.

B.0.1. Interfaz NGSI-V2

Tiene un modelo de datos sencillo, el cual se puede observar en la figura 50. Una entidad tendrá un identificador y un tipo, representados con una cadena de caracteres. Con todo esto, NGSI-V2 nos dice que con otorgarle un identificador y un tipo, ya deberíamos de ser capaces de tener un contexto suficiente para entender el dato. Estas entidades podrán tener, atributos, que tendrán un nombre, un valor y un tipo y estos a su vez metadatos. Por ejemplo, un atributo podría ser edad, en formato JSON:

```
1 { "name": "edad", "type": "number", "value": 33 }
```

De esta forma podremos otorgar a una entidad todos los atributos que queramos.

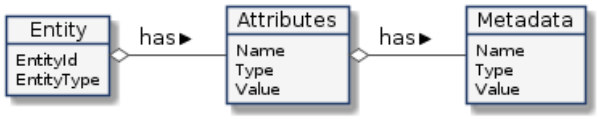


Figura 50: Modelo de datos NGSI-V2, extraídos de [6]

La idea de categorizar una entidad con un tipo es muy útil para determinar qué posibles atributos podrá tener una entidad pero a pesar de esto, pueden surgir ciertos problemas que dificulten la contextualización: ¿Qué ocurre si dos servicios diferentes definen un tipo pero cada uno le da atributos diferentes? ¿Cómo puedo asegurarme de que hay consistencia dentro de un tipo concreto? Además, el uso de identificadores no tiene restricciones de reglas por lo que no se puede asegurar la unicidad del mismo de manera global, lo cual puede dar lugar a ambigüedades por toda la red. La interfaz NGSI-LD, gracias al uso de los datos enlazados, nos resuelve estos problemas

B.0.2. Interfaz NGSI-LD

Basándose en los datos enlazados (LD, en inglés *Linked Data*), NGSI-LD puede resolver los problemas de contextualización, inconsistencia y ambigüedad. Los datos enlazados describen un método de publicación de datos estructurados para que puedan ser interconectados y más útiles, básandose en los siguientes principios:

- El uso de identificadores de recursos uniforme (URI, en inglés *Uniform Resource Identifier*) para identificar los recursos publicados en la Web
- Acceso a los recursos utilizando el URI con HTTP.
- Proporcionar información útil sobre el recurso cuando la URI haya sido desreferenciada.
- Incluir enlaces a otras URI relacionadas con los datos contenidos en el recurso, de forma que se potencie el descubrimiento de información en la Web.

Para más detalles de la implementación, refiérase a la documentación oficial [22].

Por otro lado, se puede observar que diferencia fundamental con respecto a NGSI-V2 radica en su modelo de datos. Véase figura 51. A priori podemos ver que este modelo de datos es más potente ya que representa más aspectos además de las entidades, atributos y metadatos. En este modelo de datos también hablamos de relaciones.

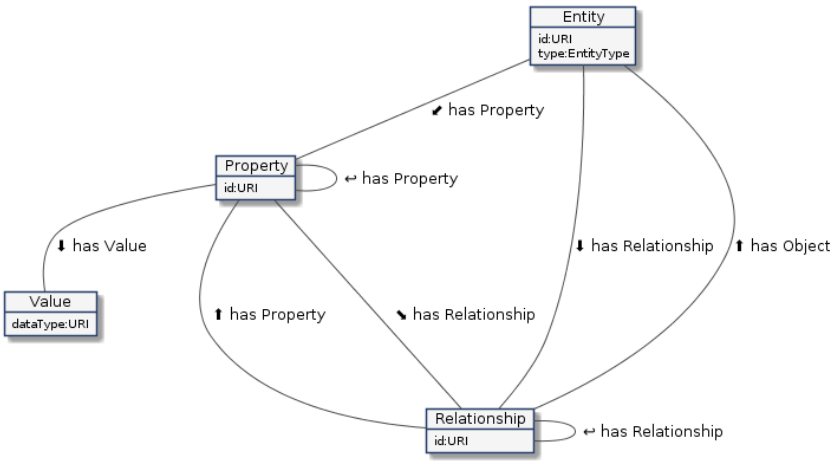


Figura 51: Modelo de datos NGSI-LD, extraídos de [6]

[22]https://www.etsi.org/deliver/etsi_gs/CIM/001_099/009/01.06.01_60/gs_CIM009v010601p.pdf

B.1. JSON-LD

Si bien JSON (en inglés, *JavaScript Object Notation*) es un formato estándar de intercambio de datos, puede resultar difícil interpretar el significado de los datos si no están documentados de manera externa. Aquí es donde entra en juego JSON-LD, una extensión de JSON que incluye anotaciones enlazadas (metainformación). Gracias a los datos enlazados, se definen tipos de datos de manera estándar lo que permite una mejor compresión del significado de cada dato incluido en el JSON. Con esto podemos ser capaces de, por ejemplo, interpretar un JSON con un campo *name* que, a priori, no se sabe si se trata del nombre del usuario de una plataforma o el nombre real de esa persona.

```
JSON: {
    "name": "alan turing"
}
JSON-LD: {
    "@id": "https://uma.es/res/etsii/tfg/2023/Alan_Turing",
    "@context": "https://json-ld.org/contexts/person.jsonld",
    "name": "alan turing"
}
```

En JSON-LD, las anotaciones enlazadas vienen precedidas de una arroba ("@"), como en el ejemplo anterior, donde "@id" es una URI del dato y "@context" contiene una URI que referencia a un diccionario que ayuda a interpretar los campos del JSON, asociando nombres con URI donde se le otorga una definición de manera estándar.

Finalmente, podemos decir que NGSI-LD es una evolución de NGSI-V2 con soporte para Linked Data. Es una extensión de JSON-LD, por lo que, sabiendo que NGSI-V2 es JSON, solo es necesario convertirlo a JSON-LD para implementar esa característica. NGSI-LD, define un "@context" por defecto denominado "Core NGSI-LD Context" que se puede consultar desde la web de la ETSI [23], por lo tanto no es necesario incluirlo en el "@context".

Este "core NGSI-LD context" crea alias con los esquemas más comunes, como las fechas o las posiciones de geolocalización. También crean alias para los campos *id* y *type*, que ya se usaban en NGSI-V2, dándole el significado que tienen "@id" y "@type" respectivamente en

[23] https://uri.etsi.org/ngsi-ld/

JSON-LD.

```
{
    "date": "https://uri.etsi.org/ngsi-ld/Date",
    "geojson": "https://purl.org/geojson/vocab#"
    "id": "@id",
    "type": "@type"
}
```

B.2. Operaciones de filtrado y ordenación en NGSI

Una de las ventajas de este modelo, es que una vez almacenados los datos, tendremos la posibilidad de filtrar por los atributos que queramos, pudiendo especificar en el parámetro "q" una consulta en "NGSI-LD Query Language" pudiendo filtrar, por ejemplo, todos los edificios de tipo Hospital.

```
?type=Building&q="category==Hospital"
```

También dispone de un lenguaje de consultas "NGSI-LD Geo-query Language" que permite hacer especificaciones relativas a la geolocalización de los objetos. Estos objetos deben tener la propiedad de posición para que tenga sentido. Por ejemplo, podemos buscar los objetos más cercanos a mi posición a menos de 2 km:

```
?type=Building&georel="near;maxDistance==2000"
&geometry=Point&coordinates=[12,42]
```

Incluso es posible especificar filtros relacionados con la temporalidad con el lenguaje de consulta "NGSI-LD Temporal Query Language". En este caso, debemos especificar en un parámetro de la consulta sobre qué atributo del objeto hacemos la pregunta temporal. Por ejemplo, puedo preguntar por fruterías que estén abiertas en la fecha indicada:

```
?timerel=before&time=2024-03-29T14:20:00Z&timeproperty=open
```

También existe un parámetro para ordenar la respuesta y traer, por ejemplo, aquellos hospitales ordenados por la disponibilidad.

```
?orderBy=availabilitySpace
```

Combinando todos los parámetros comentados, podríamos tener una petición que nos traiga todos los hospitales que estén a menos de 2km de nuestra posición que estén abiertos y ordenados por disponibilidad.

```
?type=Building&q="category==Hospital"
&georel="near;maxDistance==2000"
&geometry=Point&coordinates=[12,42]
&timerel=before&time=2024-03-29T14:20:00Z&timeproperty=open
&orderBy=availabilitySpace
```

Apéndice C
Ontología SAREF

SAREF tiene una serie de modelo de datos básico para definir todo lo relacionado con dispositivos y sensores, denominado el núcleo de SAREF, *SAREF Core* en inglés. En este modelo se definen entidades como *saref:Device*, que tienen una*saref:Property* (propiedad del dispositivo, por ejemplo la temperatura si es un dispositivo sensor de temperatura). También tienen un *saref:Measurement*, que está relacionado tanto con *saref:Property* como con *saref:Device*, y representa una medición de una propiedad de un dispositivo. Este modelado de relaciones de entidades se puede observar en la figura 52.

Esta genericidad permite incluso definir un dispositivo que simplemente tenga medidas, sin propiedad, ya sea porque se sobreentiende o porque no es relevante.

Un *saref:Device* también puede tener *saref:Task*, que representa una tarea que puede realizar el dispositivo, *saref:Service*, que representa un servicio que puede ofrecer el dispositivo, *saref:Function*, que representa una función que puede realizar el dispositivo, y *saref:Command*, que representa una orden que se le puede dar al dispositivo. Además, un *saref:Device* puede tener *saref:State*, que representa un estado del dispositivo.

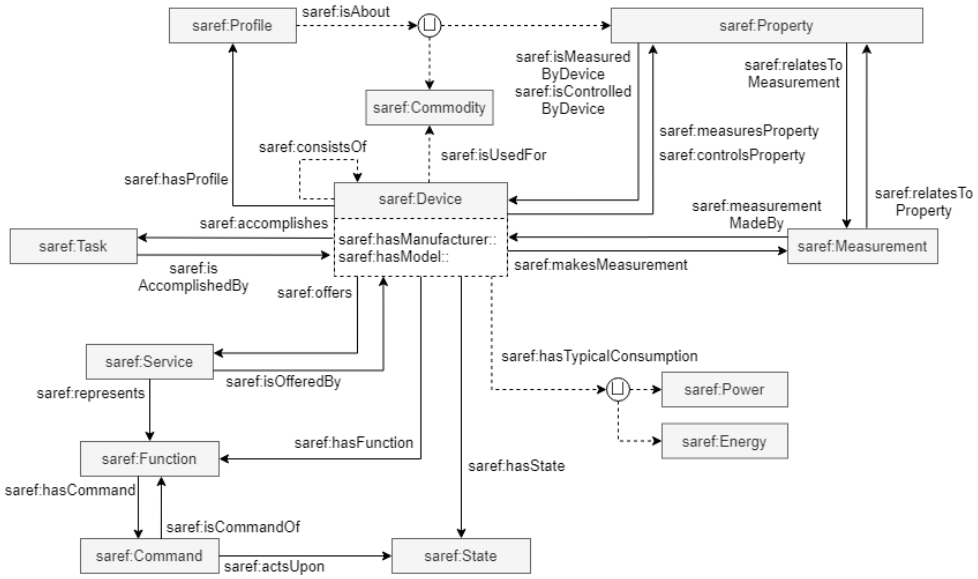


Figura 52: Núcleo de SAREF

Se define también una herencia dentro de un *saref:Device*, como observamos en la figura 53, de forma que podemos clasificar los tipos de dispositivos:

- Actuadores - *saref:Actuator* - Dispositivo que actúa sobre el entorno.
- Sensores - *saref:Sensor* - Dispositivo que mide una propiedad del entorno.

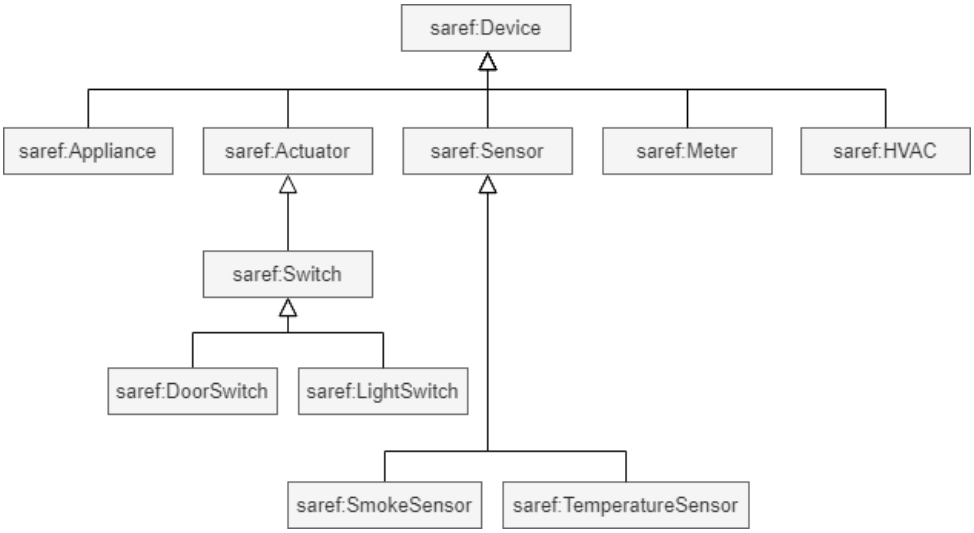


Figura 53: Herencia de dispositivos en SAREF

Por otro lado, dentro de las extensiones de SAREF, se encuentra la de Edificios, denominado "SAREF4BLDG", que define los conceptos básicos para representar edificios y espacios dentro de estos, y, aplicando el concepto de reutilización, se hacen referencia a los conceptos de SAREF Core para conectar los edificios con los dispositivos y sensores que contienen.

Un saref4bldg:Building tiene *saref4bldg:BuildingSpace*, que representa un espacio dentro del edificio, el cual tiene *saref4bldg:PhysicalObject*, que puede ser un saref:Device. Estos tres son *geo:SpatialThing*, que es un concepto de la ontología Geo, relacionada con la representación de la ubicación de los objetos. Estas relaciones se pueden observar en el diagrama presente en la figura 54.

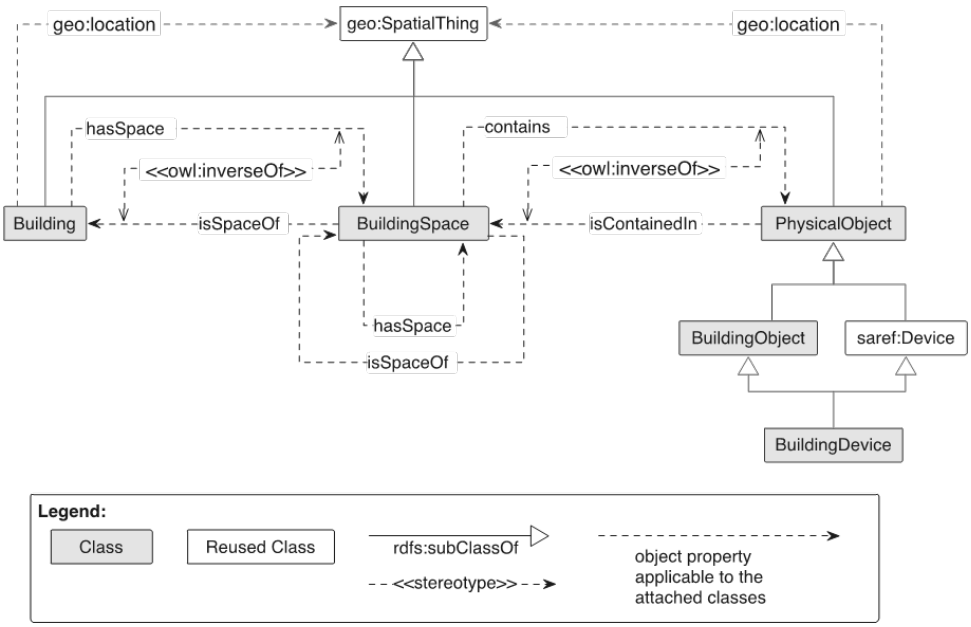


Figura 54: Extensión de SAREF para edificios

UNIVERSIDAD DE MÁLAGA | uma.es

E.T.S de Ingeniería Informática
Bulevar Louis Pasteur, 35
Campus de Teatinos
29071 Málaga

E.T.S. DE INGENIERÍA INFORMÁTICA