UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

GRADUADO EN INGENIERÍA DEL SOFTWARE

SISTEMA PARA DEFINIR APLICACIONES DE INTERNET DE LAS COSAS TOLERANTES A FALLOS A NIVEL SOFTWARE Y HARDWARE

A SYSTEM TO DEFINE FAULT-TOLERANCE IoT APPLICATIONS AT SOFTWARE AND HARDWARE LEVELS

Realizado por

Gabor, Stefan

Tutorizado por

Garrido Márquez, Daniel

Martín Fernández, Cristian

Departamento

Lenguajes y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA

MÁLAGA, JUNIO DE 2019

Fecha defensa:

Fdo. El/la Secretario/a del Tribunal

Resumen

El Internet de las Cosas (IoT) hace referencia al conjunto de objetos conectados a través de internet que tienen la capacidad de sentir y actuar sobre el entorno, logrando una representación digital del mundo físico. Estos dispositivos (IoT) presentan una gran volatilidad debido a su naturaleza y están sometidos a una gran cantidad de interrupciones de servicio (p.ej. agotamiento de batería). En el contexto de aplicaciones críticas, no se requiere únicamente una alta fiabilidad en el nivel del Software sino también en la parte Hardware. Durante los últimos años el IoT ha sido integrado en Internet gracias a arquitecturas multicapa, como la arquitectura cloud-fog-edge, que permite acercar la lógica de las aplicaciones lo más cerca posible de los dispositivos, reduciendo la latencia y el ancho de banda en las comunicaciones con el IoT.

Este Trabajo se centra en la aplicación de técnicas de tolerancia a fallos para sistemas IoT en una arquitectura cloud-fog-edge, en concreto, se persigue la definición de lo que denominamos “dispositivos Shadow”: dispositivos virtuales dentro de una aplicación IoT que se caracterizan por tener una serie de propiedades (estados) resultantes de monitorizar propiedades del mundo real. Por otra parte, también pueden recibir comandos para actuar sobre el entorno. Estos dispositivos estarán asociados a un dispositivo real, de manera que, si dejase de funcionar, podrá ser sustituido por otro con características similares, permitiendo a la aplicación IoT que siga funcionando. Estos dispositivos Shadow estarán también replicados usando contenedores, de manera que, si ocurre algún fallo en la máquina donde esté almacenado el dispositivo virtual, pueda ser sustituido por otra réplica restaurándose su estado.

Palabras clave:

Internet de las cosas, IoT, Nube, Middleware, Crítico, Disponibilidad, Docker, Tolerancia, Multicapa, Niebla, Replicado, Virtual.

Abstract

The Internet of Things (IoT) refers to the set of objects connected through Internet that are sensible to the environment and are also able to perform actions over it, so that, we get a digital representation of the real world. These devices (IoT) are very volatile due to their nature, so, they suffer a huge amount of service interruptions (e.g. battery power drain). In the field of critical applications, a high availability is required both at software and hardware level. During the last years the IoT has been integrated into Internet thanks to a multilayer architecture like cloud-fog-edge, which allows us to bring the application logic closer to the IoT devices resulting in a reduction of the latency and the bandwidth in the communications with the IoT devices.

This Project focuses on the application of fault-tolerance techniques for IoT systems on a cloud-fog-edge architecture, to be more precise, the definition of "shadow devices" is pursued. A Shadow device is a virtual device that is part of an IoT application, and it's characterized by having a set of properties (states) resulting from monitoring properties of the real world. On the other hand, these devices can also receive commands to act over the environment. They will be associated to a real device, so, if one stops working, another real device with similar specifications will be replaced, allowing the IoT application to keep working. The shadow devices will also be replicated, so, if an error occurs in the real device that is hosted by the virtual device, it could be replaced by another replica restoring its status.

Keywords:

Internet of Things, IoT, Cloud, Middleware, Critical, Availability, Docker, Tolerance, Multilayer, Fog, Replicate, Virtual.

UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA

Índice

1

Introducción

1.1 Motivación

El Internet de las Cosas (IoT) hace referencia al conjunto de objetos conectados a través de Internet que tienen la capacidad de sentir y actuar sobre el entorno, logrando una representación digital del mundo físico. Este paradigma ha tenido una gran repercusión durante los últimos años, con notables ejemplos como las ciudades inteligentes y la Industria 4.0. A pesar de que el IoT permite una digitalización del mundo físico, los dispositivos que forman parte de él presentan una gran volatilidad debido a su naturaleza y están sometidos a una gran cantidad de interrupciones de servicio, como el caso de poca cobertura temporalmente en dispositivos móviles. Por ejemplo, en aplicaciones críticas, tales como la monitorización estructural de infraestructuras, no se requiere únicamente una alta fiabilidad de los sistemas software adheridos, sino también de los dispositivos que forman parte de ellas, ya que un fallo en los sistemas de monitorización o actuación puede llevar a completas interrupciones de los servicios.

Durante los últimos años, el IoT ha sido integrado en Internet gracias a arquitecturas multicapa, como la arquitectura cloud-fog-edge, que permite acercar la lógica de las aplicaciones lo más cerca posible de los dispositivos, y por tanto reducir la latencia y el ancho de banda en las comunicaciones con los dispositivos. La consecución de tolerancia a fallos a nivel hardware a lo largo de esta arquitectura permitiría obtener los beneficios que trae consigo este paradigma a la vez que una mayor fiabilidad, lo que sería deseable para aplicaciones críticas.

Este Trabajo se centra en la aplicación de técnicas de tolerancia a fallos para sistemas IoT en una arquitectura cloud-fog-edge. En concreto, se persigue la definición de lo que denominamos dispositivos Shadow: dispositivos virtuales dentro de una aplicación IoT. Como tales, estos dispositivos disponen de una serie de propiedades (estado) resultantes de monitorizar propiedades del mundo real, y también pueden recibir comandos para actuar sobre el entorno. Estarán asociados en la aplicación a un dispositivo real, de manera que, si el dispositivo real deja de funcionar, podrá ser sustituido por otro con similares características permitiendo que la aplicación IoT siga funcionando. A su vez, estos dispositivos virtuales se encontrarán también replicados, de manera que, si ocurre algún fallo donde se esté almacenando el dispositivo virtual, pueda ser sustituido por otra réplica restaurando su estado.

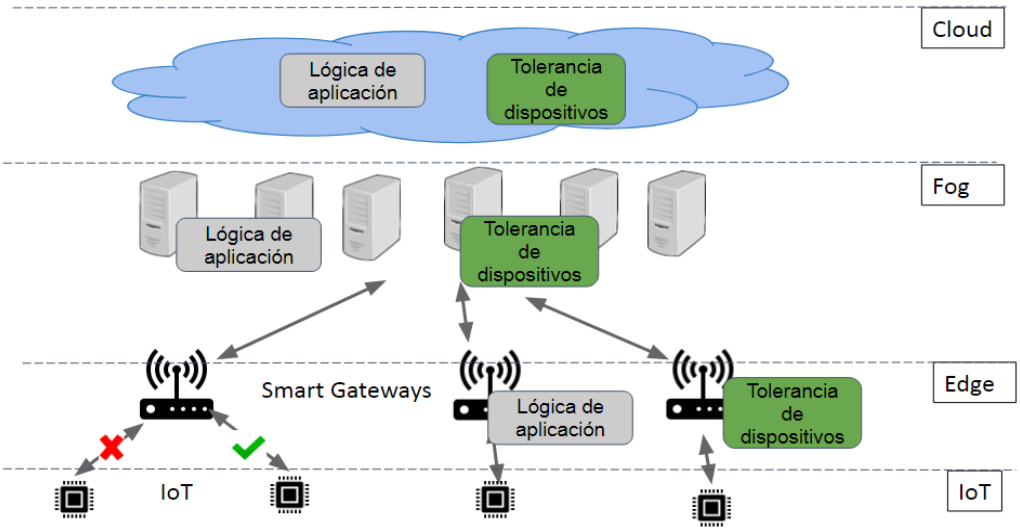


Figura 1.1: Arquitectura general del sistema

La Figura 1.1 muestra una visión general de la arquitectura de este Trabajo. La arquitectura está compuesta por diferentes capas con el fin de reducir la latencia en las comunicaciones con los dispositivos IoT, como son el cloud, fog y edge, al mismo tiempo que se aumenta la fiabilidad del sistema. En la parte inferior izquierda se puede observar cómo un dispositivo IoT se desconecta del sistema debido a un fallo, como puede ser el agotamiento de su batería. A partir de ese momento el sistema detecta el evento e inicia un proceso para alojar la lógica definida en el dispositivo desconectado a uno nuevo

subyacente y listo para abarcar esa lógica. En caso de que no haya ningún dispositivo subyacente disponible, se irán almacenando todas las interacciones con el dispositivo en un sistema de almacenamiento para recuperar su estado una vez que este esté de nuevo operativo. Con el fin de favorecer la portabilidad de los componentes a desarrollar se hará uso de una tecnología de despliegue con contenedores, que permitirá la portabilidad de los componentes a lo largo de la arquitectura y a la vez reducir la latencia y la carga de trabajo en las diferentes capas que lo componen. Esta tecnología también permitirá la tolerancia a fallos a nivel software de la arquitectura.

1.2 Objetivos

El principal objetivo de este Trabajo Fin de Grado es la definición de una arquitectura que permita la creación de aplicaciones de IoT tolerantes a fallos, no únicamente a nivel del software, sino también a nivel de los dispositivos hardware que pueden forman parte de ellas. En la Figura 1.2 se muestra la arquitectura del sistema.

El sistema dispondrá de una web en la que se podrán gestionar los dispositivos Shadow que forman el sistema y consultar el estado de los dispositivos físicos que se vayan a registrar. También permitirá borrar los dispositivos físicos junto con todos sus recursos, o borrar únicamente algún recurso específico de un dispositivo.

Por otra parte, permitirá ver las aplicaciones que han mostrado algún tipo de interés en un recurso mediante una API REST ofrecida por el módulo IoT Shadow Applications o aplicaciones que estén utilizando un recurso. Por último, se podrán ver los tipos de dispositivos IoT que el sistema puede monitorizar y añadir uno nuevo.

Los distintos dispositivos IoT serán registrados en el sistema mediante una API REST ofrecida por el módulo Register que se encarga de realizar esta tarea y también de desplegar de forma dinámica una instancia del módulo IoT Connector (dependiendo del tipo del dispositivo registrado) que automáticamente comenzará una monitorización de cada dispositivo IoT nuevo registrado. Esta monitorización consistirá en escuchar de forma continua eventos mandados por los dispositivos IoT y en cada caso, dependiendo del evento, realizar una acción concreta. Pueden ocurrir una serie de eventos:

- Eventos periódicos informando del estado del IoT
- Eventos de registro de nuevos recursos en un IoT del sistema que provoca la actualización del IoT en la base de datos
- Eventos de fallo de algún recurso o de varios.

Este último evento es el más interesante, ya que, el monitor (connector) informará del fallo al módulo encargado de la restauración poniendo en marcha el protocolo de restauración si alguno de los recursos que han fallado se estaba utilizando, con el objetivo de seguir ofreciendo servicio de una manera ininterrumpida. En caso de que un recurso que ha fallado no estaba siendo solicitado por una aplicación, no se procede al protocolo de restauración.

Por otra parte, se ofrecerá una API REST por el módulo IoT Shadow Applications para las aplicaciones finales, que podrán, en primer lugar, mostrar su interés (comprobar el estado) en algún recurso del sistema perteneciente a un dispositivo Shadow en concreto o a cualquier dispositivo, y, en segundo lugar, podrán pedir al sistema que realice una acción sobre un recurso, por ejemplo, monitorizar la temperatura en un dispositivo. Esta acción se le comunicará directamente mediante Apache Kafka al Monitor que esté monitorizando el recurso deseado y realizará la acción deseada. El Monitor le comunicará el resultado a la aplicación solicitante mediante Apache Kafka.

Por último, para conseguir el objetivo que se persigue, se definirá una API REST de la base de datos, la cual se encargará de realizar todas las consultas que los diferentes módulos necesiten hacer en la base de datos.

El entorno en el que el sistema con todos sus componentes se va a ejecutar es Docker.

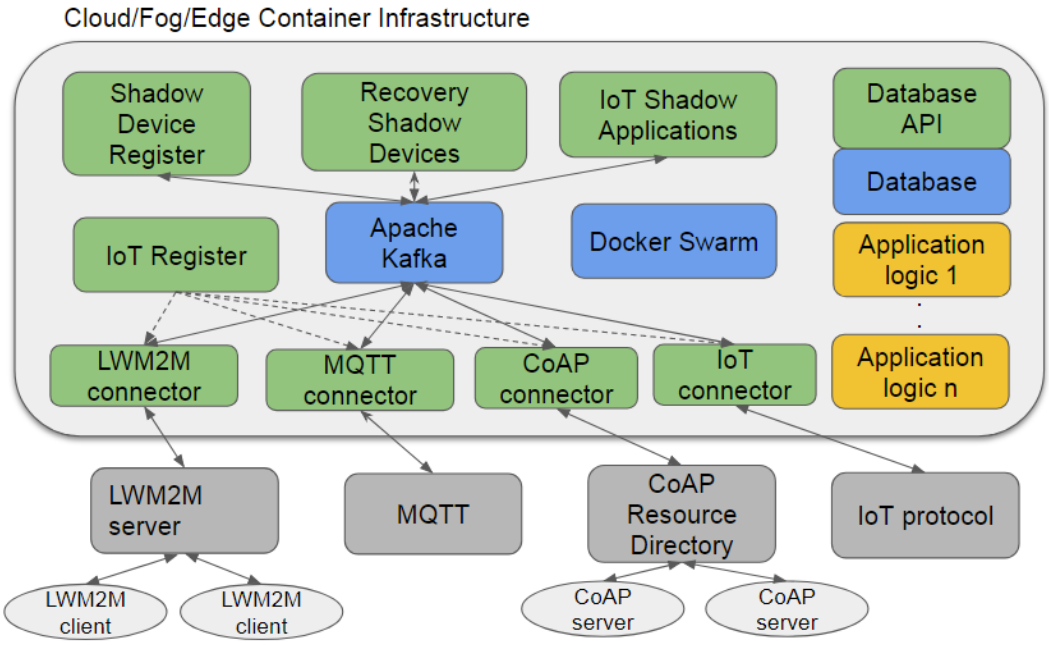


Figura 1.2: Visión general de la arquitectura de tolerancia de la capa intermedia *Fog* (Ver Figura 1.1).

1.3 Descripción de los capítulos

La memoria consta de 5 capítulos sin incluir las referencias y los dos apéndices (Manual de instalación y Manual de Uso).

A continuación, se resume brevemente lo que contiene cada capítulo:

1. **Introducción**
2. **Tecnologías y herramientas utilizadas**

   En este capítulo se hace una breve introducción a las tecnologías utilizadas destacando ligeramente sus características principales.
3. **Estudio de otros trabajos similares**

   En este capítulo se analiza varios trabajos similares destacándose las principales diferencias entre estos y el trabajo presente.
4. **Fases del desarrollo del proyecto**

   La fase de desarrollo es la más sustanciosa y está dividida en 5 partes: Metodologías, Análisis, Especificación, Diseño e Implementación.

   En este capítulo se habla al principio de algunas de las metodologías de desarrollo que existen y se escoge una que se seguirá en las cuatro fases del ciclo de vida del desarrollo de software que corresponden a los apartados Análisis, Especificación, Diseño e Implementación.
5. **Conclusiones**

   En este capítulo se recogen las conclusiones obtenidas tras realizar el proyecto, la experiencia personal y las posibles líneas futuras que el proyecto puede seguir.

2

Tecnologías y herramientas utilizadas

Las tecnologías que se han utilizado para el desarrollo e implementación son varias. A continuación, se listan y se detalla cada una de ellas en particular.

2.1 Docker

Para la parte de replicación se usa la tecnología Contenedores Docker. Docker es una plataforma (se puede ver su logo en la Figura 2.1) para desarrollar, desplegar y ejecutar aplicaciones en contenedores. El uso de contenedores para desplegar aplicaciones se llama *contenerización*. Los contenedores se caracterizan por ser: flexibles, ligeros, intercambiables, portátiles, escalables y apilables.

Un contenedor es una instancia en tiempo de ejecución de una imagen que se ejecuta de forma nativa en Linux o Windows, compartiendo el kernel de la máquina que lo alberga con otros contenedores. Por el contrario que una máquina virtual que ejecuta un sistema operativo por completo accediendo a los recursos de la máquina que lo alberga a través de un hipervisor (ver Figura 2.2).

Figura 2.1: Logo Docker (2019 Docker Inc.)

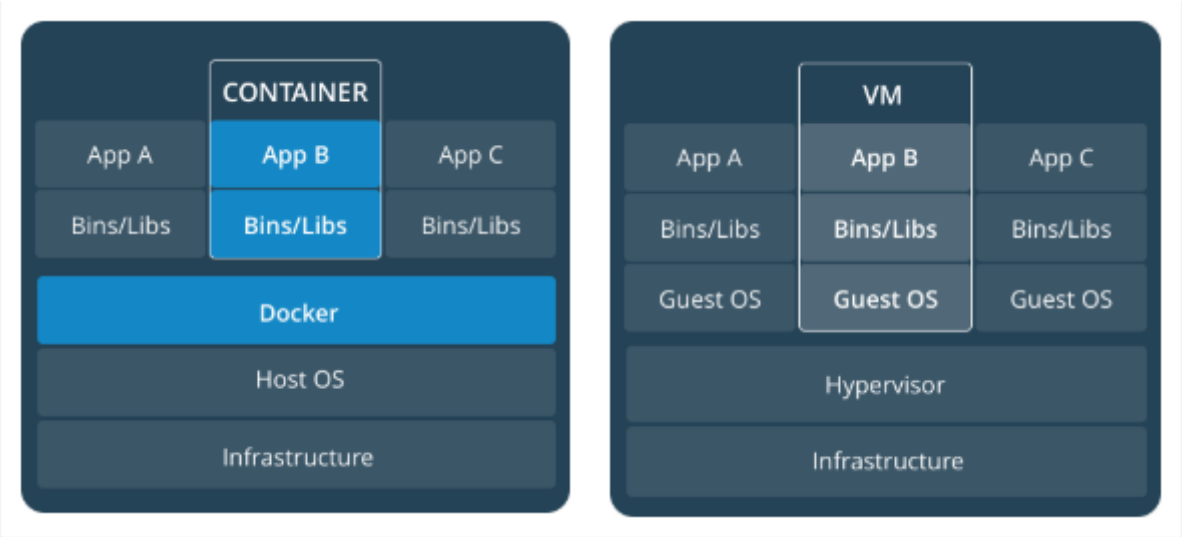


Figura 2.2: Arquitectura Docker vs Arquitectura sin Docker (2019 Docker Inc.)
Recuperado el 21 de marzo de 2019

2.2 Python 3.7

Se trata de un lenguaje interpretado, dinámico de alto nivel, orientado a objetos, interactivo, interpretado, modular, portátil y también extensible en C++ y C (se puede ver su logo en la Figura 2.3). En general es un lenguaje robusto y fácil de programar que facilita el mantenimiento y la depuración. Ha ganado bastante importancia a causa de que Google lo ha convertido en uno de sus lenguajes de programación oficiales debido a su facilidad de uso y una sintaxis poco sobrecargada y de muy alto nivel. Esto hace posible que sea usado por personas de muchas disciplinas. Es un lenguaje muy flexible, lo que conlleva a que sea algo lento afectando, por tanto, a su rendimiento.

Debido a que es un lenguaje amigable especialmente con los principiantes, ha atraído a un gran número de programadores a su amplia comunidad. También dispone de una librería muy amplia, lo que facilita muchas tareas.

Figura 2.3: Logo Python

(Copyright ©2001-2019, Python Software Foundation)

Recuperado el 21 de marzo de 2019

2.3 Django

Django (se puede observar el logo en la Figura 2.4) es un marco de trabajo (Framework) de código abierto. Se ha diseñado con el pensamiento de llevar las aplicaciones desde el concepto hasta su lanzamiento en cuestión de horas, por lo que una de sus características es la *rapidez*. Incluye muchos extras que se usan para llevar a cabo tareas comunes del desarrollo web (por ejemplo: autenticación, administración de contenidos…) de manera inmediata haciendo hincapié también en la seguridad, y previniendo así errores comunes como las inyecciones SQL, etc.

Otra característica muy deseada y que caracteriza a este marco de trabajo es la escalabilidad. Algunos de los sitios más concurridos del planeta utilizan la capacidad de Django para escalar de manera rápida y flexible.

"Django se inventó para cumplir con los plazos de las salas de redacción de forma rápida, al tiempo que satisfacía los requisitos exigentes de los desarrolladores web experimentados."

(©2005-2019 Django Software Fundation y colaboradores individuales)

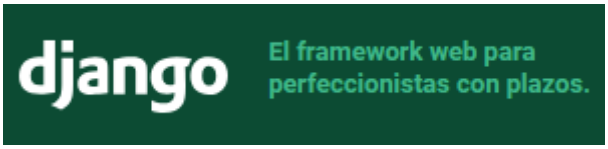


Figura 2.4: Logo de Django

(©2005-2019 Django Software Fundation y colaboradores individuales)

Recuperado el 21 de marzo de 2019

2.4 MongoDB

MongoDB (se puede ver su logo en la Figura 2.5) es una base de datos no relacional de código abierto orientada a documentos (JSON-like) que permite fácilmente organizar, usar y enriquecer los datos. Tiene un modelo de documentos de datos que hace trabajar con los datos de forma intuitiva y ofrece un control preciso sobre estos. Otra característica importante de la base de datos es la alta escalabilidad que ofrece.

Figura 2.5: Logo de MongoDB (© 2019 MongoDB, Inc.)
Recuperado el 28 de marzo de 2019

2.5 Apache Kafka

Apache Kafka es una plataforma de transmisión (se puede ver su logo en la Figura 2.6) que se usa generalmente para dos tipos de aplicaciones:

1. Para creación de flujos de datos en tiempo real con el objetivo de intercambiar datos de forma fiable entre sistemas y aplicaciones.
2. Para creación de aplicaciones de transmisión en tiempo real que transforman o reaccionan a flujos de datos.

Kafka se ejecuta en un clúster en uno o más servidores que pueden abarcar varios centros de datos.

Por otra parte, proporciona 4 APIs principales: Producer API, Consumer API, Streams API, Connector API.

Figura 2.6: Logo de Apache kafka (© 2017 Apache Software Foundation)
Recuperado el 28 de marzo de 2019

2.6 Leshan (OMA Lightweight M2M)

Eclipse Leshan (se puede ver su Logo en la Figura 2.7) es un conjunto de librerías de Java con el objetivo de ayudar a los desarrolladores a desarrollar su propio servidor y cliente Lightweight M2M. Por otra parte, OMA Lightweight M2M es un protocolo de administración de dispositivos IoT.

Figura 2.7: Logo de Leshan (© 2015 The Eclipse Foundation)
Recuperado el 28 de marzo de 2019

En el Apéndice A se muestra el funcionamiento de esta tecnología y cómo ejecutarla en modo local.

2.7 Robot 3T (anteriormente Robomongo)

Robot 3T o Robomongo (se puede ver su logo en la Figura 2.13) es una interfaz gráfica sencilla para establecer una conexión con una base de datos MongoDB. Cabe a destacar que cuenta con una Shell integrada que permite al usuario añadir, borrar, inspeccionar, consultar y editar documentos en la base de datos.

Robot 3T está disponible en cualquier plataforma distribuido como aplicación nativa, que se caracteriza por su rapidez y el escaso consumo de recursos de la máquina (CPU, RAM, …).

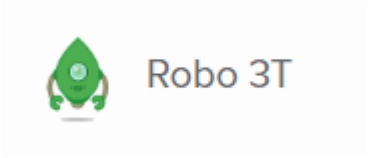
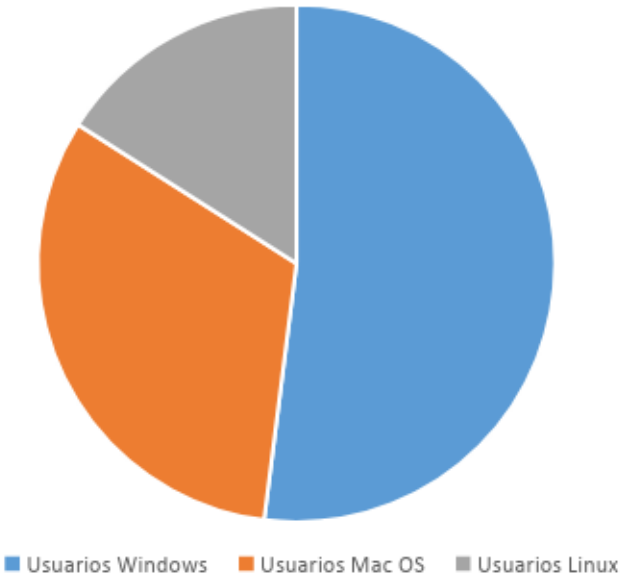


Figura 2.13: Logo de Robot 3T (© 3T Software Labs, 2017)
Recuperado el 29 de marzo de 2019

A fecha de hoy (29 de marzo de 2019) según la información disponible en su página web, la distribución de los usuarios según las plataformas se puede apreciar en la Figura 2.14.

Figura 2.14: Usuarios según plataforma



2.8 Bootstrap

Bootstrap es un marco de trabajo (framework) de código abierto para desarrollar webs que reaccionan (responsive) con HTML, CSS y JS. Es mantenido por un pequeño equipo de desarrolladores en GitHub.

Originalmente fue creado por un diseñador y desarrollador de Twitter, y se ha convertido en una de las herramientas más populares para desarrollo front-end y proyectos de código abierto en el mundo.

Bootstrap se puede decir que consta de 3 archivos principales:

- bootstrap.css
- bootstrap.js
- glyphicons

Adicionalmente Bootstrap necesita jQuery para funcionar, una librería de JavaScript muy popular, que simplifica y asegura la compatibilidad del navegador con JavaScript.

Figura 2.15: Logo de Bootstrap
Recuperado el 29 de abril de 2019

3

Estudio de otros trabajos similares

3.1 Eclipse Hono

Este trabajo tiene un enfoque muy parecido al del trabajo presente. Eclipse Hono ofrece como servicio remoto una API uniforme, para interactuar con un gran número de dispositivos. Estos dispositivos IoT están conectados a la nube de forma uniforme sin tener en cuenta el protocolo de comunicación del dispositivo.

En la Figura 3.1 se puede apreciar el esquema general de la arquitectura que es similar a la del Trabajo presente.

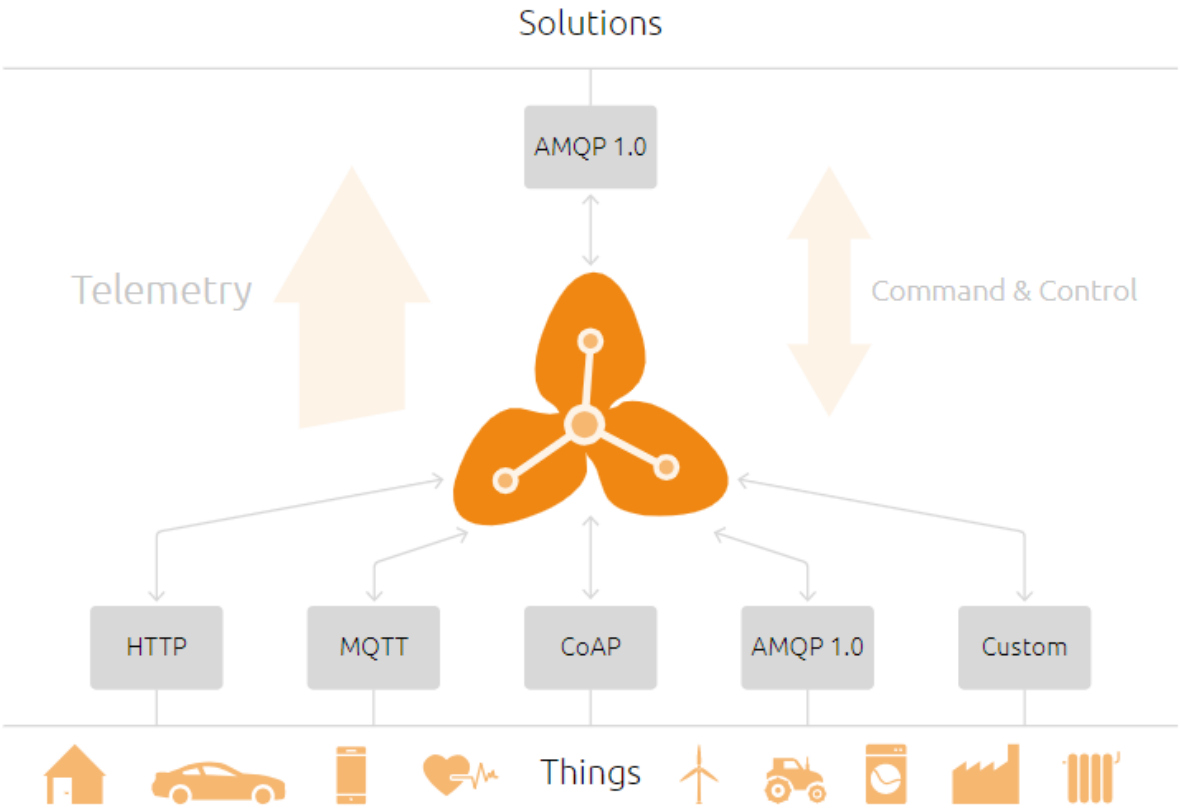


Figura 3.1: Arquitectura general Eclipse Hono (© 2015 The Eclipse Hono Project) Recuperado el 28 de marzo de 2019

Hono se ejecuta sobre Kubernetes, OpenShift y Docker Swarm.

Figura 3.2: Tecnologías base de Eclipse Hono (© 2015 The Eclipse Hono Project) Recuperado el 28 de marzo de 2019

Entre sus objetivos cabe a destacar los siguientes:

- Mensajería general por cola para soluciones IoT.
- Ofrecer estándares de APIs para interactuar con dispositivos IoT.
- Soportar protocolos de comunicación arbitrarios.
- Soportar diferentes subcapas de infraestructura de comunicación.

Por otra parte, entre sus características cabe a destacar las siguientes:

- Escalabilidad
- Manejo de múltiples instancias
- Seguridad basada en dispositivos
- Soporte multiprotocolo
- Configuración, gestión y monitorización fáciles.

En el futuro próximo de este desarrollo se espera o se desea mejorar el rendimiento y la escalabilidad, continuar mejorando las integraciones con OpenShift y EnMasse, y finalmente, ofrecer una API tentadora.

Comparativa entre soluciones

En primer lugar, la diferencia más trivial que se encuentra entre Eclipse Hono y el trabajo presente es la arquitectura. En la Figura 3.3 se describen los componentes de alto nivel que constituyen una instancia de Eclipse Hono y sus relaciones entre ellos.

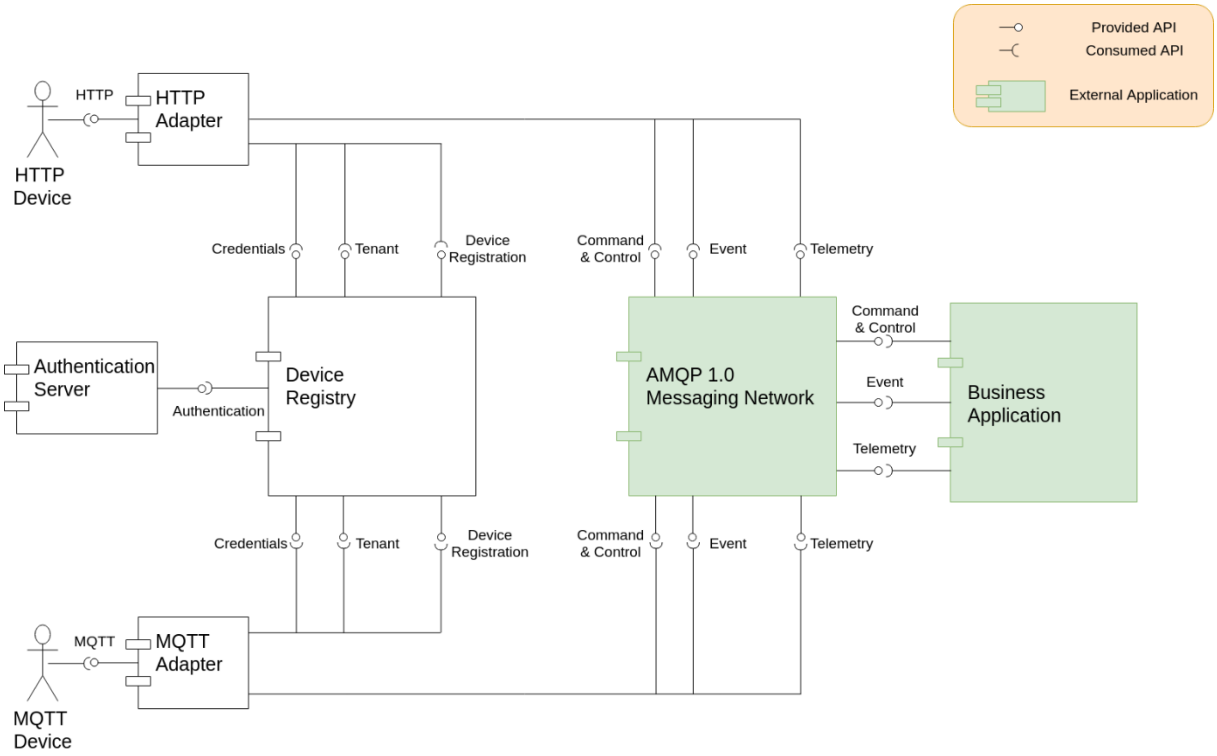


Figura 3.3: Arquitectura Eclipse Hono (© 2015 The Eclipse Hono Project) Recuperado el 2 de mayo de 2019

A simple vista su arquitectura parece más sencilla en comparación con la del proyecto presente (ver Figura 1.2), pero cada módulo está formado por numerosos submódulos internos, como se puede apreciar en la Figura 3.4, que representa el módulo de registro de un dispositivo.

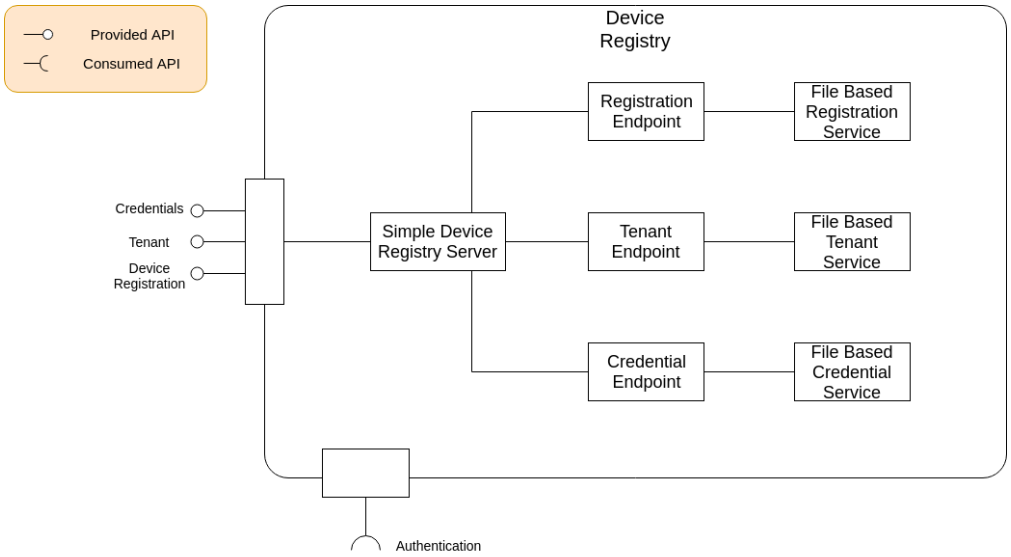


Figura 3.4: Módulo Device Registry (© 2015 The Eclipse Hono Project)
Recuperado el 2 de mayo de 2019

Otra diferencia que se puede apreciar es que, para las comunicaciones internas de los diferentes módulos, ellos han optado por AMQP 1.0 Messaging Network, que es una red de mensajería. No es un componente desarrollado por Hono sino que forma parte de otro proyecto de código abierto que se ha utilizado, ya que, proporcionaba una configuración útil para fines de desarrollo. Un inconveniente de esto es que no asegura que se cumplan los requisitos como la escalabilidad, y tampoco ofrece una solución para este problema.

A diferencia de Hono, se ha optado por utilizar Apache Kafka en vez de Messaging Network debido a que Kafka ofrece un mejor rendimiento y escalabilidad. Además, esta solución se adapta a otros protocolos de IoT que requieren una interacción previa como CoAP, al contrario que Hono.

3.2 Appdaptivity

Appdaptivity es una plataforma para desarrollar aplicaciones con las siguientes características:

- Personalizadas
- Que se adapten a cambios de contexto
- Desacopladas de los dispositivos

Adapta aplicaciones IoT con cambios en la infraestructura subyacente e inyecta dispositivos IoT en la lógica de la aplicación. Aunque Appdaptivity gestiona de forma dinámica aplicaciones IoT para adaptarlas a los cambios de contexto, requiere múltiples dispositivos IoT funcionando para ofrecer tolerancia a fallos. Por ejemplo, en caso de monitorización de temperatura, Appdaptivity debe tener múltiples dispositivos monitorizando la temperatura para tener tolerancia a fallos.

En este trabajo, en caso de mal funcionamiento de un dispositivo, la arquitectura se encarga de obtener el mismo comportamiento de otro dispositivo sin tener múltiples dispositivos haciendo la misma tarea, lo que puede llevar a un incremento del consumo de energía.

3.3 CEFIoT

En CEFIoT se presenta una arquitectura IoT tolerante a fallos para Edge y Cloud. El enfoque es similar al del trabajo actual, ya que, ambos usan Apache Kafka como plataforma de publicación/suscripción y Docker como plataforma de virtualización. CEFIoT también proporciona una plataforma para la gestión de contenedores, Kubernetes, similar a Docker Swarm, sin embargo, CEFIoT solo administra tolerancia a fallos respecto a fallos en nodos a lo largo de la arquitectura. No contempla cuando una fuente de monitorización (un dispositivo IoT) ha interrumpido su servicio para adaptar los demás componentes que la utilizan, como las aplicaciones finales.

En este trabajo, se propone una arquitectura tolerante a fallos que no solo administra la tolerancia a fallos a lo largo de la arquitectura, sino que, también en las fuentes de datos que pertenecen al sistema.

4

Fases del desarrollo del proyecto

Las diferentes fases del desarrollo del proyecto marcarán y guiarán el progreso de este y serán las siguientes:

- **Análisis:** Se realizará un análisis del proyecto (descripción), identificando las bases de este y además se describirán las funcionalidades (requisitos).
- **Especificación:** Se realizará la especificación del proyecto a partir de los requisitos, se constituirá el modelo del dominio y los casos de uso correspondientes.
- **Diseño:** Se definirá la arquitectura del sistema junto con el diseño de los componentes.
- **Implementación:** Se mostrará la implementación por iteraciones a partir del Análisis, Especificación y Diseño del sistema.

4.1 Metodologías

Las metodologías son representaciones abstractas (o simplificadas) de un proceso software. Establecen un conjunto de reglas y estándares que se usan para planear y controlar el proceso de desarrollo de los proyectos software.

Existen diferentes tipos de metodologías debido a la gran diversidad de aplicaciones y productos software. En cada caso se escoge la metodología de desarrollo más apropiada para cada tipo de proyecto que se desarrollará.

Si este paso no se realiza correctamente, es decir, no se escoge una metodología apropiada, esto implicará problemas durante el desarrollo y otras dificultades.

Algunas de las metodologías más usadas y/o conocidas se agrupan en los siguientes 3 grupos:

- Metodologías Clásicas:
  - *Desarrollo en cascada*

    Parte de la especificación hasta la finalización del proyecto y las actividades se van sucediendo a lo largo del desarrollo. Es una metodología adecuada cuando los requisitos están totalmente claros al principio del desarrollo ya que es muy poco flexible.

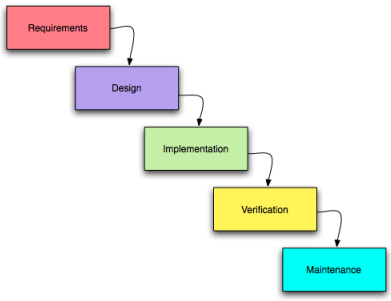


Figura 4.1: Diagrama metodología de desarrollo en cascada.

  - *Desarrollo basado en prototipos*

    Es un desarrollo centrado en el aspecto visible para los clientes para obtener y validar los requisitos del sistema, explorar soluciones de diseño y también para realizar pruebas.

- *Desarrollo incremental*

  La idea es desarrollar un producto inicial, presentarlo al cliente e ir refinándolo hasta llegar al producto final. El propósito es desarrollar software de alta calidad de una manera iterativa. Tiene ventajas como satisfacer una necesidad inmediata del cliente, pero también tiene inconvenientes ya que la arquitectura no es estable, sino que sufre cambios continuos.

- *Proceso Unificado*

  Es un modelo que se divide en ciclos, y cada ciclo se divide en cuatro fases diferentes: Inicio, Elaboración, Construcción y Transición. Cada fase se divide a su vez en iteraciones. Al finalizar un ciclo, se produce una nueva versión del sistema. Este modelo de desarrollo es muy lento y muy sobrecargado.

- Metodologías Ágiles:
  - *Desarrollo iterativo*

    El desarrollo iterativo (o incremental) es un proceso que surge como solución a los problemas del desarrollo en cascada. Es un desarrollo de tipo evolutivo, en el que, las actividades se presentan en forma de espiral. Cada ciclo (o Iteración) de la espiral representa una fase del proceso de desarrollo.

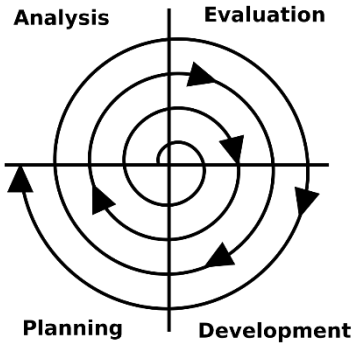


Figura 4.2: Diagrama metodología de desarrollo iterativo.

  - *Programación extrema (XP)*

    Es un proceso ágil muy conocido que se adecúa muy bien a proyectos en los que se prevé que sufrirán cambios a lo largo del tiempo y con equipos de desarrollo pequeños. Se basa en cuatro pilares: Comunicación, Simplicidad, Retroalimentación y Coraje.

  - *Scrum*

    Es un proceso iterativo que define un conjunto de prácticas (backlogs, sprints, mítines, demos) y roles (scrum master, team, product owner). Se adecúa bien a proyectos de tamaño pequeño y mediano y también para proyectos con cierto nivel de incertidumbre.

- Metodologías especializadas:
  - *Desarrollo basado en componentes*

    Es un enfoque basado en la reutilización. Los programas se construyen integrando componentes independientes débilmente acoplados entre ellos.

- *Métodos formales*

  Se realiza una especificación matemática del software y además se detectan errores e inconsistencias a través de modelos matemáticos.

Para el desarrollo del proyecto se decidió seguir una metodología ágil de desarrollo iterativa, ya que, habrá varias iteraciones con resultados intermedios en cada iteración. En un principio se ha estimado que habrá 3 iteraciones.

4.2 Análisis

El análisis es una parte fundamental a la hora de llevar a cabo la realización de un proyecto software. Es muy importante asegurarse de realizar bien este paso, ya que, en esta etapa se identifican las bases del proyecto, se extraen los requisitos del producto Software, que son las principales características del producto y se identifican además los actores principales, que serán los usuarios finales del producto.

Si este paso no se realiza correctamente, es decir, no se identifica bien las necesidades y no se hace una buena captura de requisitos, los errores que esto implica (requisitos incompletos, ambiguos o contradictorios) dificultará las fases posteriores del proceso de desarrollo.

4.2.1 Descripción del sistema

El sistema se compone de varios módulos desplegados todos sobre Contenedores Docker, en un enjambre (Docker Swarm), debido a que Docker ofrece un entorno flexible, ligero, portable y escalable para aplicaciones.
Estos módulos cooperan en conjunto con el fin de aplicar técnicas de tolerancia a fallos para sistemas IoT en una arquitectura cloud-fog-edge.

A continuación, se describe de manera breve cada módulo que compone el sistema:

- **IoT Register**

  Cuando un dispositivo IoT como un Servidor LWM2M se conecta al sistema, envía un mensaje a través de HTTP con autenticación a través de token a este componente para que lo registre en el sistema. A partir de este momento, se inicia el componente IoT Connector correspondiente, que empieza a monitorizar los dispositivos IoT del sistema (e.g, LWM2M Client).

  Este componente, y los demás hacen uso de la información del sistema, almacenan los datos en un sistema de almacenamiento externo como una Base de Datos no relacional (MongoDB), para que se guarde el estado del sistema en caso de fallo.

- **IoT Connector**

  Este componente es encargado de la monitorización de los dispositivos IoT. En caso de que detecte que un dispositivo está desconectado, envía un mensaje a través de Kafka para que se inicie el proceso de recuperación del dispositivo shadow por parte del componente Recovery Shadow Devices.

  También, tiene un consumidor de Kafka que acepta mensajes de otros componentes, como por ejemplo cuando se crea una aplicación y se quiere monitorizar un dispositivo. Este componente, por tanto, le mandaría una petición al Servidor LWM2M para iniciar la monitorización de un recurso y enviaría los datos recibidos por Kafka para su posterior procesamiento en Kafka Streaming.

- **Database**

  Este módulo es simplemente una imagen de Docker, de hecho, es la imagen oficial de MongoDB descargada del repositorio oficial de Docker. Su finalidad es garantizar la persistencia de los datos del sistema.

- **Database API**

  Este componente es una API REST que se encarga de comunicarse con la base de datos. Todos los módulos del sistema se comunican con esta API, cuando quieren realizar una consulta en la base de datos o guardar algún dato, en vez de comunicarse con el módulo Database directamente. Tiene la función de intermediario entre los demás módulos y la base de datos y proveer una capa de abstracción con la base de datos.

  Cabe a destacar también que este módulo ofrece autenticación basada en token (JWT), de tal forma que todas las comunicaciones con esta API van a ser verificadas por token.

- **Shadow Device Register**

  Este componente ofrecerá una aplicación web para que los usuarios finales puedan registrar un nuevo dispositivo Shadow en el sistema. En los dispositivos Shadow se especifican el nombre del dispositivo y el tipo. Esta última propiedad

será utilizada por los dispositivos IoT para establecer una relación con el dispositivo shadow.

- **Recovery Shadow Devices**

  Este componente una vez recibido que un dispositivo de IoT ha sido desconectado, inicia un proceso de recuperación del estado del dispositivo shadow en otro subyacente, o almacena sus transacciones para su posterior recuperación.

- **Apache Kafka**

  Este módulo es clave en el sistema, más concretamente, en lo que a Tolerancia a fallos se refiere, ya que, hace posible las comunicaciones entre los módulos Recovery Shadow Devices, IoT Shadow Applications y los diferentes IoT Connectors que puede haber, de una forma muy rápida, en tiempo real, tolerante a fallo.

- **Docker Swarm**

  Un enjambre es un grupo de máquinas que ejecutan Docker y se unen en un clúster. Las máquinas de un enjambre pueden ser físicas o virtuales (en el caso del proyecto presente será un clúster de Raspberry's Pi3). Habrá un dispositivo que será el administrador del enjambre y los demás serán nodos que se unen al enjambre.

  Sabiendo esto, hay que mencionar que este módulo no se desarrolla, sino que es una característica configurable que Docker ofrece.

- **IoT Shadow Applications**

  Este componente proporcionará una API REST mediante la cual las aplicaciones finales podrán expresar su interés en aspectos de dispositivos Shadow. Una vez recibidos los requisitos de una aplicación, este componente selecciona los dispositivos Shadow correspondientes y empieza a monitorizarlos para que la aplicación correspondiente los procese.

4.2.2 Requisitos funcionales

Los requisitos funcionales describen características, comportamiento o función particular de un sistema software. Un requisito también se puede ver como una condición o capacidad que debe cumplirse o debe tener un sistema o un componente para satisfacer un contrato, especificación u otros documentos impuestos formalmente.

A continuación, se listan los requisitos funcionales del sistema por módulos.

IoT Register

RF.1 – Registrar un nuevo dispositivo físico en el sistema

RF.2 – Desplegar de forma dinámica instancias de un monitor IOT Connector.

Shadow Device Register (Web)

RF.3 – El usuario podrá registrarse en el sistema desde la web con email y contraseña.

RF.4 – El usuario podrá identificarse en el sistema desde la web con email y contraseña.

RF.5 – El sistema asociará un Token de identificación de usuario con validez.

RF.6 – El usuario podrá cerrar sesión en cualquier momento.

RF.7 – El sistema mostrará al usuario los dispositivos Shadow registrados por él.

RF.8 – El sistema mostrará al usuario los tipos de dispositivos físicos disponibles que hay registrados en el sistema.

RF.9 – El usuario podrá Crear, Editar y Borrar dispositivos Shadow.

RF.10 – El usuario podrá ver los tokens asociados a un dispositivo Shadow.

RF.11 – El usuario podrá crear un Token que representará un dispositivo físico asociado al dispositivo Shadow.

RF.12 – El usuario podrá revocar desde la interfaz web en cualquier momento un Token de un dispositivo físico asociado.

RF.13 – El usuario podrá agregar desde la interfaz web un nuevo tipo de dispositivo físico.

RF.14 – El usuario podrá ver los dispositivos físicos de un Dispositivo Shadow registrados en el sistema e información básica sobre ellos: Nombre, Última vez conectado, Token y Estado de uso.

RF.15 – El usuario podrá ver los recursos asociados a un dispositivo físico e información básica sobre ellos: Código, Tipo de recurso y Estado de uso.

RF.16 – El usuario podrá ver todos los recursos asociados a un dispositivo Shadow e información básica sobre ellos: Código, Tipo de recurso y Estado de uso.

RF.17 – El usuario podrá borrar un dispositivo físico junto con todos sus recursos.

RF.18 – El usuario podrá borrar un recurso de un dispositivo físico.

IoT Connector

RF.19 – Identificar los Endpoints de un dispositivo físico registrado y almacenar sus datos en la Base de Datos.

RF.20 – Identificar los recursos de cada Endpoint y almacenar los datos en la Base de Datos.

RF.21 – Recibir eventos de un dispositivo en tiempo real**.**

RF.22 – Almacenar en la base de datos los datos de los eventos recibidos asociados a un dispositivo físico.

RF.23 – Notificar mediante Kafka la desconexión y/o cambio de un dispositivo.

RF.24 – Leer mensajes desde un tópico Kafka.

RF.25 – Operación READ para leer el valor de un recurso.

RF.26 – Operación EXECUTE para realizar una acción sobre un recurso.

RF.27 – Operación WRITE para escribir el valor de un recurso.

RF.28 – Operación OBSERVE para monitorizar/observar el valor de un recurso.

Database API

RF.29 – Verificar en cada consulta la validez del token.

RF.30 – Guardar información de un dispositivo real en la base de datos.

RF.31 – Consultar en la base de datos un dispositivo físico por su id.

RF.32 – Borrar un dispositivo físico de la base de datos por su id.

RF.33 – Consultar el estado de uso de un dispositivo físico por su id.

RF.34 – Guardar información de un endpoint en la base de datos.

RF.35 – Consultar en la base de datos un endpoint por su id.

RF.36 – Actualizar un endpoint en la base de datos por su id.

RF.37 – Guardar información de un recurso en la base de datos.

RF.38 – Consultar en la base de datos un recurso por su id.

RF.39 – Actualizar un recurso en la base de datos por su id.

RF.40 – Consultar en la base de datos los recursos de un dispositivo Shadow por el id de este último.

RF.41 – Consultar en la base de datos los recursos de un dispositivo (IoT Connector) por el id de este último.

RF.42 – Consultar el estado de uso de un recurso por su id.

RF.43 – Guardar el tipo de dispositivo físico.

RF.44 – Consultar el tipo de dispositivo físico.

RF.45 – Consultar en la base de datos todos los tipos de dispositivos físicos.

RF.46 – Generar un Token y guardar en la base de datos.

RF.47 – Validar un Token.

RF.48 – Revocar un Token (actualizar su estado en la base de datos).

RF.49 – Consultar en la base de datos un Token por su id.

RF.50 – Consultar en la base de datos un Token por el id del usuario al que pertenece.

RF.51 – Consultar en la base de datos los Token por el id del dispositivo Shadow al que pertenece.

RF.52 – Crear un dispositivo Shadow.

RF.53 – Actualizar un dispositivo Shadow.

RF.54 – Borrar un dispositivo Shadow.

RF.55 – Consultar en la base de datos los dispositivos Shadow que pertenecen a un usuario por el id de este último.

RF.56 – Consultar en la base de datos un dispositivo Shadow por su id.

RF.57 – Consultar en la base de datos los Token de un dispositivo Shadow.

RF.58 – Consultar en la base de datos los dispositivos físicos de un dispositivo Shadow.

RF.59 – Guardar información de registro de un usuario en la base de datos.

RF.60 – Actualizar información de registro de un usuario de la base de datos.

RF.61 – Verificar los datos de autentificación un usuario.

RF.62 – Buscar si existe una lógica exacta ya creada en un dispositivo Shadow concreto.

RF.63 – Buscar si existe una lógica exacta ya creada en cualquier dispositivo Shadow.

RF.64 – Registrar o Actualizar una aplicación dado un nombre.

RF.65 – Consultar los datos de una aplicación por su id.

RF.66 – Consultar todas las aplicaciones que hay en la base de datos.

RF.67 – Borrar una aplicación por su id.

RF.68 – Actualizar un dispositivo físico de la base de datos por su id.

RF.69 – Buscar un recurso similar en la base de datos dado un tipo de recurso.

RF.70 – Buscar un recurso similar en la base de datos dado un tipo de recurso y especificando un dispositivo Shadow en concreto mediante id.

RF.71 – Guardar los datos del uso de un recurso en la base de datos.

RF.72 – Borrar los datos del uso de un recurso en la base de datos por el id.

RF.73 – Actualizar los datos del uso de un recurso en la base de datos por el id.

RF.74 – Consultar los datos de uso de un recurso de un Endpoint en concreto de un dispositivo Shadow en concreto dados sus identificadores.

Recovery Shadow Devices

RF.75 – Consultar si un recurso está siendo utilizado por una aplicación final.

RF.76 – Recibir mensajes de eventos (eliminación / desconexión de recurso / registro) en endpoints mediante Kafka.

RF.77 – Buscar otro endpoint en el mismo dispositivo Shadow que tenga la misma lógica que el dispositivo caído / desconectado.

RF.78 – Notificar mediante Kafka a las aplicaciones que estén usando un dispositivo Shadow.

RF.79 – Mandar un mensaje a través de Kafka a un IoT Connector para que haga la lógica que se requiere.

IoT Shadow Application

RF.80 – Permitir a una aplicación mostrar interés en un recurso de un dispositivo Shadow en concreto

RF.81 – Permitir a una aplicación mostrar interés en un recurso sin importar de qué dispositivo Shadow proviene

RF.82 – Pedir un recurso de un dispositivo Shadow en concreto.

RF.83 – Pedir un recurso sin importar de qué dispositivo Shadow proviene.

RF.84 – Mandar mensaje mediante Kafka a un IoT Connector para crear la lógica deseada.

RF.85 – Crear un tópico Kafka (o devolver uno ya existente) entre la aplicación final y el IoT Connector para que haya comunicación entre estos dos.

Database

No es un módulo desarrollado, sino que es un proyecto de terceros que se ha integrado en el sistema, por tanto, no se procede a su análisis.

Apache Kafka

No es un módulo desarrollado, sino que es un proyecto de terceros que se ha integrado en el sistema, por tanto, no se procede a su análisis.

Docker Swarm

No es un módulo desarrollado, sino que es un proyecto de terceros que se ha integrado en el sistema, por tanto, no se procede a su análisis.

4.2.3 Requisitos no funcionales

Los requisitos no funcionales son restricciones sobre el sistema que no presentan una funcionalidad o característica. Entre los ejemplos de requisitos no funcionales se pueden encontrar los referidos a atributos como la eficiencia, seguridad, usabilidad del sistema, etc.

A continuación, se listan los requisitos no funcionales del sistema:

RnF.1 – La interfaz de la página web debe ser amigable para el usuario.

RnF.2 – Los módulos IoT Register, Database e IoT web se comunicarán mediante REST.

RnF.3 – Para la comunicación interna se usará Apache Kafka.

RnF.4 – Se utilizará una base de datos no relacional basada en documentos JSON-like.

RnF.5 – Todos los módulos irán desplegados en contenedores Docker sobre Docker Swarm.

RnF.6 – La creación de tópicos Kafka deberá ser dinámica.

RnF.7 – Los tokens no se borran de la base de datos, sino que solamente se revocan.

RnF.8 – El formato de los mensajes en las comunicaciones será JSON.

RnF.9 – La arquitectura del módulo IoT Monitor será transparente al tipo de dispositivo registrado.

RnF.10 – Las consultas a la base de datos estarán protegidas por Token.

RnF.11 – Los tokens de usuario tendrán una validez de 30 minutos.

RnF.12 – Los tokens de dispositivos físicos no tendrán límite de validez a no ser que se revoquen manualmente desde la web.

RnF.13 – Un usuario obtendrá su token para navegar en la aplicación web en el momento en el que se registra o se identifica con unos datos válidos.

RnF.14 – La identificación y el registro serán mediante usuario (email) y contraseña.

RnF.15 – Se utilizarán imágenes oficiales de Docker para la base de datos, zookeeper y Kafka.

4.3 Especificación

La especificación de un proyecto software contiene las descripciones de cómo se usará el software desde la perspectiva del usuario. Una buena especificación es de mucha ayuda, incluso esencial en un proyecto software. Esta debe ser simple pero completa en cuanto a las características técnicas que debe cumplir el sistema y en cuanto a la funcionalidad y propósito del software.

En este capítulo se realizará la especificación del sistema mediante modelo de dominio, casos de uso y descripción de los casos de uso a partir del análisis del sistema que se ha realizado en el capítulo **Análisis.**

4.3.1 Modelo de dominio

Un modelo de dominio es una abstracción de un sistema real. Describe entidades y relaciones entre estas. El modelo de dominio es derivado del análisis y comprensión de requisitos a nivel de sistema y proporciona una base sólida a los participantes involucrados en el desarrollo del proyecto para comprender mejor el sistema.

Puede ser de mucha ayuda durante el desarrollo del sistema, ya que, ayuda a controlar la complejidad desarrollo y resuelve ambigüedades en los requisitos y en el diseño también.

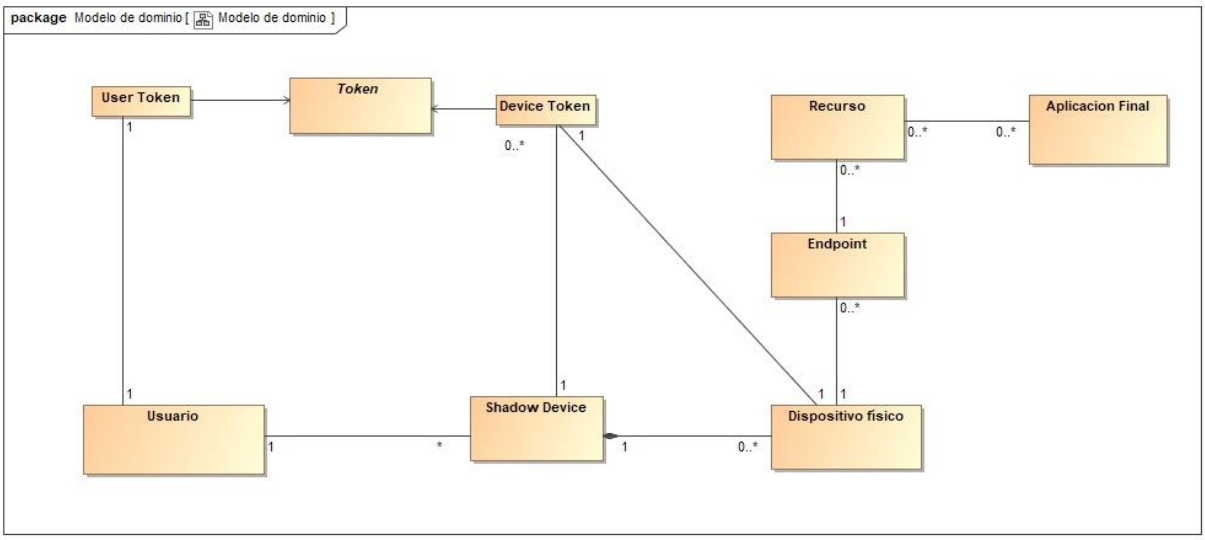


Figura 4.3: Modelo del dominio.

Casos de uso

Los casos de uso son una secuencia de acciones o pasos que típicamente definen la interacción entre un actor y un sistema. El actor puede ser un usuario humano o también podría ser un sistema externo.

Los casos de uso son beneficiosos para el desarrollo del proyecto ya que ayuda a capturar los requisitos funcionales de un sistema, son fáciles de entender por los usuarios y se puede hacer un fácil seguimiento de ellos entre otras ventajas.

A continuación, se enumeran una serie de casos de uso:

- Caso de uso Registro dispositivo físico

Figura 4.4: Caso de uso Registrar un dispositivo físico.

- Caso de uso Web registro

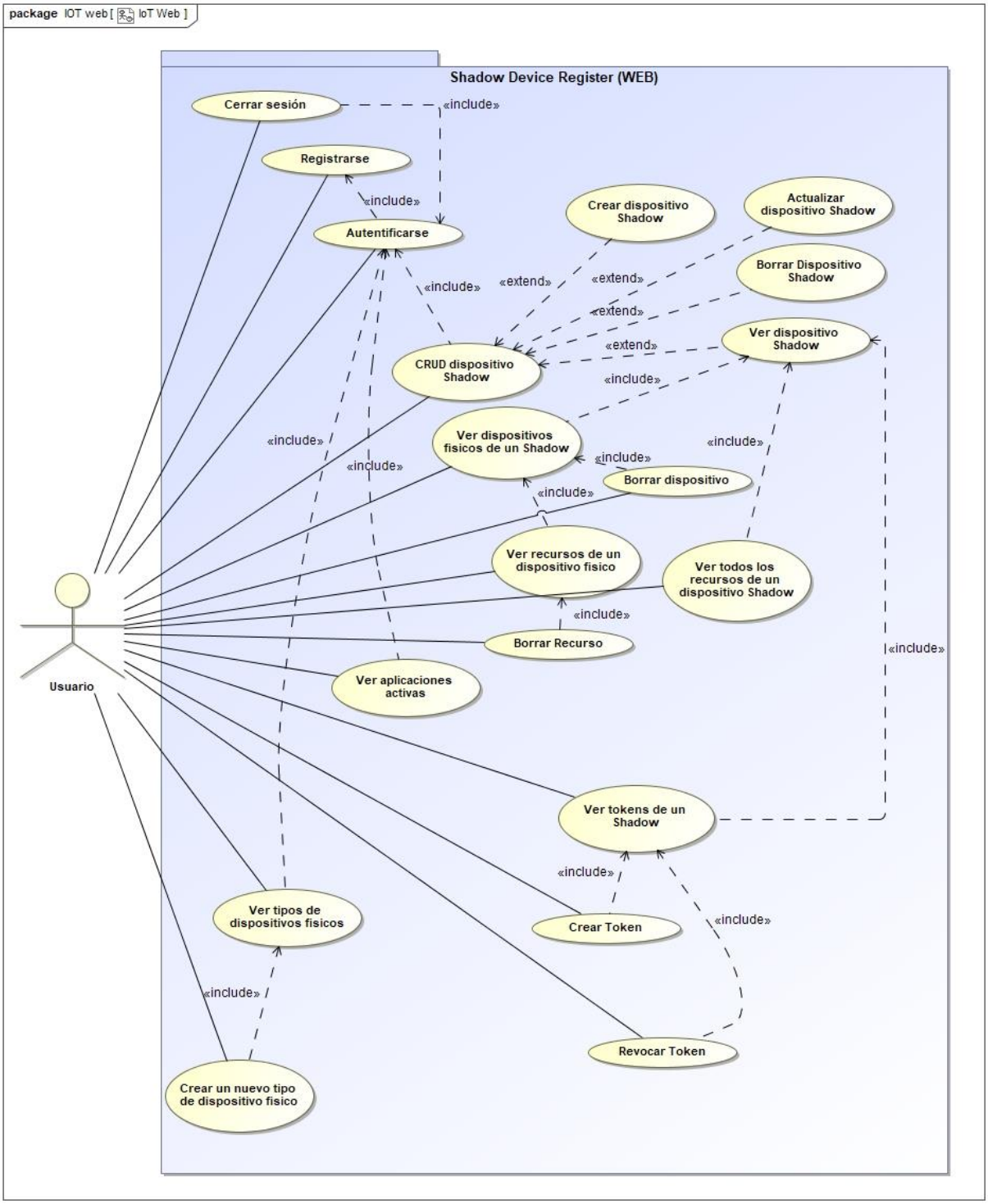


Figura 4.5: Caso de uso Shadow Device Register.

- Caso de uso aplicación final

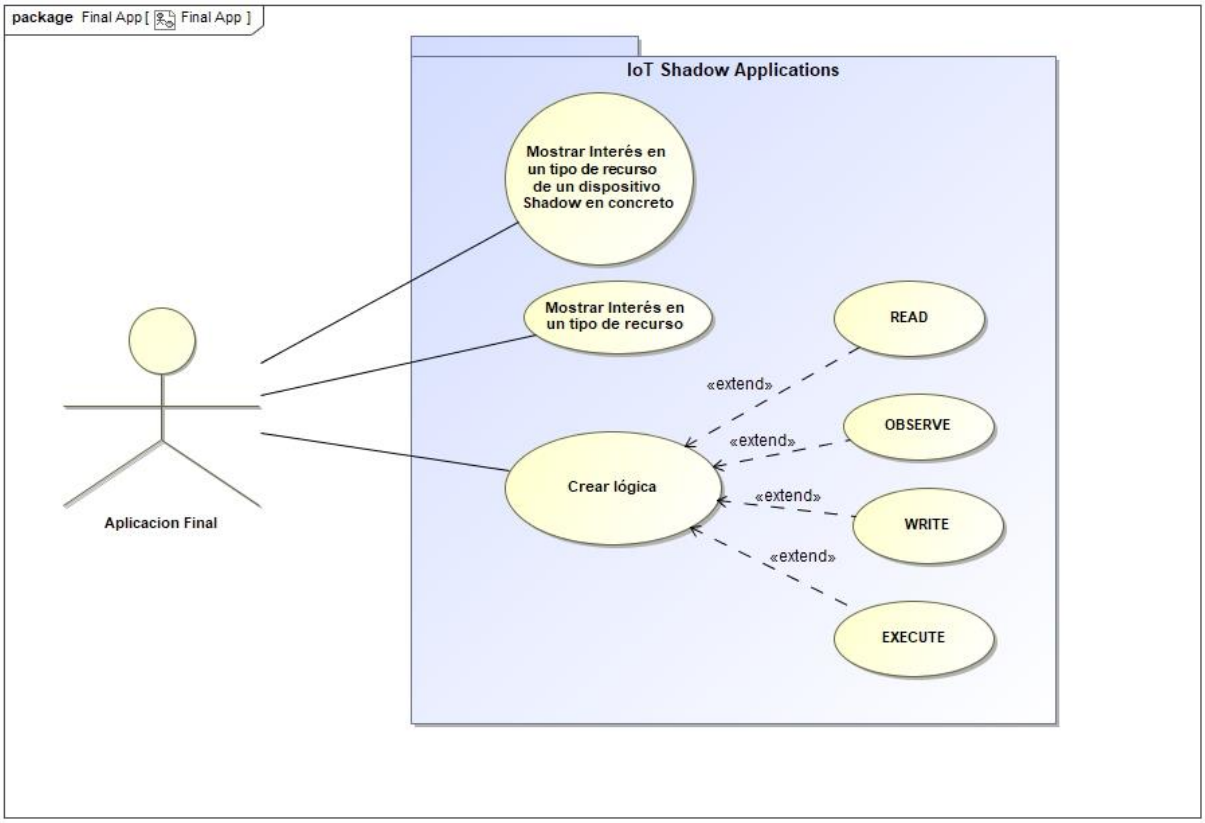


Figura 4.6: Caso de uso Aplicación final.

Descripción de los casos de uso

En este apartado se describen textualmente los casos de uso más representativos del sistema.

Título	Registro
Descripción	Un usuario que se registra en la Web
Pre-condición	Ninguna
Post-condición	Estar identificado con el usuario y contraseña introducidos
Escenario principal	
1. El usuario introduce email y contraseña 2. El sistema verifica que no hay ningún usuario registrado con ese email y genera un token para el nuevo usuario y se lo devuelve 3. El sistema redirige al usuario a la pantalla principal de su cuenta	
Escenario alternativo	
1.a El usuario cancela el registro 1. El sistema devuelve al usuario a la pantalla Identificarse o Registrarse	
2.a Ya existe un usuario registrado con ese email 1. El sistema devuelve una pantalla de error indicando que intente identificarse en vez de registrarse 2. El usuario pincha en el enlace volver 3. El sistema devuelve la pantalla para identificarse	

Título	Identificación
Descripción	Un usuario que se identifica en la Web
Pre-condición	Estar registrado de antemano
Post-condición	Estar identificado con el usuario y contraseña introducidos
Escenario principal	
1. El usuario introduce email y contraseña 2. El sistema valida los datos y genera un token para el nuevo usuario y se lo devuelve 3. El sistema redirige al usuario a la pantalla principal de su cuenta	
Escenario alternativo	
1.a El usuario pulsa cancelar 1. El sistema devuelve al usuario a la pantalla Identificarse o Registrarse	
2.a El email o la contraseña no coincide con los de ningún usuario registrado 1. El sistema devuelve una pantalla de error 2. El usuario pincha en el enlace volver 3. El sistema regresa al paso 1	

Título	Cerrar sesión
Descripción	Un usuario que cierra sesión en la Web
Pre-condición	Estar registrado de antemano Estar identificado de antemano
Post-condición	Estar en la pantalla de inicio de sesión o registro
Escenario principal	
1. El usuario pulsa el botón cerrar sesión 2. El sistema revoca el token del usuario 3. El sistema redirige al usuario a la pantalla de inicio de sesión o registro	
Escenario alternativo	

Título	Creación de un dispositivo Shadow
Descripción	Un usuario que crea un dispositivo Shadow
Pre-condición	Estar registrado en la web Haber iniciado sesión en la web Disponer de un token de usuario válido
Post-condición	Ver en pantalla el nuevo dispositivo Shadow creado
Escenario principal	
1. El usuario pulsa el botón para crear dispositivo Shadow 2. El sistema devuelve la vista para introducir los datos del dispositivo 3. El usuario introduce los datos y pulsa el botón crear 4. El sistema pide una confirmación de la acción 5. El usuario confirma la acción 6. El sistema crea el nuevo dispositivo Shadow y devuelve la página principal del usuario	
Escenario alternativo	
3.a El usuario pulsa el botón cancelar 1. El sistema devuelve la página principal del usuario sin hacer ninguna modificación	
5.a El usuario no confirma la acción 1. El sistema vuelve al paso 3	
El token del usuario puede expirar en cualquier momento 1. El sistema devuelve una pantalla de error 2. El usuario pulsa el botón volver 3. El sistema vuelve a la pantalla de inicio de sesión 4. El usuario inicia sesión de nuevo	

Título	Borrar un dispositivo Shadow
Descripción	Un usuario que borra un dispositivo Shadow
Pre-condición	Estar registrado en la web Haber iniciado sesión en la web Disponer de un token de usuario válido Tener al menos un dispositivo Shadow creado
Post-condición	Ver en pantalla que el dispositivo Shadow borrado ya no está
Escenario principal	
1. El usuario pulsa el botón borrar 2. El sistema pide una confirmación de la acción 3. El usuario confirma la acción 4. El sistema devuelve la vista actualizada	
Escenario alternativo	
3.a El usuario no confirma la acción 1. El sistema no hace nada y se queda como estaba	
El token del usuario puede expirar en cualquier momento 1. El sistema devuelve una pantalla de error 2. El usuario pulsa el botón volver 3. El sistema vuelve a la pantalla de inicio de sesión 4. El usuario inicia sesión de nuevo	

Título	Editar un dispositivo Shadow
Descripción	Un usuario que edita un dispositivo Shadow
Pre-condición	Estar registrado en la web Haber iniciado sesión en la web Disponer de un token de usuario válido Tener al menos un dispositivo Shadow creado
Post-condición	Ver en pantalla el dispositivo Shadow con la información actualizada

Escenario principal

1. El usuario pulsa el botón editar
2. El sistema devuelve la pantalla para editar el dispositivo
3. El usuario introduce la nueva información y pulsa el botón actualiza
4. El sistema pide una confirmación de la acción
5. El usuario confirma la acción
6. El sistema redirige al usuario a la pantalla principal de su cuenta

Escenario alternativo

3.a El usuario pulsa el botón cancelar

1. El sistema devuelve la página principal del usuario sin hacer ninguna modificación

4.a El usuario no confirma la acción

1. El sistema vuelve al paso 3

El token del usuario puede expirar en cualquier momento

1. El sistema devuelve una pantalla de error
2. El usuario pulsa el botón volver
3. El sistema vuelve a la pantalla de inicio de sesión
4. El usuario inicia sesión de nuevo

Título	Ver tokens de un dispositivo Shadow
Descripción	Un usuario que ve los tokens de un dispositivo Shadow
Pre-condición	Estar registrado en la web Haber iniciado sesión en la web Disponer de un token de usuario válido Tener al menos un dispositivo Shadow creado
Post-condición	Ver en pantalla los tokens del dispositivo Shadow
Escenario principal	
1. El usuario pulsa el botón ver Tokens 2. El sistema devuelve la pantalla de visualización de tokens	
Escenario alternativo	
El token del usuario puede expirar en cualquier momento 1. El sistema devuelve una pantalla de error 2. El usuario pulsa el botón volver 3. El sistema vuelve a la pantalla de inicio de sesión 4. El usuario inicia sesión de nuevo	

Título	Generar token
Descripción	Un usuario que genera un token
Pre-condición	Estar registrado en la web Haber iniciado sesión en la web Disponer de un token de usuario válido Tener al menos un dispositivo Shadow creado Encontrarse en la vista de los tokens
Post-condición	Ver en pantalla un nuevo token creado
Escenario principal	
1. El usuario pulsa el botón crear token 2. El sistema pide una confirmación de la acción 3. El usuario confirma la acción 4. El sistema genera el token y actualiza la vista	
Escenario alternativo	
3.a El usuario no confirma la acción 1. El sistema no hace nada y se queda tal como estaba	
El token del usuario puede expirar en cualquier momento 1. El sistema devuelve una pantalla de error 2. El usuario pulsa el botón volver 3. El sistema vuelve a la pantalla de inicio de sesión 4. El usuario inicia sesión de nuevo	

Título	Ver recursos de un dispositivo Shadow
Descripción	Un usuario que ve todos los recursos de un dispositivo Shadow
Pre-condición	Estar registrado en la web Haber iniciado sesión en la web Disponer de un token de usuario válido Tener al menos un dispositivo Shadow creado
Post-condición	Ver en pantalla todos los recursos asociados al dispositivo Shadow
Escenario principal	
1. El usuario pulsa el botón Recursos 2. El sistema devuelve la pantalla de visualización de recursos	
Escenario alternativo	
El token del usuario puede expirar en cualquier momento 1. El sistema devuelve una pantalla de error 2. El usuario pulsa el botón volver 3. El sistema vuelve a la pantalla de inicio de sesión 4. El usuario inicia sesión de nuevo	

Título	Ver dispositivos físicos de un dispositivo Shadow
Descripción	Un usuario que ve los dispositivos físicos registrados en un dispositivo Shadow
Pre-condición	Estar registrado en la web Haber iniciado sesión en la web Disponer de un token de usuario válido Tener al menos un dispositivo Shadow creado
Post-condición	Ver en pantalla los dispositivos físicos asociados al dispositivo Shadow
Escenario principal	
1. El usuario pulsa el botón dispositivos 2. El sistema devuelve la pantalla de visualización de dispositivos físicos	
Escenario alternativo	
El token del usuario puede expirar en cualquier momento 1. El sistema devuelve una pantalla de error 2. El usuario pulsa el botón volver 3. El sistema vuelve a la pantalla de inicio de sesión 4. El usuario inicia sesión de nuevo	

Título	Borrar dispositivo físico
Descripción	Un usuario que borra un dispositivo físico
Pre-condición	Estar registrado en la web Haber iniciado sesión en la web Disponer de un token de usuario válido Tener al menos un dispositivo Shadow creado Encontrarse en la vista de dispositivos físicos de un dispositivo Shadow Tener al menos un dispositivo físico asociado al dispositivo Shadow
Post-condición	Ver en pantalla que el dispositivo físico ya no está
Escenario principal	
1. El usuario pulsa el botón borrar 2. El sistema pide una confirmación de la acción 3. El usuario confirma la acción 4. El sistema devuelve la vista actualizada	
Escenario alternativo	
3.a El usuario no confirma la acción 1. El sistema no hace nada y se queda como estaba	
El token del usuario puede expirar en cualquier momento 1. El sistema devuelve una pantalla de error 2. El usuario pulsa el botón volver 3. El sistema vuelve a la pantalla de inicio de sesión 4. El usuario inicia sesión de nuevo	

Título	Ver recursos de un dispositivo físico
Descripción	Un usuario que consulta los recursos de un dispositivo físico
Pre-condición	Estar registrado en la web Haber iniciado sesión en la web Disponer de un token de usuario válido Tener al menos un dispositivo Shadow creado Encontrarse en la vista de los dispositivos físicos de un Shadow Tener al menos un dispositivo físico asociado al dispositivo Shadow
Post-condición	Ver en pantalla los recursos asociados a un dispositivo físico
Escenario principal	
1. El usuario pulsa el botón Recursos 2. El sistema devuelve la pantalla de visualización de recursos	
Escenario alternativo	
El token del usuario puede expirar en cualquier momento 1. El sistema devuelve una pantalla de error 2. El usuario pulsa el botón volver 3. El sistema vuelve a la pantalla de inicio de sesión 4. El usuario inicia sesión de nuevo	

Título	Borrar Recurso
Descripción	Un usuario que borra un recurso de un dispositivo físico
Pre-condición	Estar registrado en la web Haber iniciado sesión en la web Disponer de un token de usuario válido Tener al menos un dispositivo Shadow creado Tener al menos un dispositivo físico asociado al dispositivo Shadow Encontrarse en la vista de los recursos de un dispositivo físico Tener al menos un recurso
Post-condición	Ver en pantalla que el dispositivo Recurso borrado ya no está
Escenario principal	
1. El usuario pulsa el botón borrar 2. El sistema pide una confirmación de la acción 3. El usuario confirma la acción 4. El sistema devuelve la vista actualizada	
Escenario alternativo	
3.a El usuario no confirma la acción 1. El sistema no hace nada y se queda como estaba	
El token del usuario puede expirar en cualquier momento 1. El sistema devuelve una pantalla de error 2. El usuario pulsa el botón volver 3. El sistema vuelve a la pantalla de inicio de sesión 4. El usuario inicia sesión de nuevo	

Título	Ver tipos de dispositivos IoT
Descripción	Un usuario que ve los tipos de dispositivos físicos que hay en el sistema
Pre-condición	Estar registrado en la web Haber iniciado sesión en la web Disponer de un token de usuario válido
Post-condición	Ver en pantalla los tipos de dispositivos físicos que se puede registrar en el sistema
Escenario principal	
1. El usuario pulsa el botón Connectors 2. El sistema devuelve la pantalla de visualización de tipos de dispositivos IoT	
Escenario alternativo	
El token del usuario puede expirar en cualquier momento 1. El sistema devuelve una pantalla de error 2. El usuario pulsa el botón volver 3. El sistema vuelve a la pantalla de inicio de sesión 4. El usuario inicia sesión de nuevo	

Título	Añadir connector
Descripción	Un usuario que añade un nuevo tipo de dispositivo IoT al sistema
Pre-condición	Estar registrado en la web Haber iniciado sesión en la web Disponer de un token de usuario válido Encontrarse en la vista de los tipos de dispositivo
Post-condición	Ver en pantalla el nuevo tipo de dispositivo
Escenario principal	
1. El usuario pulsa el botón añadir 2. El sistema devuelve un formulario 3. El usuario introduce los datos y pulsa el botón crear 4. El sistema pide confirmación de la acción 5. El usuario confirma la acción 6. El sistema devuelve la vista de los tipos de dispositivo actualizada	
Escenario alternativo	
3.a El usuario pulsa el botón cancelar 1. El sistema devuelve la vista de los dispositivos físicos sin hacer ningún cambio	
5.a El usuario no confirma la acción 1. El sistema no hace nada y se queda en el paso 3	
El token del usuario puede expirar en cualquier momento 1. El sistema devuelve una pantalla de error 2. El usuario pulsa el botón volver 3. El sistema vuelve a la pantalla de inicio de sesión 4. El usuario inicia sesión de nuevo	

Título	Ver aplicaciones
Descripción	Un usuario que ve las aplicaciones que consumen algún recurso o que han mostrado interés en algún recurso
Pre-condición	Estar registrado en la web Haber iniciado sesión en la web Disponer de un token de usuario válido
Post-condición	Ver en pantalla todas las aplicaciones que estén consumiendo algún recurso
Escenario principal	
1. El usuario pulsa el botón Aplicaciones 2. El sistema devuelve la pantalla de visualización de aplicaciones que consumen recursos	
Escenario alternativo	
El token del usuario puede expirar en cualquier momento 1. El sistema devuelve una pantalla de error 2. El usuario pulsa el botón volver 3. El sistema vuelve a la pantalla de inicio de sesión 4. El usuario inicia sesión de nuevo	

Título	Aplicación crea lógica
Descripción	Una aplicación final que solicita que el sistema haga una acción
Pre-condición	Haber mostrado interés en el recurso Haber recibido una respuesta afirmativa en cuanto a la disponibilidad del recurso
Post-condición	El sistema realiza la acción deseada
Escenario principal	
1. La aplicación hace una petición para realizar la lógica deseada sobre un recurso 2. El sistema realiza la lógica requerida y le manda los datos a la aplicación	
Escenario alternativo	
1.a La aplicación pide leer el valor de un recurso 1. El sistema lee el valor del recurso y se lo devuelve	
1.b La aplicación pide observar el valor de un recurso 1. El sistema manda lecturas del valor del recurso periódicamente	
1.c La aplicación pide ejecutar una acción sobre un recurso 1. El sistema ejecuta la acción	
1.d La aplicación pide escribir un valor a un recurso 1. El sistema hace la escritura	
El token del usuario puede expirar en cualquier momento 1. El sistema devuelve una pantalla de error 2. El usuario pulsa el botón volver 3. El sistema vuelve a la pantalla de inicio de sesión 4. El usuario inicia sesión de nuevo	

Título	Registro de un dispositivo real.
Descripción	Registro de un dispositivo real en el sistema junto con sus endpoints y recursos
Pre-condición	Estar registrado en la web Haber iniciado sesión en la web Haber creado un dispositivo Shadow Haber creado un token de dispositivo
Post-condición	Ver en la pantalla de dispositivos el nuevo dispositivo registrado junto con sus recursos
Escenario principal	
1. El usuario realiza una petición de registro al sistema 2. El sistema registra el dispositivo, sus endpoints y sus recursos 3. El sistema devuelve una respuesta de éxito al usuario	
Escenario alternativo	
El token del usuario puede expirar en cualquier momento 1. El sistema devuelve una pantalla de error 2. El usuario pulsa el botón volver 3. El sistema vuelve a la pantalla de inicio de sesión 4. El usuario inicia sesión de nuevo	

4.3.4 Diagramas de secuencia

Los diagramas de secuencia muestran cómo un grupo de objetos interactúan entre sí pudiéndose observar el intercambio de mensajes que tiene lugar entre estos objetos a lo largo del tiempo.

A continuación, se muestran los diagramas de clases de los procesos más importantes del sistema.

- **Registro de un dispositivo IoT**

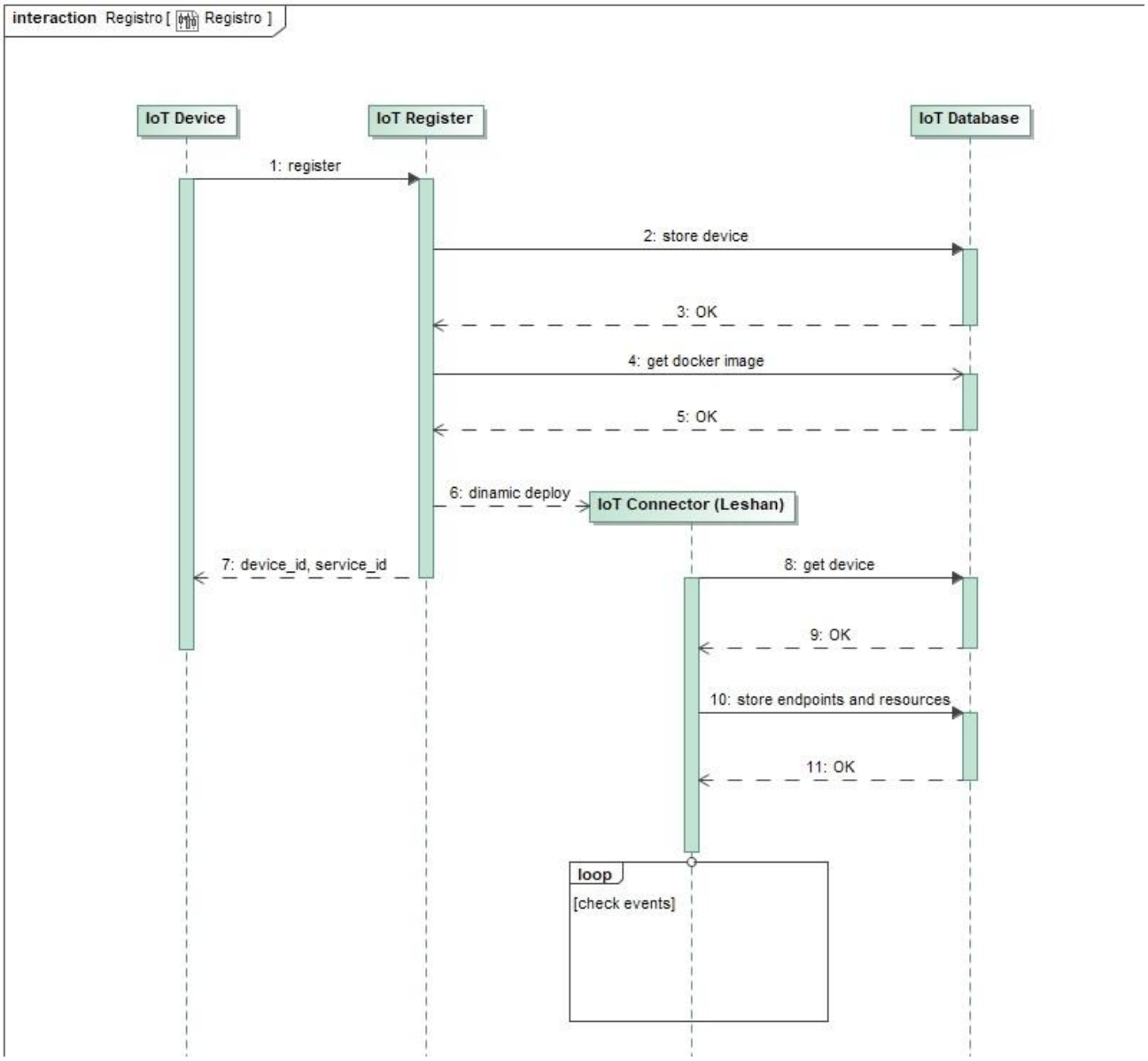


Figura 4.7: Diagrama de secuencia: Registro de un dispositivo IoT.

- **Monitorización IoT Connector (Leshan)**

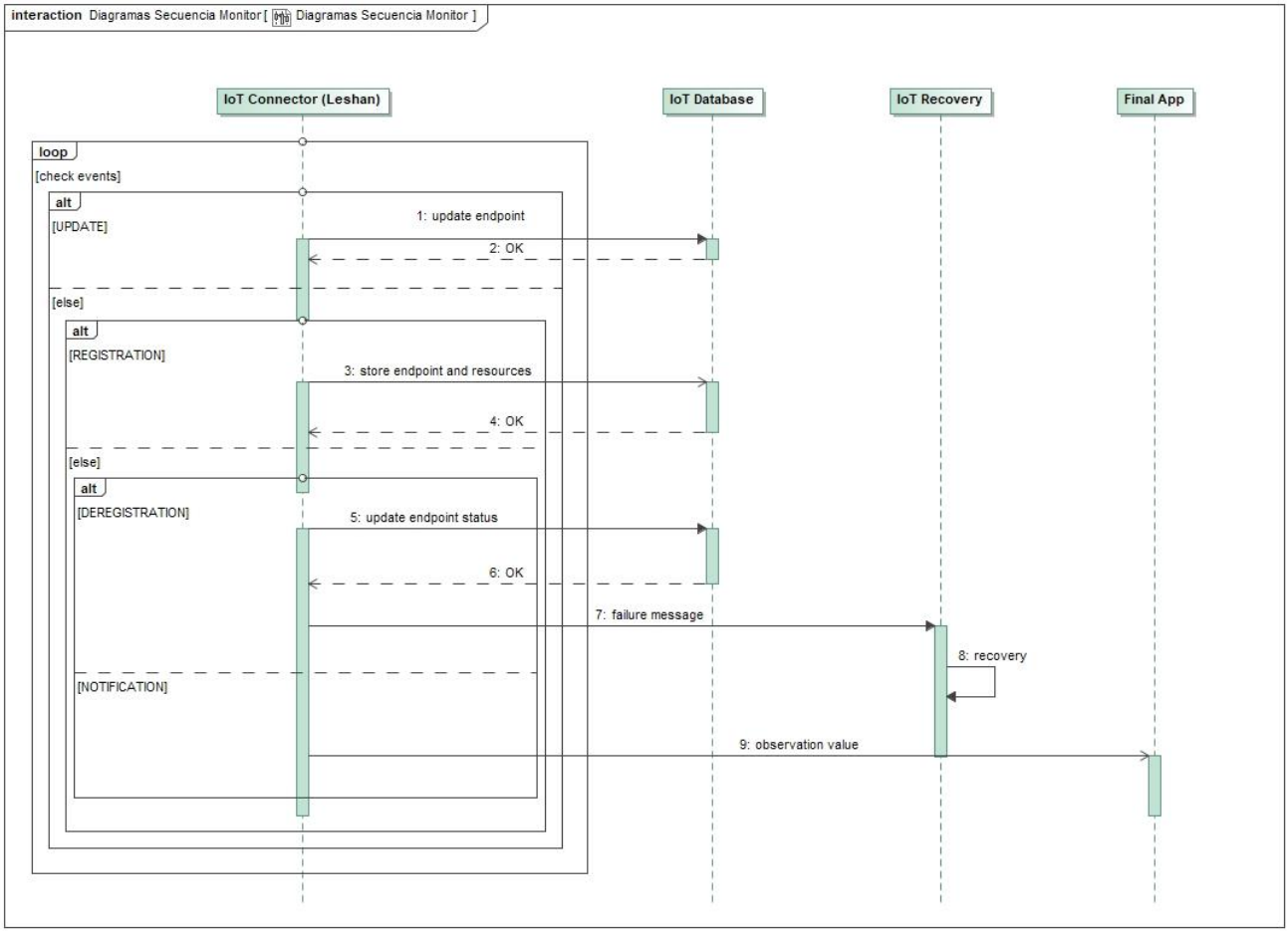


Figura 4.8: Diagrama de secuencia: Monitorización dispositivo IoT.

- **Recovery**

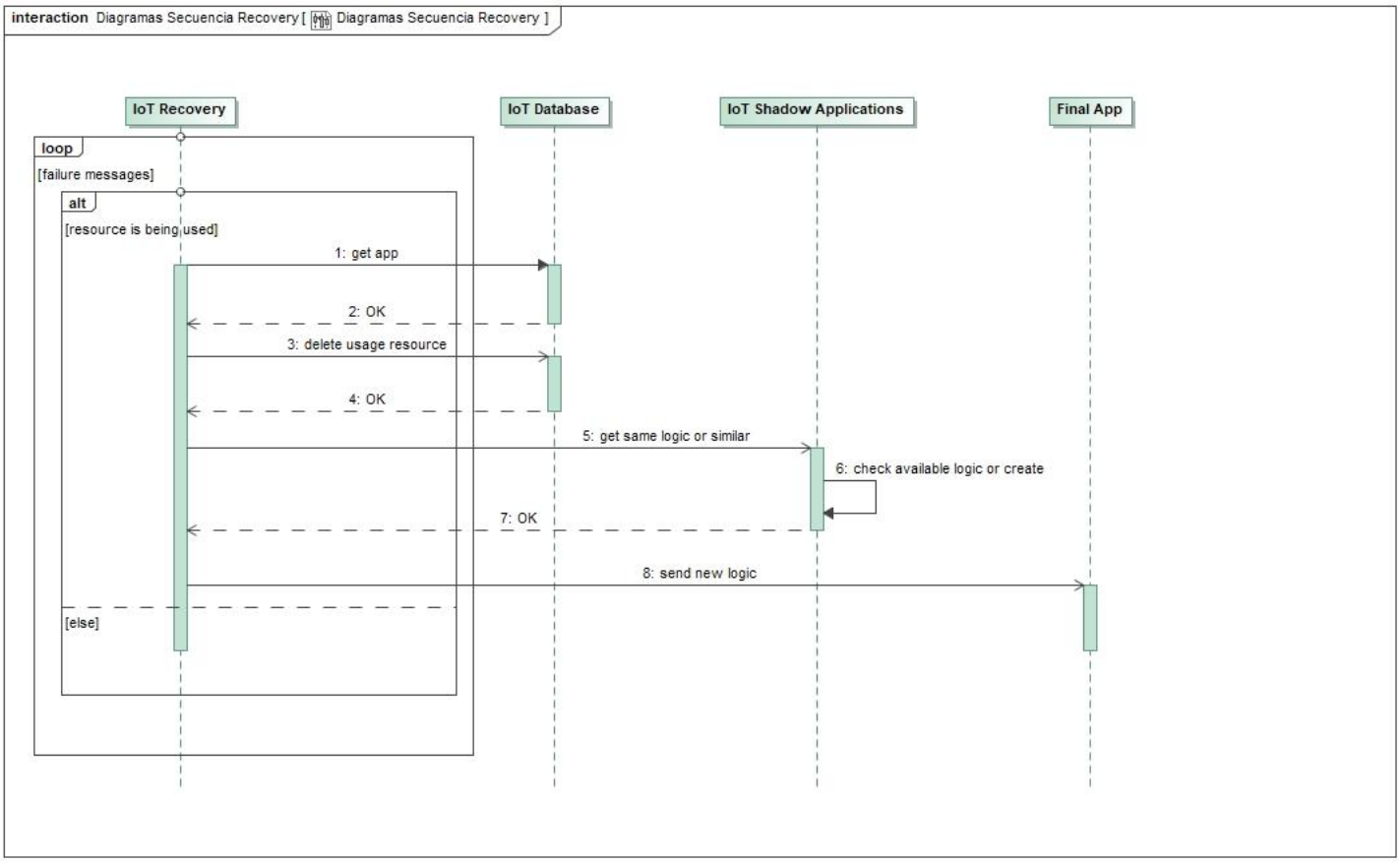


Figura 4.9: Diagrama de secuencia: Recovery

- **IoT Shadow Applications**

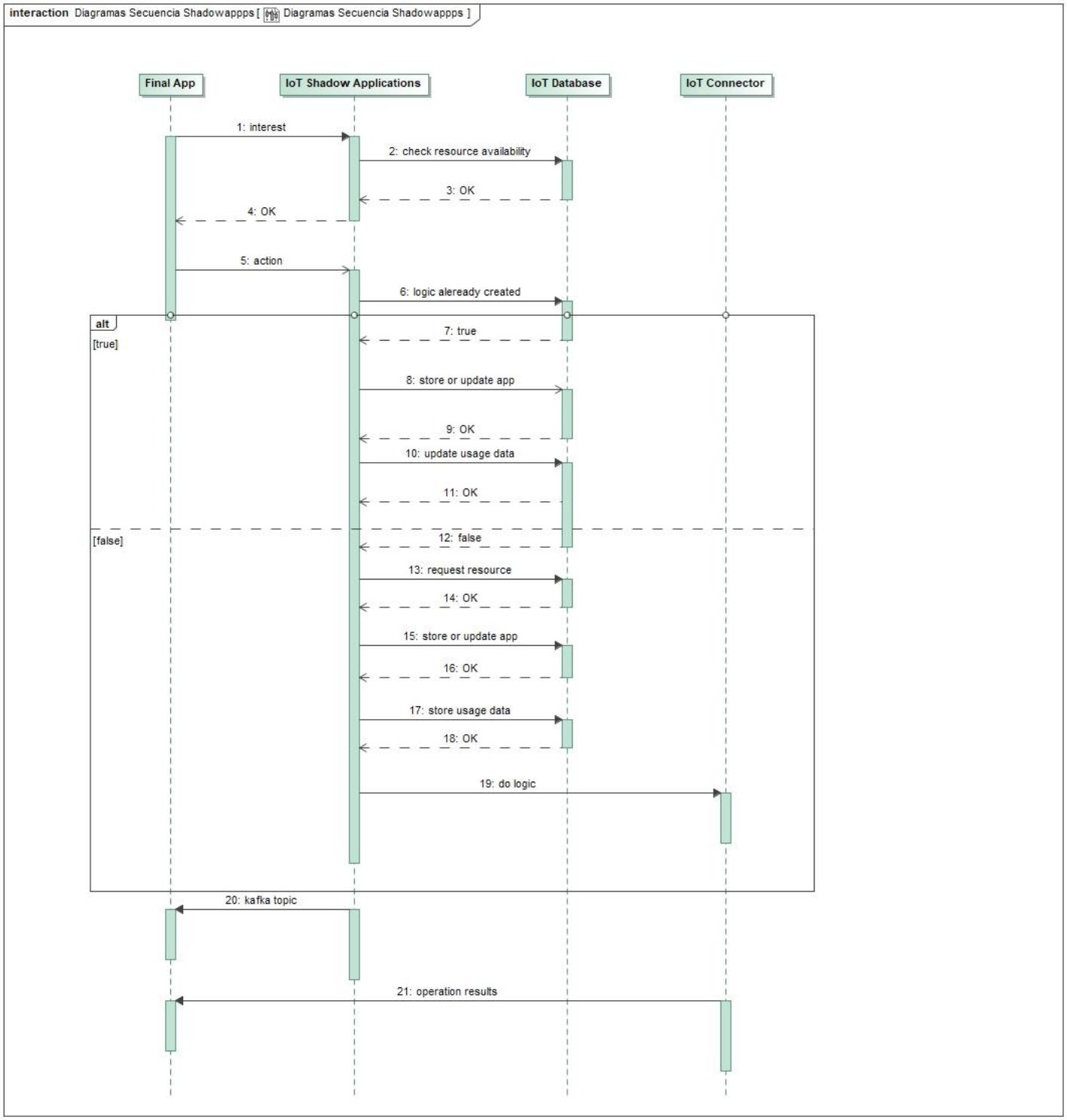


Figura 4.10: Diagrama de secuencia: Consulta disponibilidad de un recurso y creación de lógica.

4.4 Diseño

El diseño es una fase muy importante en el proceso de desarrollo de ingeniería del software, ya que, en este paso se diseñan los componentes que componen el sistema y se decide, además, cómo interaccionan entre si y también se define la arquitectura del sistema.

El diseño debe ser una guía que deben ser capaces de leer las personas que se encargarán del desarrollo, pruebas y mantenimiento del software y además debe proporcionar una idea completa del producto.

4.4.1 Arquitectura del sistema

La arquitectura del sistema está basada en componentes con lo que se consigue una mejor escalabilidad del sistema, ya que, el acoplamiento reducido de los módulos permite incluir uno nuevo de manera relativamente fácil, con lo que, la complejidad del sistema en conjunto se ve reducida. En la Figura 4.11 se puede contemplar la arquitectura del sistema, que es el resultado del conjunto de todos sus componentes.

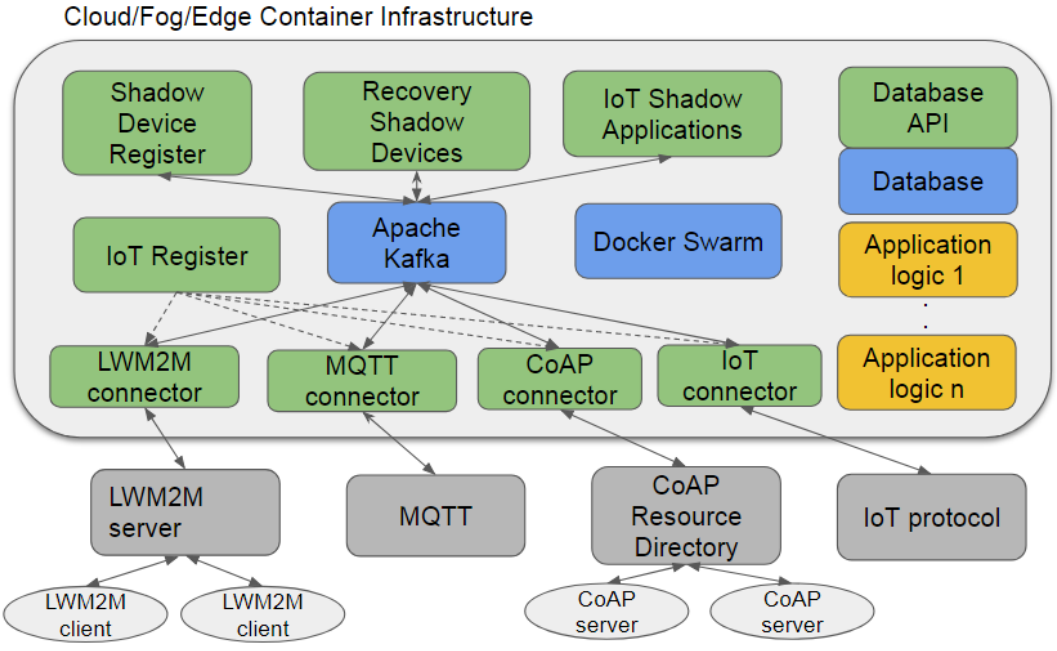


Figura 4.11: Arquitectura del Sistema.

En cuanto a la comunicación, algunos módulos se comunican solamente mediante peticiones HTTP, ya que ofrecen una API REST y luego para la comunicación interna de los demás módulos se utiliza la plataforma de comunicaciones Apache Kafka.

4.4.2 Formato de los datos

Tanto en las comunicaciones internas mediante Apache Kafka como en las comunicaciones mediante HTTP se ha utilizado el formato de datos JSON. Incluso la base de datos que se ha elegido (MongoDB) trabaja con este formato de datos, guardando documentos JSON (JavaScript Object Notation).

JSON es un formato ligero de intercambio de datos, fácil de leer y también de escribir. Se basa en un subconjunto del lenguaje de programación JavaScript, *Estándar ECMA-262 3ª Edición – diciembre 1999*. JSON es un formato de texto que es completamente independiente del lenguaje en el que se use y usa convenios que son familiares a los lenguajes de la familia C, incluyendo C, C#, C++, Java, JavaScript, Perl, Python y muchos más. Estas propiedades hacen que JSON sea un formato de intercambio de datos ideal.

JSON se basa en 2 estructuras:

1. Una colección clave/valor (objeto)
2. Una lista ordenada de valores

Estas estructuras son universales, cualquier lenguaje de programación moderno las soporta de una forma u otra.

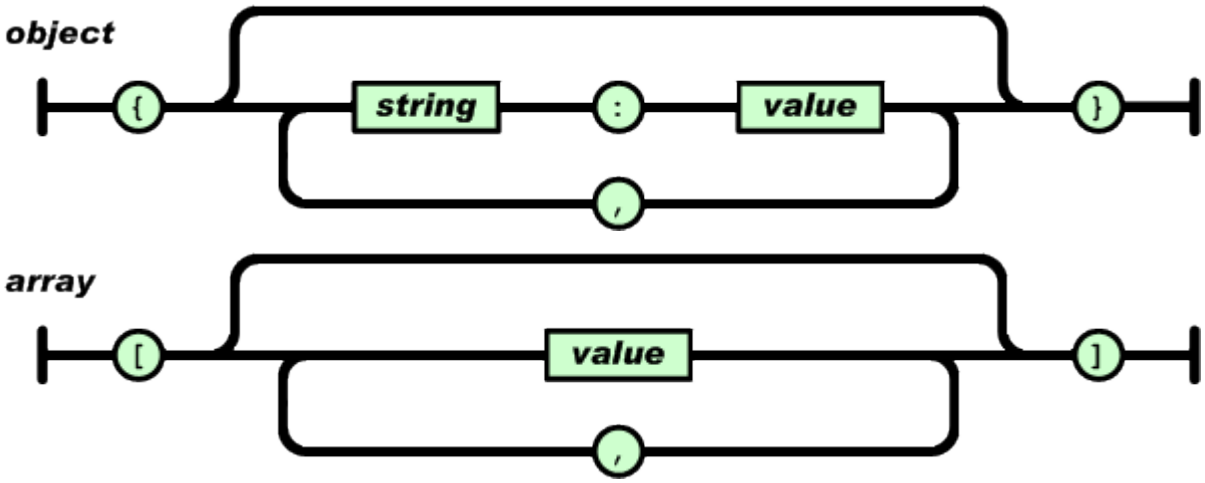


Figura 4.12: Estructuras Objeto y Lista.

Un valor válido puede ser uno de los siguientes: string, number, object, array, true, false, null**.** Además, estas estructuras se pueden anidar. En la Figura 4.13 se puede apreciar un ejemplo de un documento JSON.

```
{
        "nombre": "Juan",
        "apellidos": "Pérez Moreno",
        "aficiones": [
                "fútbol", "tenis", "correr"
        ],
        "dirección": {
             "calle": "C/Ríos n 21",
             "municipio": "Cártame",
             "ciudad": "Málaga"
         }
}
```

Figura 4.13: Ejemplo objeto JSON.

4.4.3 Diagrama Entidad Relación

A pesar de que se utiliza una base de datos no relacional, esto no significa que los datos se van a almacenar de cualquier forma, sino que, hace falta cierto orden y se ha considerado apropiado adaptar un esquema para la base de datos, sino, se convertiría en un cúmulo de información sin orden alguno que carecería de sentido común a medida que la cantidad de datos aumentaría.

En la Figura 4.14 se puede observar el diagrama de entidad relación de la base de datos.

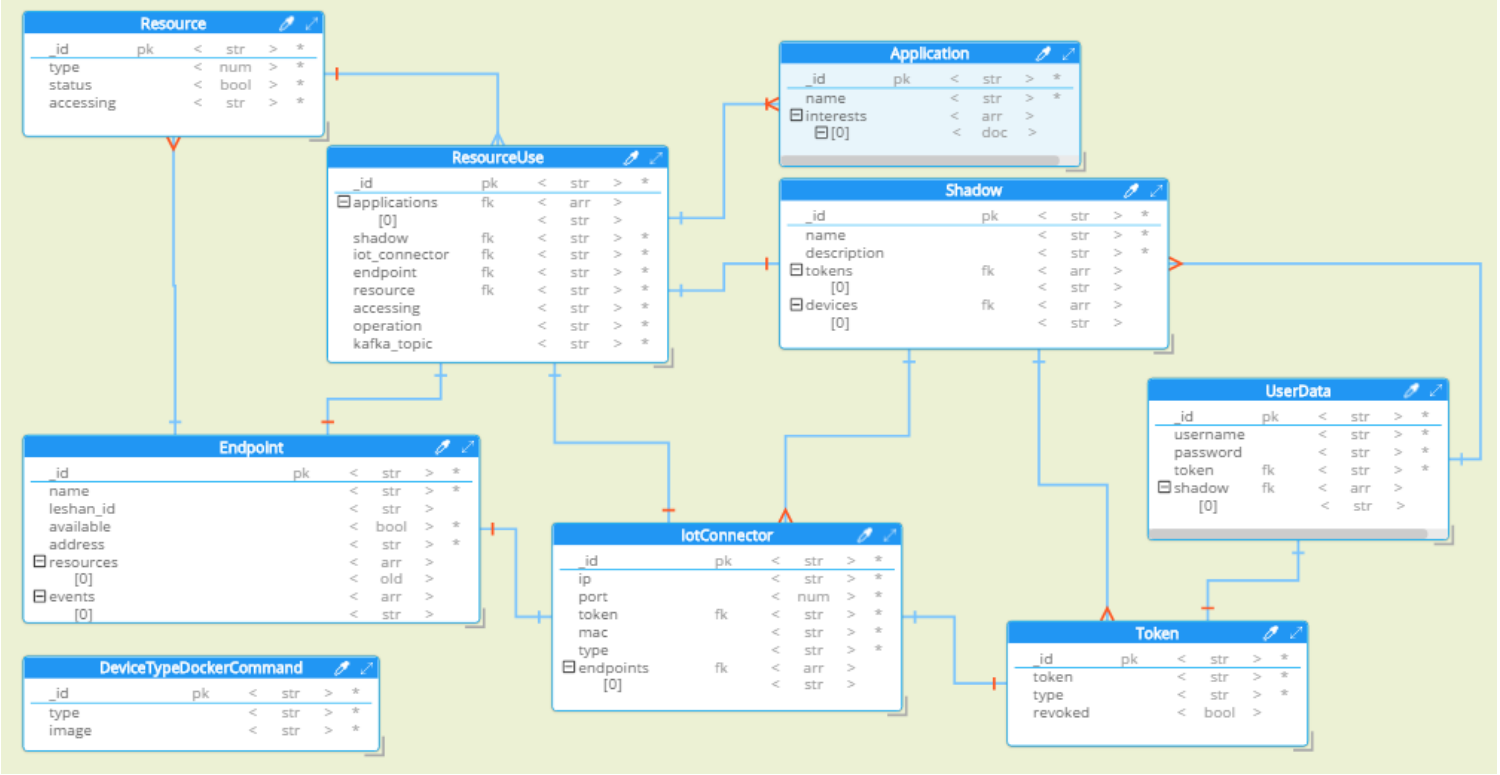


Figura 4.14: Diagrama de la base de datos.

4.5 Implementación

La implementación es otra actividad fundamental del ciclo de vida de desarrollo del software. Esta fase se basa en todas las fases anteriores (Análisis, Especificación y Diseño) y en ella se lleva a cabo la realización de los modelos del sistema.

El sistema está estructurado de forma modular buscando con esto características como la manejabilidad del código, sobrecarga reducida, reutilización de código y una mejor legibilidad y comprensión de este. La comunicación entre los diferentes módulos se realiza o a través de APIs REST y/o Apache Kafka dependiendo en cada caso.

El framework Django añade muchos archivos extras de configuración que se generan automáticamente y están dispuestos en una jerarquía compleja, por tanto, lo que figura en los diagramas de clases de los módulos son los archivos principales que se han desarrollado.

Esto es válido para todos los módulos en los que se ha utilizado este framework.

4.5.1 IoT Register

El módulo IoT Register ofrece una API REST con un único punto final en el que se realiza el registro de un dispositivo real (dispositivo IoT) en el sistema. Para ello se ha utilizado el framework de Python para web, Django y en la parte del servidor se ha utilizado el lenguaje Python 3.6. No ofrece una interfaz gráfica, solamente la API.

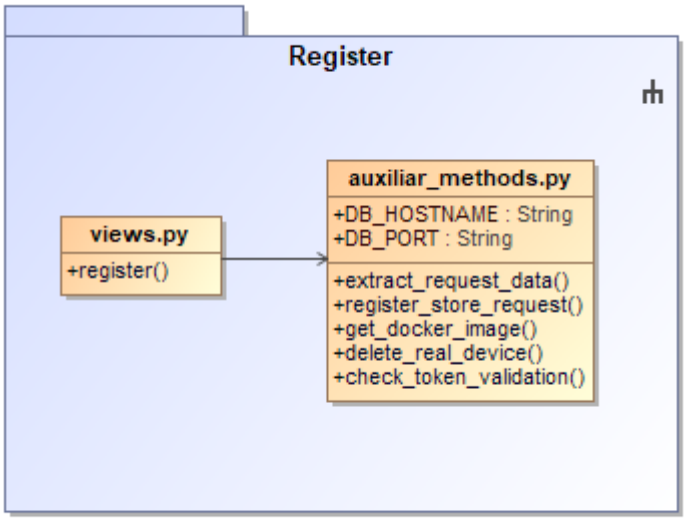


Figura 4.15: Diagrama de clases del módulo Register.

El núcleo de este módulo es la clase **views.py** que contiene la funcionalidad principal, que es el método register. Este método es la única función de la API que se puede utilizar directamente y se encarga de guardar en la base de datos la información sobre un dispositivo físico y además desplegar de forma dinámica un contenedor de Docker con una instancia del módulo IoT Connector, que se encargará de hacer la monitorización del nuevo dispositivo físico registrado. Para ello, se apoya en una clase auxiliar cuyo propósito es no sobrecargar el código y aumentar la reutilización de este.

4.5.2 Shadow Device Register

El módulo Shadow Device Register ofrece una API REST y una interfaz web para gestionar los dispositivos que pertenecen al sistema. Para ello se ha utilizado el framework de Python para web, Django y en la parte del servidor se ha utilizado el lenguaje Python 3.6

En la implementación se ha utilizado el patrón de diseño modelo vista controlador (MVC) por los beneficios que este patrón aporta, separando la representación de los datos de la lógica que los maneja y del modelo. El framework Django se adapta muy bien a este patrón de diseño y permite su implementación de manera muy sencilla.

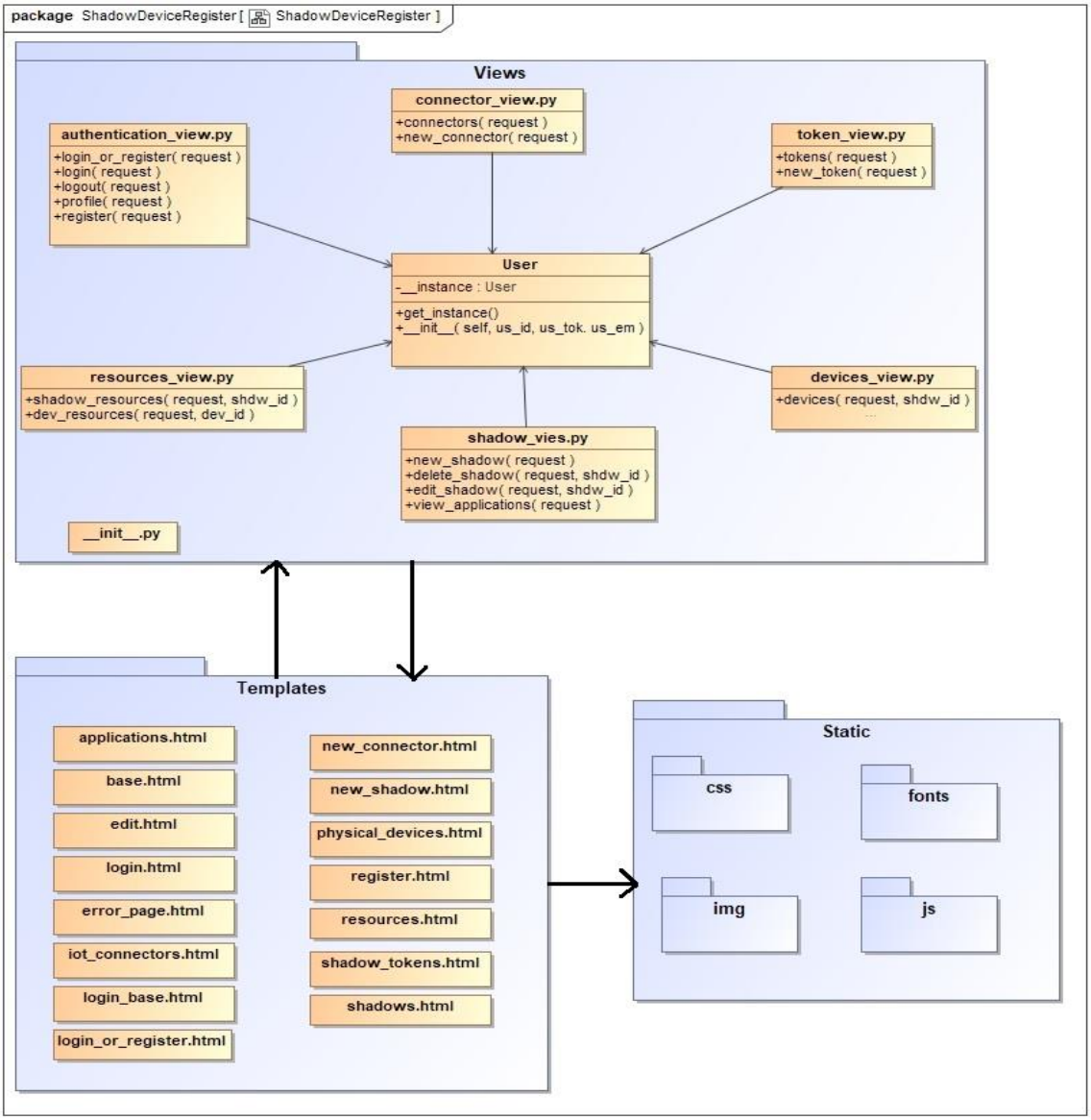


Figura 4.16: Diagrama de clases del módulo Shadow Device Register.

En la implementación también se ha hecho uso de otro patrón de diseño, concretamente, el patrón Singleton, en la clase **User** para tener los datos del usuario de la sesión activa en cada momento.

Este patrón previene que se instancien otros objetos con datos de la sesión actual del usuario y en este caso asegura que todas las clases acceden a la misma información sin que haya varias sesiones simultáneas o inconsistencia en los datos.

La clase **authentication_view** se ocupa de las vistas y la lógica de la página web que tienen que ver con las acciones de autentificación, registro, inicio y cierre de sesión y también una vez identificado el usuario, mostrar la pantalla principal de su perfil.

La clase **connector_view** contiene la lógica acerca de los conectores que hay generados en el sistema con 2 funcionalidades básicas: ver los conectores que hay y crear uno nuevo. Hay que destacar que los conectores no pertenecen a un usuario, sino que, pertenecen al sistema, es decir, los conectores que se crean son comunes a todos los usuarios.

La clase **token_view** contiene la lógica acerca de los tokens de un dispositivo Shadow con 3 funcionalidades básicas: ver los tokens que hay, crear uno nuevo y revocar un token. La acción de revocado la realiza el mismo método que los muestra, solo que se realiza mediante peticiones HTTP diferentes. Para ver es una petición GET y para revocar una petición POST.

La clase **devices_view** contiene la lógica acerca de los dispositivos físicos asociados a un dispositivo Shadow. Contiene 2 funcionalidades básicas: mostrar todos los dispositivos físicos asociados a un dispositivo Shadow y borrar un dispositivo físico en concreto.

La clase **shadow_view** contiene la lógica acerca de los dispositivos Shadow con 4 funcionalidades básicas: crear, editar y borrar un dispositivo Shadow y además permite consultar las aplicaciones que han mostrado algún interés en el sistema solicitando algún recurso.

Por último, la clase **resources_view** contiene la lógica acerca de los recursos de los dispositivos físicos y tiene 3 funcionalidades básicas: ver los recursos de un dispositivo físico, ver los recursos de un dispositivo Shadow y borrar un recurso.

4.5.3 IoT Connector (Leshan Monitor)

El módulo IoT Connector tiene como objetivo realizar la monitorización de un dispositivo físico (dispositivo IoT) registrado en el sistema y además desempeña también otras actividades de lectura y envío de datos de un dispositivo físico a una aplicación.

Para ello se ha desarrollado en Python 3.4 ya que con versiones posteriores se han obtenido problemas y en esta versión es estable y funciona bien.

Para la implementación se ha hecho uso del patrón de diseño Singleton en una clase y también se ha intentado hacer uso de buenas prácticas de programación.

En la Figura 4.13 se puede ver el diagrama de clases de este módulo,

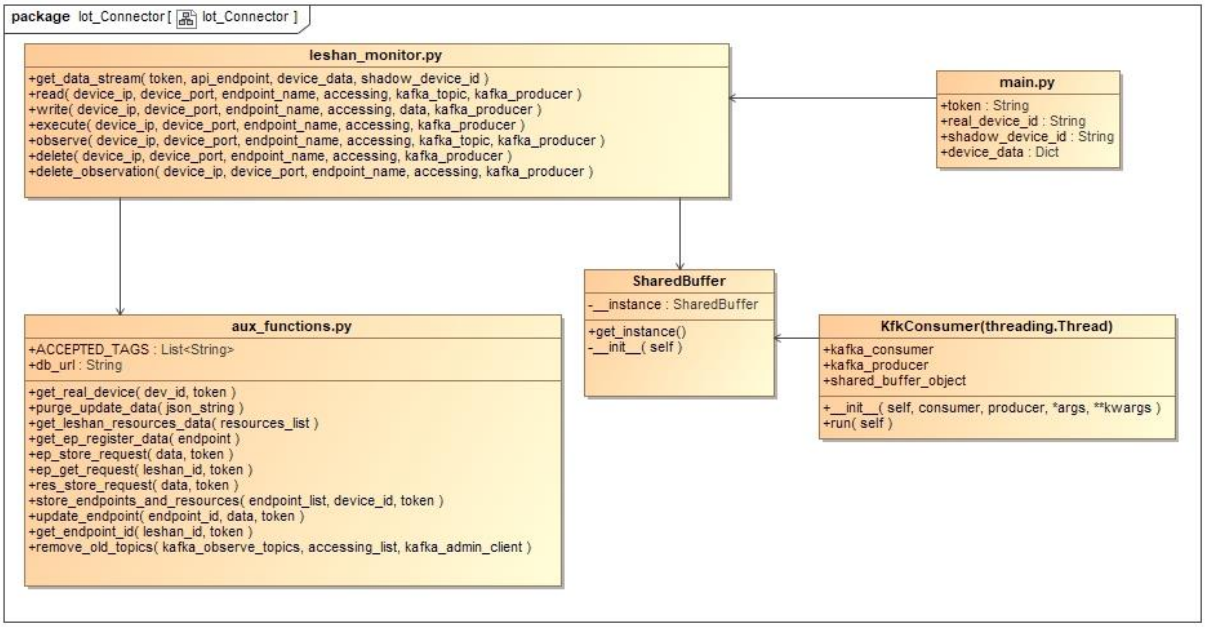


Figura 4.17: Diagrama de clases del módulo IoT Connector.

La clase **main** se encarga de conseguir unas variables de entorno que el contenedor recibe del módulo IoT Register y también ejecutar la clase del monitor, **leshan_monitor** pasándole los datos que esta necesita.

La clase **leshan_monitor** contiene la lógica que es el núcleo de este módulo y basándose en una clase auxiliar (**aux_functions**) hace la funcionalidad de monitorizar los eventos de un dispositivo físico Leshan. Los eventos pueden ser los siguientes:

- **UPDATED**: Evento de rutina que se guarda en la base de datos en el endpoint específico al que pertenece.

- **REGISTRATION**: Indica que un nuevo endpoint se ha registrado en el dispositivo leshan.
- **DEREGISTRATION**: Indica que un endpoint de un dispositivo Leshan ha fallado.
- **NOTIFICATION**: Contiene el valor de un dato que se está observando.

Cuando un evento REGISTRATION ocurre, el monitor lo que hace es tratar ese evento registrando en la base de datos el nuevo endpoint que se ha registrado y todos los recursos que este ofrece.

Cuando un evento DEREGISTRATION ocurre, el monitor lo que hace es tratar ese evento actualizando en la base de datos el endpoint y todos los recursos que este ofrece. La actualización consiste en indicar que no están disponibles, no se borran.

Cuando un evento NOTIFICATION ocurre, mediante Kafka envía el dato a la aplicación que lo requiere.

Otra funcionalidad es la realización de acciones sobre los recursos que se monitorizan. La hebra KfkConsumer está constantemente recibiendo peticiones por parte de las aplicaciones y las apila en el buffer compartido desde donde la hebra principal las coge y las ejecuta.

La clase **shared_buffer** ofrece una instancia de sí misma mediante el patrón de diseño Singleton y es utilizada por la hebra principal y la hebra KfkConsumer. Las hebras han de acceder al buffer siempre en exclusión mutua.

La clase **KfkConsumer** tiene una única funcionalidad y es leer constantemente las peticiones de recursos y de acciones sobre recursos por parte de las aplicaciones. Se encarga de recibir estas peticiones y apilarlas en el buffer para que la hebra principal las vaya atendiendo.

4.5.4 IoT Shadow Applications

El módulo IoT Shadow Applications ofrece una API REST con 2 puntos finales para aplicaciones que quieran interaccionar con el sistema. Una aplicación puede mostrar interés por un recurso (consultar disponibilidad) y además puede pedir al sistema realizar una operación sobre un recurso. Para ello se ha utilizado el framework de Python para web, Django y en la parte del servidor se ha utilizado el lenguaje Python 3.6. Además, se ha hecho uso de patrones de diseño. El módulo no ofrece una interfaz gráfica, solamente la API.

En la Figura 4.18 se puede ver el diagrama de clases de este módulo.

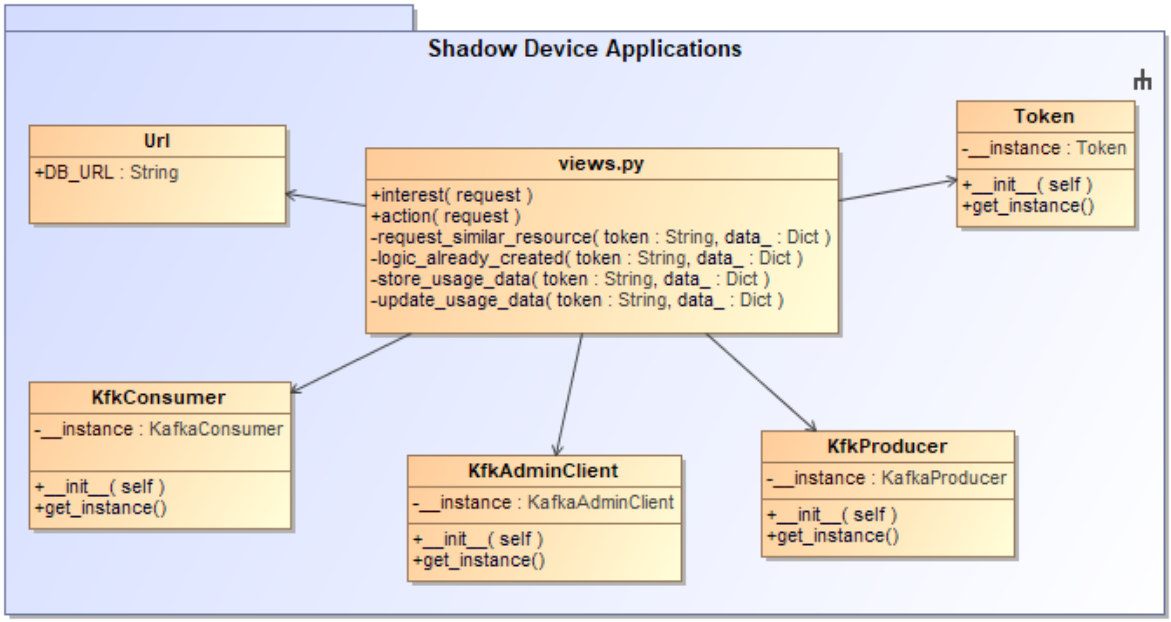


Figura 4.18 Diagrama de clases del módulo IoT Shadow Applications.

La clase **URL** contiene la URL de acceso a la base de datos.

Las clases **KfkConsumer, KfkAdminClient, KfkProducer y Token** ofrecen una instancia de ellas mismas mediante el patrón de diseño Singleton y son utilizadas por la clase principal **views.** Estas clases no tienen funcionalidad por ellas mismas, sino que son clases auxiliares de la clase principal.

La clase **views** es el núcleo de este módulo y dispone de dos métodos principales y varios auxiliares. Los dos métodos principales son los que ofrecen en la API REST. La

funcionalidad de este módulo es permitir que las aplicaciones finales muestren interés en los diferentes tipos de recursos que hay o puede haber en el sistema, lo que se traduce en consultar la disponibilidad de un recurso.

También permite a las aplicaciones ejecutar operaciones (o lo que es lo mismo, crear una lógica) sobre estos recursos. Las operaciones que se pueden ejecutar son varias: **READ, OBSERVE, WRITE, EXECUTE, DELETE y DELETE_OBSERVATION**.

- READ: consiste en consultar el valor de un recurso en el instante de tiempo exacto en el que se hace la petición.
- OBSERVE: consiste en realizar una observación prolongada en el tiempo sobre un recurso y obtener los valores de esta observación.
- WRITE: consiste en asignar los valores deseados a los recursos.
- EXECUTE: lo que hace es reiniciar los valores de un recurso.
- DELETE: solicita borrar un recurso.
- DELETE_OBSERVATION: consiste en parar una observación.

Cuando se recibe una solicitud para realizar una de las operaciones descritas sobre un recurso disponible, esta clase manda un mensaje mediante Kafka a una instancia concreta de IoT Monitor para que haga una acción. Este mensaje será recogido en el IoT Connector por la hebra que está continuamente escuchando peticiones, como se ha explicado en el apartado de implementación del IoT Connector.

4.5.5 Recovery Shadow Devices

El objetivo de este módulo es realizar la recuperación en caso de fallo de un dispositivo IoT y asegurar la continuidad del servicio que las aplicaciones están solicitando. Se ha implementado en Python 3.6 y se ha hecho uso del patrón de diseño Singleton.

En la Figura 4.19 se puede ver el diagrama de clases de este módulo.

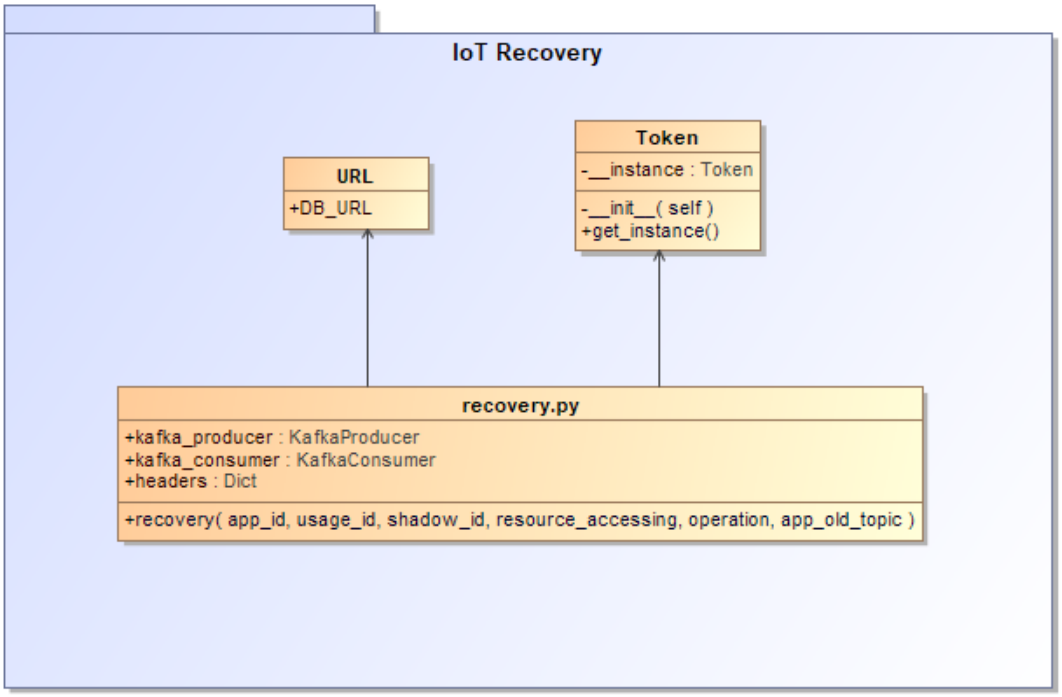


Figura 4.19 Diagrama de clases del módulo IoT Shadow Applications.

La clase **URL** contiene la URL de acceso a la base de datos.

La clase **Token** está implementada utilizando el patrón Singleton porque realmente no interesa que haya más de una instancia de la clase. Es una clase contenedora que tiene un token, el cual se utiliza para establecer comunicaciones seguras con las diferentes APIs del sistema.

La clase **recovery** es el núcleo de este módulo y su objetivo es estar constantemente recibiendo mensajes a través de Kafka para iniciar el protocolo de restauración de la lógica (cuando sea posible) en caso de fallo.

En caso de que ninguna aplicación esté utilizando algunos recursos de los dispositivos físicos que han fallado, no hace nada. Si algún recurso o varios recursos estaban siendo utilizados en el momento que ha fallado, esta clase se encarga de buscar una lógica similar que esté ya creada o crear una nueva lógica en un recurso similar al recurso que ya no está disponible.

Cuando en el sistema no haya más recursos similares disponibles, lo único que hará es mandar una notificación a la aplicación de que no hay más recursos disponibles.

4.5.6 Database API

El objetivo de este módulo es realizar todas las interacciones con la base de datos mediante una API REST protegida por token y también proveer una capa de abstracción con la base de datos. Está formado por varios submódulos, principalmente Modelo, Vistas y Auth.

Se ha implementado en Python 3.6 y haciendo uso del framework Django. El módulo Database API no ofrece una interfaz gráfica, solamente la API REST.

En la Figura 4.20 se puede ver el diagrama de clases de este módulo.

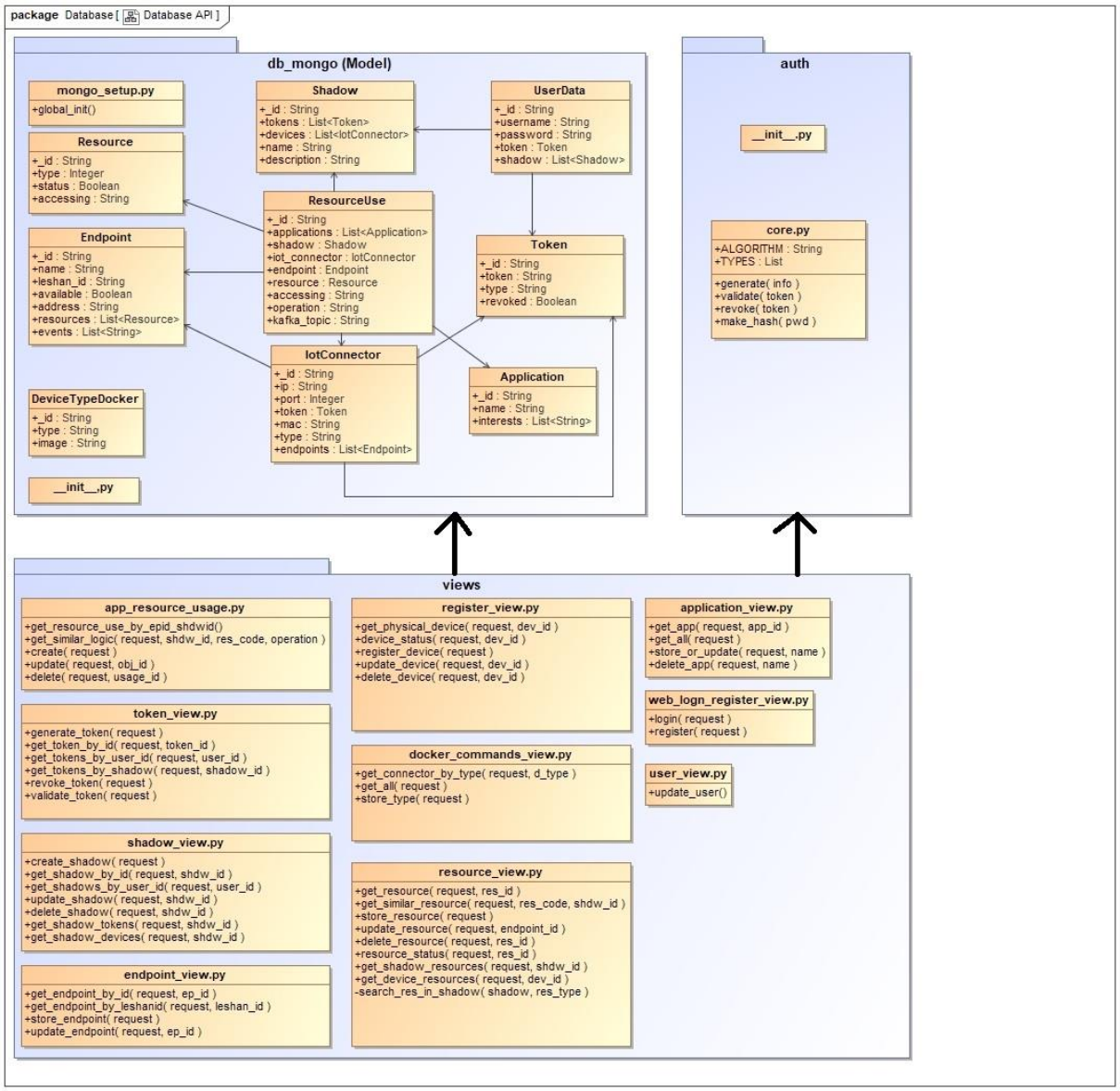


Figura 4.20 Diagrama de clases del módulo IoT Database.

En el módulo **views** se encuentra una vista (lógica) por cada clase del modelo que representa una entidad de la base de datos, y en estas vistas se realizan las diferentes consultas que se pueden ver en la Figura 4.20 que representa el diagrama de clases de la base de datos.

Las vistas se han separado en clases diferentes, exactamente una clase por cada entidad de la base de datos para evitar sobrecarga de complejidad de una única clase y favorecer en cambio la legibilidad, comprensión, modularidad y reutilización del código.

El módulo **auth** está formado por una clase principal (**core**) cuya función es realizar todas las operaciones relacionadas con la autentificación por token. Las funcionalidades son:

- Generar un token y guardarlo en la base de datos
- Validar un token
- Revocar un token y actualizar su estado de la base de datos
- Hacer el hash a una contraseña

Todas las vistas utilizan esta clase para verificar que las peticiones que se realizan a la API llevan un token expedido por esta clase y que sea válido. Si un token no es válido (revocado o ha caducado) o si el token no se pasa en la cabecera de la petición HTTP, la consulta a la base de datos no se efectuará.

El módulo **db_mongo** contiene diez clases y cada una de ellas representa una colección o entidad de la base de datos. Estas clases extienden de la clase **mongoengine.Document** que representa un objeto de la base de datos y sobre el cual se puede realizar operaciones relativas a la manipulación de un objeto de la base de datos que se heredan de dicha clase.

4.5.7 Iteraciones de desarrollo.

I. **Iteración 1**

Objetivos iteración:

La primera iteración comprende el desarrollo de 3 componentes principales de la arquitectura (ver Figura 1.2):

1. IoT Register (IoT Register)
2. IoT Monitor (LWM2M connector)
3. IoT Database (Database y Database API).

Estos componentes corresponden a los siguientes requisitos funcionales del sistema obtenidos en la fase de análisis:

RF.1 – Registrar un nuevo dispositivo físico en el sistema

RF.2 – Desplegar de forma dinámica instancias de un monitor IOT Connector.

RF.19 – Identificar los Endpoints de un dispositivo físico registrado y almacenar sus datos en la Base de Datos.

RF.20 – Identificar los recursos de cada Endpoint y almacenar los datos en la Base de Datos.

RF.21 – Recibir eventos de un dispositivo en tiempo real**.**

RF.22 – Almacenar en la base de datos los datos de los eventos recibidos asociados a un dispositivo físico.

RF.29 – Verificar en cada consulta la validez del token.

RF.30 – Guardar información de un dispositivo real en la base de datos.

RF.31 – Consultar en la base de datos un dispositivo físico por su id.

RF.32 – Borrar un dispositivo físico de la base de datos por su id.

RF.34 – Guardar información de un endpoint en la base de datos.

RF.35 – Consultar en la base de datos un endpoint por su id.

RF.36 – Actualizar un endpoint en la base de datos por su id.

RF.37 – Guardar información de un recurso en la base de datos.

RF.38 – Consultar en la base de datos un recurso por su id.

RF.39 – Actualizar un recurso en la base de datos por su id.

II. Iteración 2

Objetivos iteración:

La segunda iteración comprende el desarrollo de 1 componente principal de la arquitectura (ver Figura 1.2) y la ampliación de otros módulos, que son resultado de la Iteración 1:

1. IoT Web (Shadow Device Register)
2. Ampliación del módulo Database API:
    a. Nuevas colecciones añadidas: Token, Shadow, UserData.
    b. Operaciones CRUD para las nuevas colecciones
    c. Autentificación por Token (JWT)
3. Refactorización de las vistas de la base de datos.

Estos componentes corresponden a los siguientes requisitos funcionales del sistema obtenidos en la fase de análisis:

RF.3 – El usuario podrá registrarse en el sistema desde la web con email y contraseña.

RF.4 – El usuario podrá identificarse en el sistema desde la web con email y contraseña.

RF.5 – El sistema asociará un Token de identificación de usuario con validez.

RF.6 – El usuario podrá cerrar sesión en cualquier momento.

RF.7 – El sistema mostrará al usuario los dispositivos Shadow registrados por él.

RF.8 – El sistema mostrará al usuario los tipos de dispositivos físicos disponibles que hay registrados en el sistema.

RF.9 – El usuario podrá Crear, Editar y Borrar dispositivos Shadow.

RF.10 – El usuario podrá ver los tokens asociados a un dispositivo Shadow.

RF.11 – El usuario podrá crear un Token que representará un dispositivo físico asociado al dispositivo Shadow.

RF.12 – El usuario podrá revocar desde la interfaz web en cualquier momento un Token de un dispositivo físico asociado.

RF.44 – Consultar el tipo de dispositivo físico.

RF.45 – Consultar en la base de datos todos los tipos de dispositivos físicos.

RF.46 – Generar un Token y guardar en la base de datos.

RF.47 – Validar un Token.

RF.48 – Revocar un Token (actualizar su estado en la base de datos).

RF.49 – Consultar en la base de datos un Token por su id.

RF.50 – Consultar en la base de datos un Token por el id del usuario al que pertenece.

RF.51 – Consultar en la base de datos los Token por el id del dispositivo Shadow al que pertenece.

RF.52 – Crear un dispositivo Shadow.

RF.53 – Actualizar un dispositivo Shadow.

RF.54 – Borrar un dispositivo Shadow.

RF.55 – Consultar en la base de datos los dispositivos Shadow que pertenecen a un usuario por el id de este último.

RF.56 – Consultar en la base de datos un dispositivo Shadow por su id.

RF.59 – Guardar información de registro de un usuario en la base de datos.

RF.60 – Actualizar información de registro de un usuario de la base de datos.

RF.61 – Verificar los datos de autentificación un usuario.

III. **Iteración 3**

Objetivos iteración:

La segunda iteración comprende el desarrollo de 2 componentes principales de la arquitectura (ver Figura 1.2), la ampliación de otros módulos, que son resultado de la Iteración 2:

1. Recovery Shadow Devices
2. IoT Shadow Applications
3. Kafka / Zookeeper
4. Ampliación del módulo Database API:
    a. Buscar recurso similar en un dispositivo Shadow concreto
    b. Buscar recurso similar en cualquier dispositivo Shadow
5. Ampliación del módulo Shadow Device Register:
    a. Añadir vista dispositivos físicos.
    b. Borrar dispositivo físico.
    c. Añadir vista Recursos.
    d. Borrar un recurso.
6. Ampliación del módulo IoT Connector:
    a. Leer mensajes desde Kafka
    b. Mandar mensajes por Kafka
    c. Notificar al módulo Recovery Shadow Devices la caída o modificación de un dispositivo.
    d. Nuevas operaciones: read, observe, execute, write

Estos componentes corresponden a los siguientes requisitos funcionales del sistema obtenidos en la fase de análisis:

RF.13 – El usuario podrá agregar desde la interfaz web un nuevo tipo de dispositivo físico.

RF.14 – El usuario podrá ver los dispositivos físicos de un Shadow registrados en el sistema e información básica sobre ellos: Nombre, Última vez conectado, Token y Estado de uso.

RF.15 – El usuario podrá ver los recursos asociados a un dispositivo físico e información básica sobre ellos: Código, Tipo de recurso y Estado de uso.

RF.16 – El usuario podrá ver todos los recursos asociados a un dispositivo Shadow e información básica sobre ellos: Código, Tipo de recurso y Estado de uso.

RF.17 – El usuario podrá borrar un dispositivo físico junto con todos sus recursos.

RF.18 – El usuario podrá borrar un recurso de un dispositivo físico.

RF.23 – Notificar mediante Kafka la desconexión y/o cambio de un dispositivo.

RF.24 – Leer mensajes desde un tópico Kafka.

RF.25 – Operación READ para leer el valor de un recurso.

RF.26 – Operación EXECUTE para realizar una acción sobre un recurso.

RF.27 – Operación WRITE para leer el valor de un recurso.

RF.28 – Operación OBSERVE para monitorizar / observar el valor de un recurso.

RF.33 – Consultar el estado de uso de un dispositivo físico por su id.

RF.40 – Consultar en la base de datos los recursos de un dispositivo Shadow por el id de este último.

RF.41 – Consultar en la base de datos los recursos de un dispositivo (IoT Connector) por el id de este último.

RF.42 – Consultar el estado de uso de un recurso por su id.

RF.43 – Guardar el tipo de dispositivo físico.

RF.57 – Consultar en la base de datos los Token de un dispositivo Shadow.

RF.58 – Consultar en la base de datos los dispositivos físicos de un dispositivo Shadow.

RF.62 – Buscar un endpoint con la misma lógica en un dispositivo Shadow concreto.

RF.63 – Buscar un endpoint con la misma lógica en cualquier dispositivo Shadow.

RF.64 – Registrar los datos de una nueva aplicación.

RF.65 – Leer los datos de una aplicación.

RF.66 – Consultar todas las aplicaciones que hay en la base de datos.

RF.67 – Borrar una aplicación.

RF.68 – Actualizar un dispositivo físico de la base de datos por su id.

RF.69 – Buscar un recurso similar en la base de datos dado un tipo de recurso.

RF.70 – Buscar un recurso similar en la base de datos dado un tipo de recurso y especificando un dispositivo Shadow en concreto mediante id.

RF.71 – Guardar los datos del uso de un recurso en la base de datos.

RF.72 – Borrar los datos del uso de un recurso en la base de datos por el id.

RF.73 – Actualizar los datos del uso de un recurso en la base de datos por el id.

RF.74 – Consultar los datos de uso de un recurso de un Endpoint en concreto de un dispositivo Shadow en concreto dados sus identificadores.

RF.75 – Consultar si un recurso está siendo utilizado por una aplicación final.

RF.76 – Recibir mensajes de eventos (eliminación / desconexión de recurso / registro) en endpoints mediante Kafka.

RF.77 – Buscar otro endpoint en el mismo dispositivo Shadow que tenga la misma lógica que el dispositivo caído / desconectado.

RF.78 – Notificar mediante Kafka a las aplicaciones que estén usando un dispositivo Shadow.

RF.79 – Mandar un mensaje a través de Kafka a un IoT Connector para que haga la lógica que se requiere.

RF.80 – Permitir a una aplicación mostrar interés en un recurso de un dispositivo Shadow en concreto

RF.81 – Permitir a una aplicación mostrar interés en un recurso sin importar de qué dispositivo Shadow proviene

RF.82 – Pedir un recurso de un dispositivo Shadow en concreto.

RF.83 – Pedir un recurso sin importar de qué dispositivo Shadow proviene.

RF.84 – Mandar mensaje mediante Kafka a un IoT Connector para crear la lógica deseada.

RF.85 – Crear un tópico Kafka (o devolver uno ya existente) entre la aplicación final y el IoT Connector para que haya comunicación entre estos dos.

5

Conclusiones

La realización de este proyecto ha dado como resultado un sistema de monitorización de dispositivos IoT tolerante a fallos mediante la definición de los llamados Dispositivos Shadow. Estos dispositivos Shadow no son físicos, sino que, son una entidad abstracta que está compuesta por varios dispositivos físicos (dispositivos IoT) que se registran en el sistema.

De momento el sistema es capaz de monitorizar solamente dispositivos Leshan, pero, de cara al futuro, la lógica del sistema se puede ampliar para abarcar otros tipos de dispositivos como MQTT y CoAP. Esto supone el desarrollo e implementación de nuevos módulos IoT Monitor específicos para cada tipo de dispositivo IoT.

La estructura modular del sistema se adecúa muy bien a esta visión al futuro ya que integrar un nuevo módulo en una arquitectura con esta característica resultará en una tarea de complejidad reducida.

Por otra parte, la metodología ágil e iterativa adoptada para el desarrollo de este proyecto ha supuesto una ventaja, ya que, en las distintas reuniones con el tutor surgían nuevos cambios constantemente, ya sean nuevos requisitos o modificaciones en los requisitos ya existentes o se veía una manera mejor de hacer las cosas y se apostaba por ella.

De cara a la experiencia personal, este proyecto ha resultado ser todo un reto, ya que, abarca muchas tecnologías nuevas con las que tenía que familiarizarme sobre la marcha, lo que ha supuesto posteriores refactorizaciones y mejoras. Ha sido y es un proyecto muy interesante y también bonito, un proyecto con el que se ha disfrutado al ver los resultados positivos y por supuesto ha sido beneficioso para mí como futuro profesional debido a la necesidad de profundizar en conocimientos previos y aprender nuevas tecnologías.

Referencias

Kafka:

- https://hub.docker.com/_/zookeeper
- https://hub.docker.com/r/wurstmeister/kafka
- http://wurstmeister.github.io/kafka-docker/
- https://github.com/edenhill/kafkacat
- https://kafka-python.readthedocs.io/en/master/usage.html
- https://kafka-python.readthedocs.io/en/master/apidoc/modules.html
- https://medium.com/@mukeshkumar_46704/consume-json-messages-from-kafka-using-kafka-pythons-deserializer-859f5d39e02c
- https://dzone.com/articles/case-study-to-understand-kafka-consumer-and-its-of

Docker:

- https://docs.docker.com/install/linux/docker-ce/ubuntu/#docker-ee-customers
- https://www.shubhamdipt.com/blog/python3-and-mongodb-using-docker/
- https://forums.docker.com/t/docker-build-ignoring-dockerignore/11991
- https://www.projectatomic.io/blog/2015/07/what-are-docker-none-none-images/
- https://nickjanetakis.com/blog/docker-tip-54-fixing-connection-reset-by-peer-or-similar-errors
- https://docs.docker.com/reference/
- https://docs.docker.com/get-started/
- https://www.docker.com/why-docker
- https://docs.docker.com/develop/sdk/examples/

- https://docs.docker.com/develop/sdk/
- https://medium.com/@vladyslav.krylasov/how-to-use-pdb-inside-a-docker-container-eeb230de4d11

Mongo:

- https://www.mongodb.com/what-is-mongodb
- https://django-mongodb-engine.readthedocs.io/en/latest/
- https://django-mongodb-engine.readthedocs.io/en/latest/topics/setup.html
- https://github.com/docker-library/docs/tree/master/mongo
- https://docs.mongodb.com/manual/reference/command/
- https://docs.mongodb.com/manual/tutorial/getting-started/
- https://docs.mongodb.com/manual/
- https://docs.mongodb.com/
- https://docs.mongodb.com/manual/tutorial/install-mongodb-on-ubuntu/
- http://docs.mongoengine.org/apireference.html#module-mongoengine.queryset
- https://docs.mongodb.com/manual/tutorial/insert-documents/
- https://docs.mongodb.com/manual/tutorial/query-documents/
- https://docs.mongodb.com/manual/tutorial/update-documents/
- https://docs.mongodb.com/manual/tutorial/remove-documents/
- http://docs.mongoengine.org/apireference.html#module-mongoengine.queryset
- http://docs.mongoengine.org/guide/querying.html
- http://docs.mongoengine.org/guide/index.html
- http://docs.mongoengine.org/guide/document-instances.html#saving-and-deleting-documents
- http://docs.mongoengine.org/guide/defining-documents.html?highlight=listfield
- https://docs.mongodb.com/manual/reference/operator/update/set/
- https://stackoverflow.com/questions/51258463/mongoengine-how-to-update-specific-fields-of-an-existing-document

Python:

- https://www.python.org/
- https://docs.python.org/3.3/library/index.html
- https://docs.python.org/3.3/reference/index.html
- https://www.python.org/about/apps/
- https://www.python.org/about/quotes/
- http://docs.python-requests.org/en/master/user/quickstart/#json-response-content
- https://docs.python.org/3/library/subprocess.html
- https://docs.python.org/3/library/subprocess.html#frequently-used-arguments
- https://docs.python.org/3/library/http.html
- https://www.tutorialspoint.com/python/python_date_time.htm
- https://www.programiz.com/python-programming/datetime/timestamp-datetime
- https://docker-py.readthedocs.io/en/stable/
- https://pypi.org/search/?q=docker
- https://python-3-patterns-idioms-test.readthedocs.io/en/latest/Singleton.html
- https://www.tutorialspoint.com/python_design_patterns/python_design_patterns_singleton.htm
- https://python-3-patterns-idioms-test.readthedocs.io/en/latest/Comprehensions.html
- https://www.geeksforgeeks.org/append-extend-python/
-

Django:

- https://www.djangoproject.com/start/
- https://docs.djangoproject.com/en/2.1/
- https://docs.djangoproject.com/en/2.1/intro/install/
- https://docs.djangoproject.com/en/1.11/intro/tutorial01/
- https://www.djangoproject.com/start/overview/
- https://docs.djangoproject.com/en/2.1/ref/request-response/

- https://docs.djangoproject.com/en/2.1/ref/
- https://hostadvice.com/how-to/how-to-create-a-virtual-environment-for-your-django-projects-using-virtualenv/
- https://docs.djangoproject.com/en/2.2/ref/csrf/
- https://www.django-rest-framework.org/api-guide/requests/
- https://realpython.com/django-redirects/
- https://stackoverflow.com/questions/8018973/how-to-iterate-through-dictionary-in-a-dictionary-in-django-template
- https://stackoverflow.com/questions/11372177/django-template-tag-in-if-block
-

Apache Kafka – A distributed streaming platform:

- https://kafka.apache.org/intro
- https://kafka.apache.org/documentation/
- https://www.confluent.io/what-is-apache-kafka/
- https://kafka.apache.org/

Leshan:

- https://www.eclipse.org/leshan/
- https://github.com/eclipse/leshan
- https://github.com/eclipse/leshan/wiki

Eclipse Hono:

- https://www.eclipse.org/hono/
- https://wiki.eclipse.org/images/7/77/Eclipse-IoT-Days-Grenoble-2018-Hono.pdf
- https://www.eclipse.org/hono/architecture/component-view/component-view/

GitHub:

- https://github.com/
- http://alblue.bandlem.com/2011/11/git-tip-of-week-git-submodules.html
- https://git-scm.com/book/en/v2/Git-Tools-Submodules

- https://www.youtube.com/watch?v=UQvXst5I41I

Robot 3T:

- https://robomongo.org/

StackOverflow:

- https://stackoverflow.com/questions/34402579/mongoengine-remove-string-from-listfield
- https://stackoverflow.com/questions/34445457/404-error-for-bootstrap-min-css-and-bootstrap-min-js
- https://stackoverflow.com/questions/19895894/could-not-parse-the-remainder-0-from-item0-django
- https://stackoverflow.com/questions/38493057/getting-django-for-python-3-started-for-mac-django-admin-not-working
- https://stackoverflow.com/questions/51709525/mongoengine-how-to-append-document-to-listfield
- https://stackoverflow.com/questions/31880227/django-rest-framework-method-put-not-allowed-in-viewset-with-def-update
- https://stackoverflow.com/questions/51258463/mongoengine-how-to-update-specific-fields-of-an-existing-document
- https://stackoverflow.com/questions/38084136/save-reference-field-mongoengine
- https://stackoverflow.com/questions/16849117/html-how-to-do-a-confirmation-popup-to-a-submit-button-and-then-send-the-reque

Otras:

- https://es.wikipedia.org/wiki/Metodolog%C3%ADa_de_desarrollo_de_software
- https://upload.wikimedia.org/wikipedia/commons/thumb/3/37/Software_Development_Spiral.svg/1024px-Software_Development_Spiral.svg.png
- https://upload.wikimedia.org/wikipedia/commons/5/51/Waterfall_model.png

- https://www.w3schools.com/cssref/css3_pr_word-break.asp
- http://www.tagindex.net/css/table/max_width.html
- https://getbootstrap.com/docs/3.3/components/
- https://www.cyberciti.biz/python-tutorials/securely-hash-passwords-in-python/
- https://www.w3schools.com/html/html_form_attributes.asp
- https://www.w3schools.com/bootstrap/bootstrap_buttons.asp
- https://jsonformatter.curiousconcept.com/
- https://www.w3schools.com/howto/howto_css_circles.asp
- https://www.quackit.com/css/css_color_codes.cfm
- https://github.com/eclipse/leshan/wiki/Cluster
- http://robertsrhapsody.blogspot.com/2018/01/eclipse-leshan-rest-apis.html
- https://www.javatips.net/api/leshan-master/leshan-core/src/main/java/org/eclipse/leshan/core/request/ExecuteRequest.java
- http://catarina.udlap.mx/u_dl_a/tales/documentos/lis/fuentes_k_jf/capitulo2.pdf
- http://json.org/
- https://stackoverflow.com/questions/36432630/class-diagram-viewer-application-for-python3-source
- https://www.youtube.com/watch?v=fMRA93R5VZ8

Apéndice A

Manual de Instalación

El manual de instalación será para el entorno en el que se ha desarrollado, **Linux.**

Requerimientos:

- Docker CE - https://docs.docker.com/install/linux/docker-ce/ubuntu/#docker-ee-customers

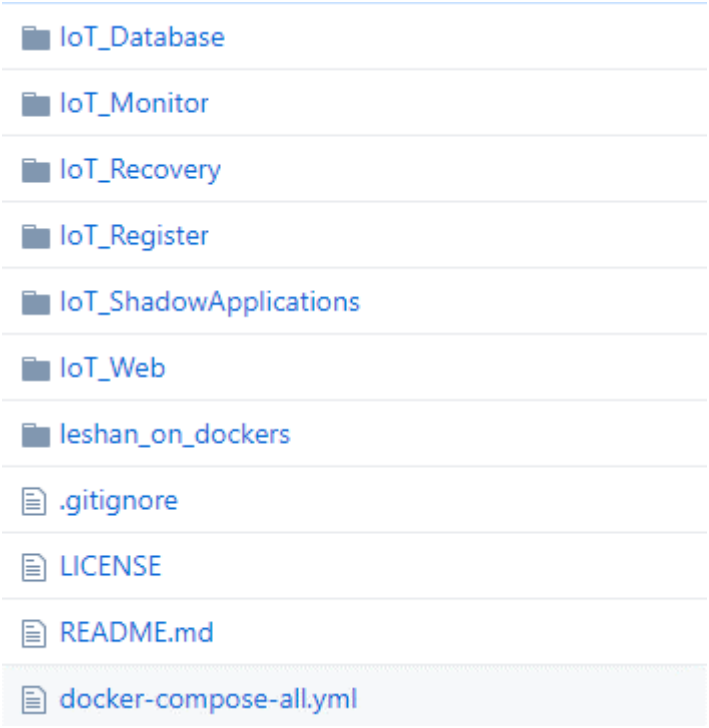


Figura MI.1: Jerarquía del Proyecto

Disponiendo de la jerarquía que se presenta en la Figura MI.1, el primer paso es crear la imagen Docker de cada módulo, por tanto, se accede a la raíz de cada uno de los módulos (donde se puede observar la presencia de un fichero Dockerfile) y ejecutar los siguientes comandos en cada caso:

- docker build --tag=iotdatabase .
- docker build --tag=iotregister .
- docker build --tag=iotweb .
- docker build --tag=iotshadowapp .
- docker build --tag=leshanmonitor .
- docker build --tag=iotrecovery .

*Nota: los nombres de los tags están en concordancia con el fichero **compose**, si se prefieren otros nombres, deben cambiarse en el fichero también.*

Por otra parte, se hacen uso de algunas imágenes oficiales (hace falta descargarlas también) que se pueden obtener de los siguientes enlaces:

- https://hub.docker.com/_/mongo
- https://hub.docker.com/_/zookeeper
- https://hub.docker.com/r/wurstmeister/kafka

Una vez se tienen todas las imágenes, el siguiente paso es desplegarlas sobre Docker Swarm, para ello se abre una consola y se inicializa un nuevo enjambre con el siguiente comando:

- docker swarm init

Una vez inicializado el enjambre, lo siguiente es desplegar las imágenes Docker para que se ejecuten en contenedores. Para ello, mediante la consola hay que ir a la ruta donde se encuentra el fichero **docker-compose-all.yml** y ejecutar el siguiente comando:

- docker stack deploy -c docker-compose-all.yml IOT

Nota: El nombre IOT es muy importante al final del comando por 2 razones:

1. *Docker lo pide como argumento*

2. *Con ese nombre se creará una red llamada **IOT_default**, que es por donde se comunicarán los módulos en el enjambre y es la red a la que se despliegan de forma dinámica las instancias de IoT Monitor.*

Funcionamiento Leshan y ejecución en modo local.

A continuación, se procede a ejecutar un servidor Leshan y a conectar un cliente al servidor. El servidor corresponderá con un servidor de endpoints que se registra en el sistema para realizar sobre él las tareas de monitorización, restauración, etc…

El Cliente Leshan, en la arquitectura del proyecto corresponde con un Endpoint del dispositivo que se monitoriza. Ese Enpoint posee una serie de recursos que pueden ser utilizados o no.

En la Figura MI.2 se ejecuta el servidor Leshan, ofreciendo una interfaz gráfica en la URLhttp://0.0.0.0:8080/ que se visita y permite gestionar los clientes que se registran. Inicialmente no habrá ninguno (Ver Figura MI.3).

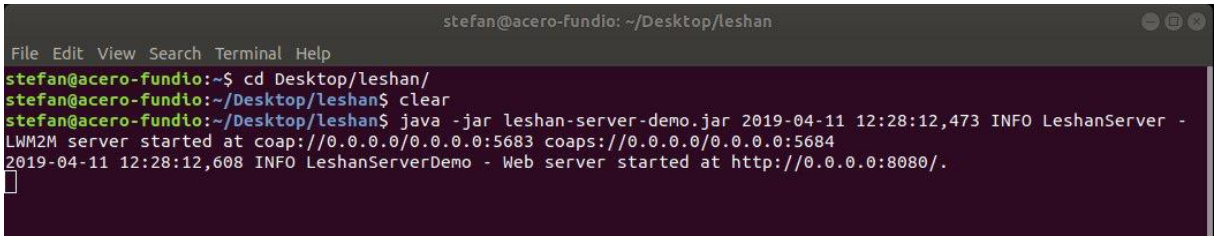


Figura MI.2: Servidor Leshan en ejecución

Figura MI.3: Interfaz gráfica Servidor Leshan

El siguiente paso es ejecutar un cliente de la forma que se puede ver en la Figura MI.4**,** que automáticamente se registrará en el Servidor y aparecerá en la interfaz gráfica (Ver Figura MI.5).

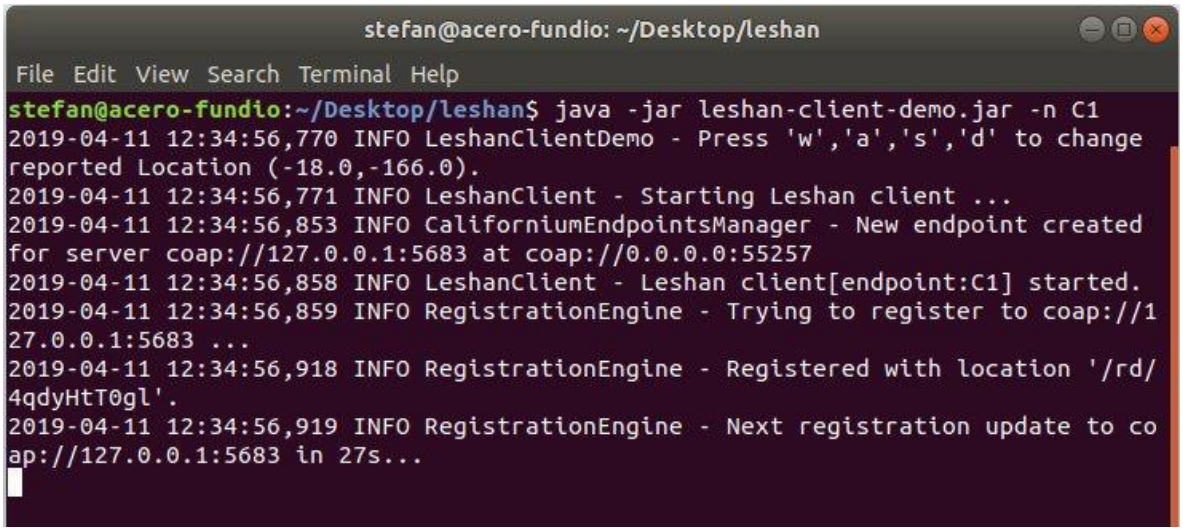


Figura MI.4: Arrancando un Cliente (C1) Leshan

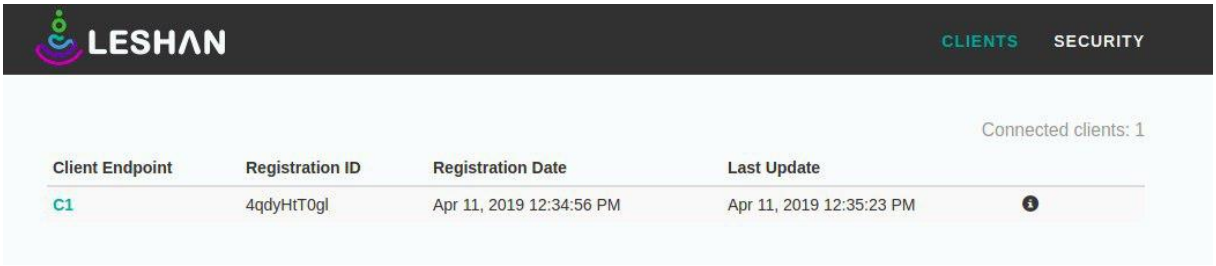


Figura MI.5: Interfaz Gráfica de Leshan actualizada con un Cliente

Esto provoca en Leshan un evento REGISTRATION, que un módulo del sistema (IOT_Monitoring) escuchará y procederá a almacenar los datos nuevos en el sistema y a actualizarlos a medida que reciba nuevos eventos.

Por último, la interfaz de Leshan también permite acceder a un Cliente / Endpoint y ver los diferentes recursos que tiene. Esto se puede apreciar en la Figura MI.6.

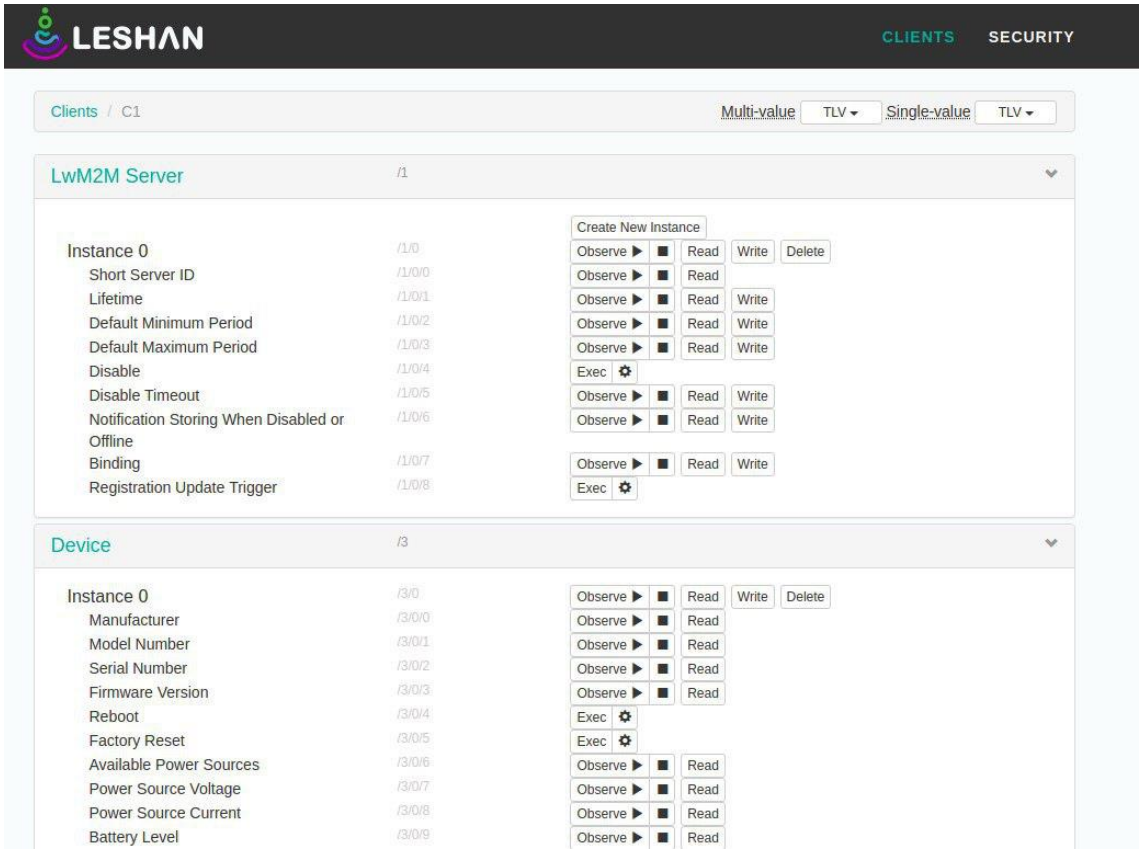


Figura MI.6: Interfaz Gráfica de Leshan: Detalles Cliente

Apéndice B

Manual de Uso

La parte de interacción con el usuario se centra en el módulo **IoT Web** cuya funcionalidad se explica a continuación.

Al acceder la página web, la primera pantalla que aparece es la indicada en la Figura MU.1 que permite elegir entre las opciones registrarse o iniciar sesión.

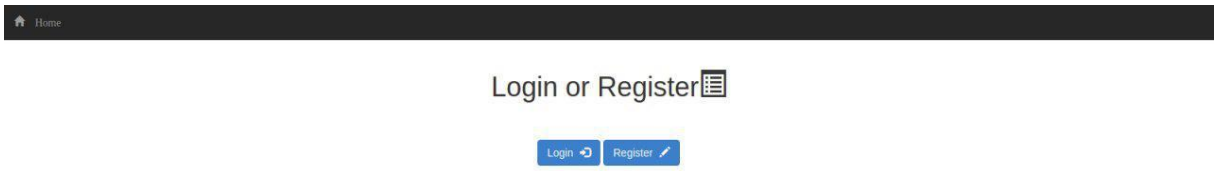


Figura MU.1: Inicio de sesión o Registro

La primera vez que se accede, obviamente hará falta registrarse, por tanto, se selecciona la acción respectiva, lo que lleva a la pantalla de registro indicada en la Figura MU.2.

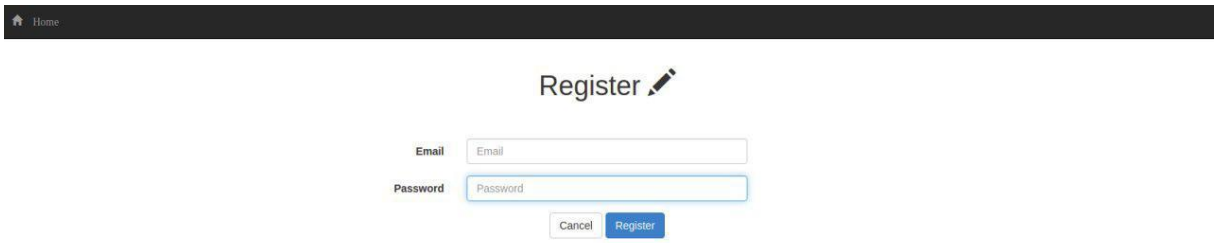


Figura MU.2: Registro

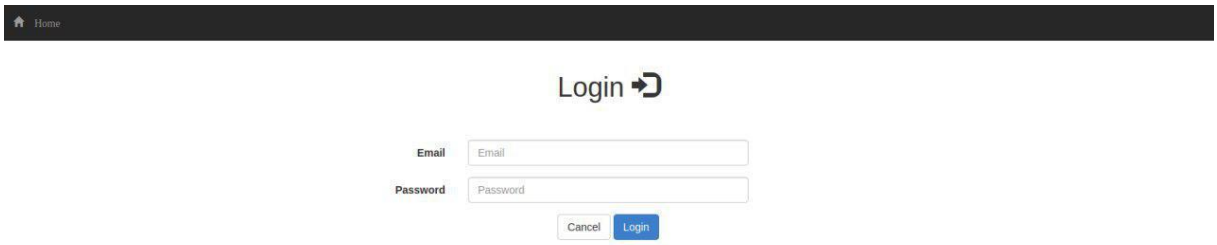


Figura MU.3: Inicio de sesión

Los botones cancelar de las pantallas de inicio de sesión y registro devolverán a la primera pantalla (Figura MU.1).

Una vez realizado el registro o inicio de sesión con éxito, el usuario será redirigido a su página principal (ver Figura MU.4) donde hay un listado de dispositivos Shadows creados por el usuario presente. Inicialmente no hay ninguno, por tanto, se procede a la creación de un dispositivo Shadow (ver Figura MU.5). Para ello se pulsa el botón **Create Shadow Device** que redirige al usuario a un formulario que, tras ser completado y guardado con éxito, redirige al usuario a la página principal en la que se muestra el nuevo dispositivo Shadow creado (ver Figura MU.6).

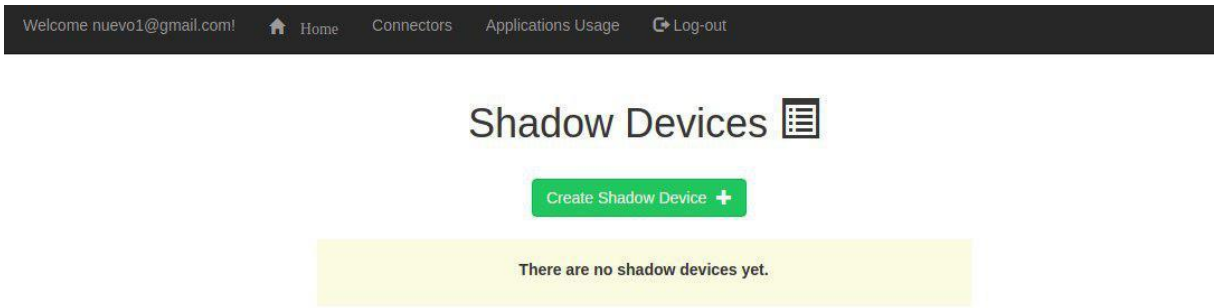


Figura MU.4: Pantalla principal (ningún dispositivo Shadow)

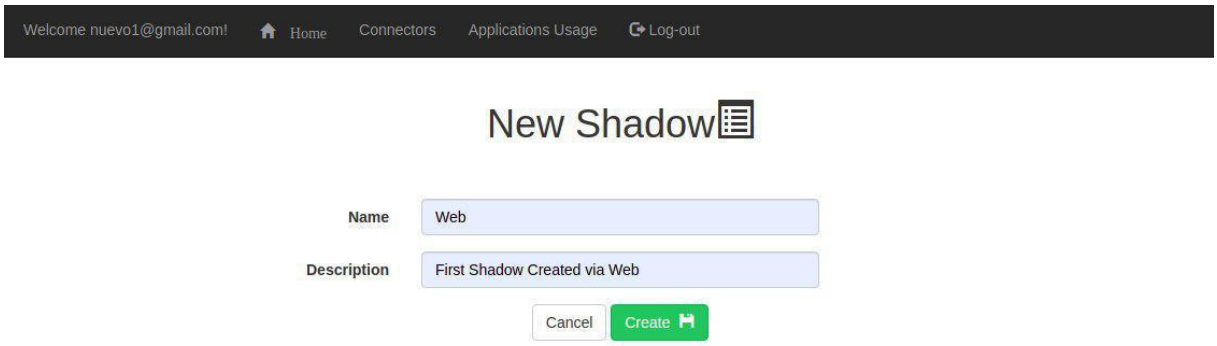


Figura MU.5: Nuevo dispositivo Shadow

Figura MU.6: Pantalla principal (con dispositivos Shadow)

Este paso puede hacerse antes o después de crear un dispositivo Shadow indistintamente, el orden no afectará al resultado. Sin importar cuando se haga, tiene que hacerse obligatorio.

Arriba en la barra si se pulsa sobre el enlace **Connectors**, se redirige a la pantalla que se observa en la Figura MU.7, donde se ven todos los tipos de monitores creados, inicialmente ninguno, por tanto, se procede a la creación de un conector.

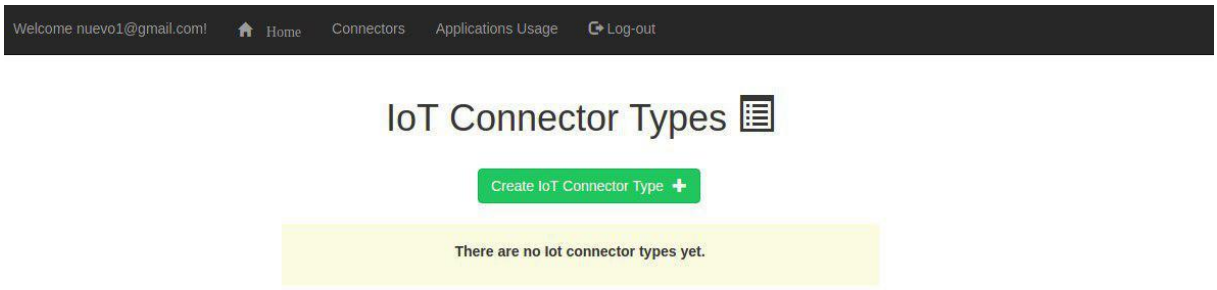


Figura MU.7: Vista tipos de conectores (sin conector registrado)

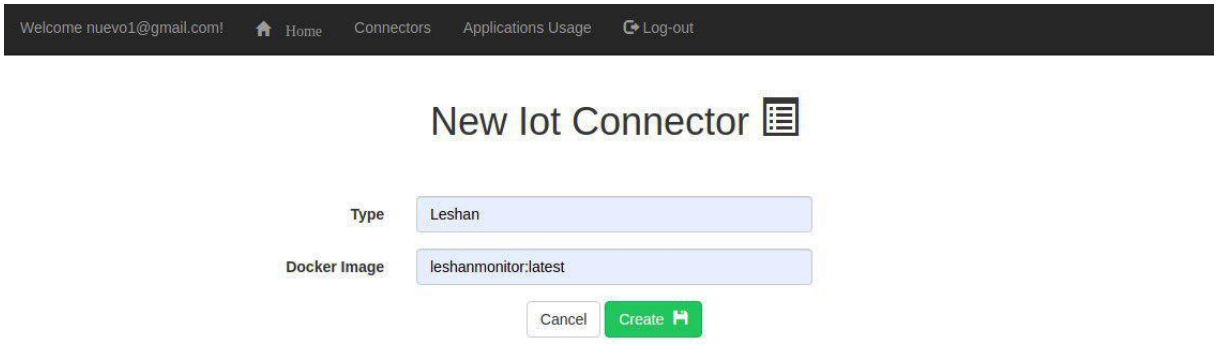


Figura MU.8: Vista creación tipo de conector

Nota: Si se añade un tipo que no está implementado, no funcionará el despliegue dinámico de un monitor.

Nota: Asegurarse de tener la imagen de Docker creada con el nombre que se registre en la web, sino el despliegue dinámico del monitor no funcionará, no se desplegará nada.

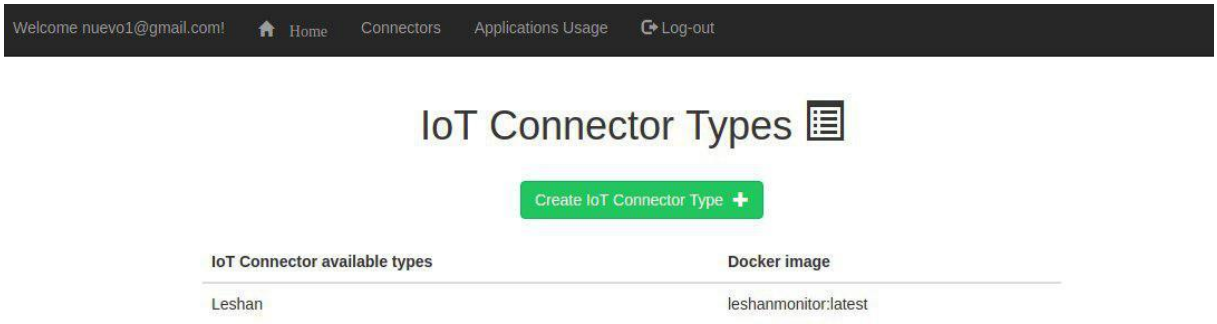


Figura MU.9: Vista tipos de conectores (con conector válido registrado)

Al accionar el botón **Tokens** de un dispositivo Shadow, la página será redirigida a la vista de la Figura MU.10, inicialmente sin ningún token creado. Al accionar el botón **Create token** y tras confirmar la acción, un nuevo token se generará y la vista será actualizada (ver Figura MU.11). Estos tokens que se crean en esta vista se utilizarán para registrar un dispositivo físico en el sistema.

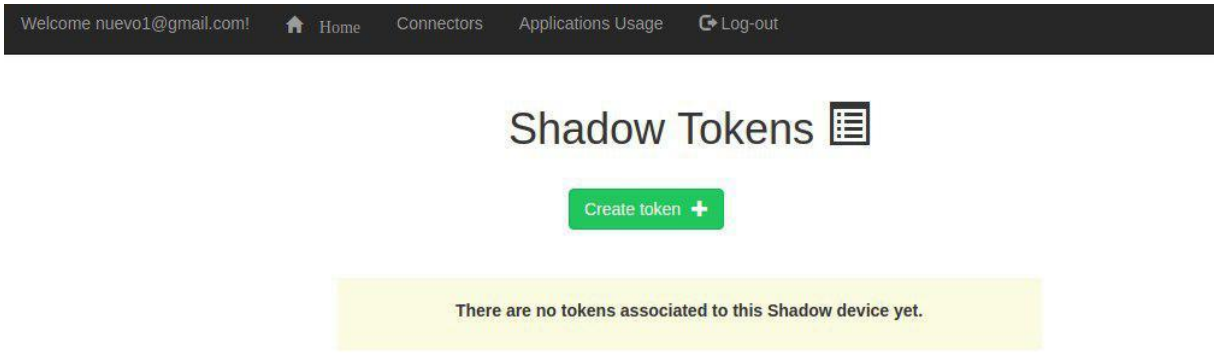


Figura MU.10: Vista tokens asociados a un Shadow (sin tokens)

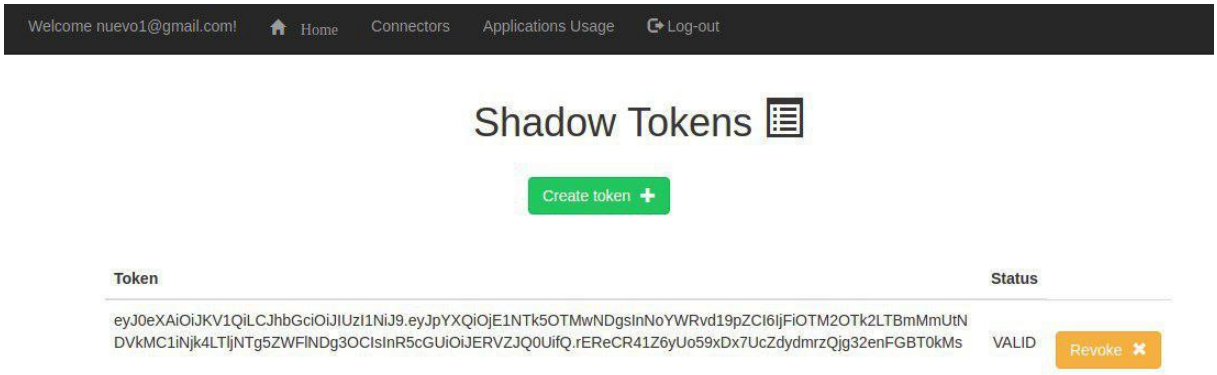


Figura MU.11: Vista tokens asociados a un Shadow (con un token creado)

Llegados a este punto, se puede registrar en el sistema un dispositivo físico en el sistema para ser monitorizado y para ofrecer sus recursos a posibles aplicaciones finales.

Al pulsar el botón **Devices** de un dispositivo Shadow, se mostrará una página (ver Figura MU.12 y Figura MU.13) con todos los dispositivos físicos registrados en el sistema

pertenecientes al dispositivo Shadow respectivo. Inicialmente no habrá ningún dispositivo físico (o dispositivo IoT) registrado, como se puede observar.

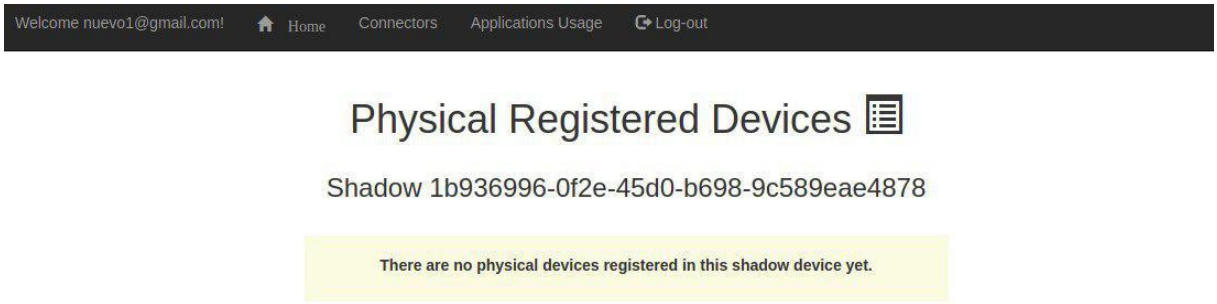


Figura MU.12: Vista dispositivos físicos registrados y asociados a un Shadow (sin dispositivo físico)

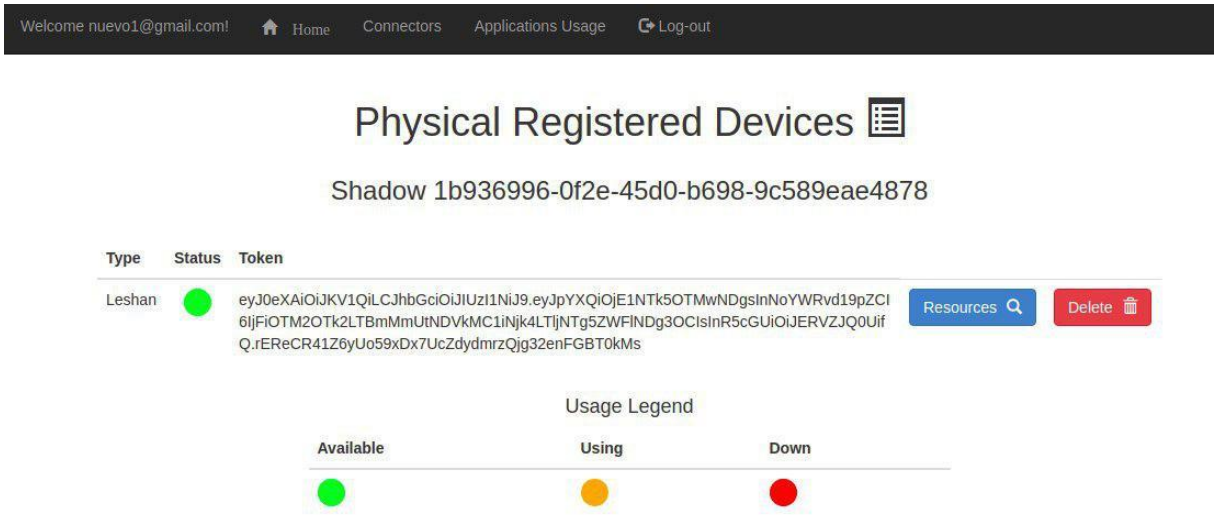


Figura MU.13: Vista dispositivos físicos registrados y asociados a un Shadow

Una vez registrado un dispositivo, aparecerá en una tabla y se permitirá ver sus recursos y también borrarlo del sistema. Al pulsar el botón **Delete** y tras confirmar la acción, el dispositivo físico será borrado y se actualizará la vista.

Al consultar los recursos de un dispositivo físico, la página se redirige a la vista de la Figura MU.14 donde se puede apreciar información sobre los recursos, los puntos

finales a los que pertenecen y el estado de uso de los recursos, junto con una leyenda para el estado de uso.

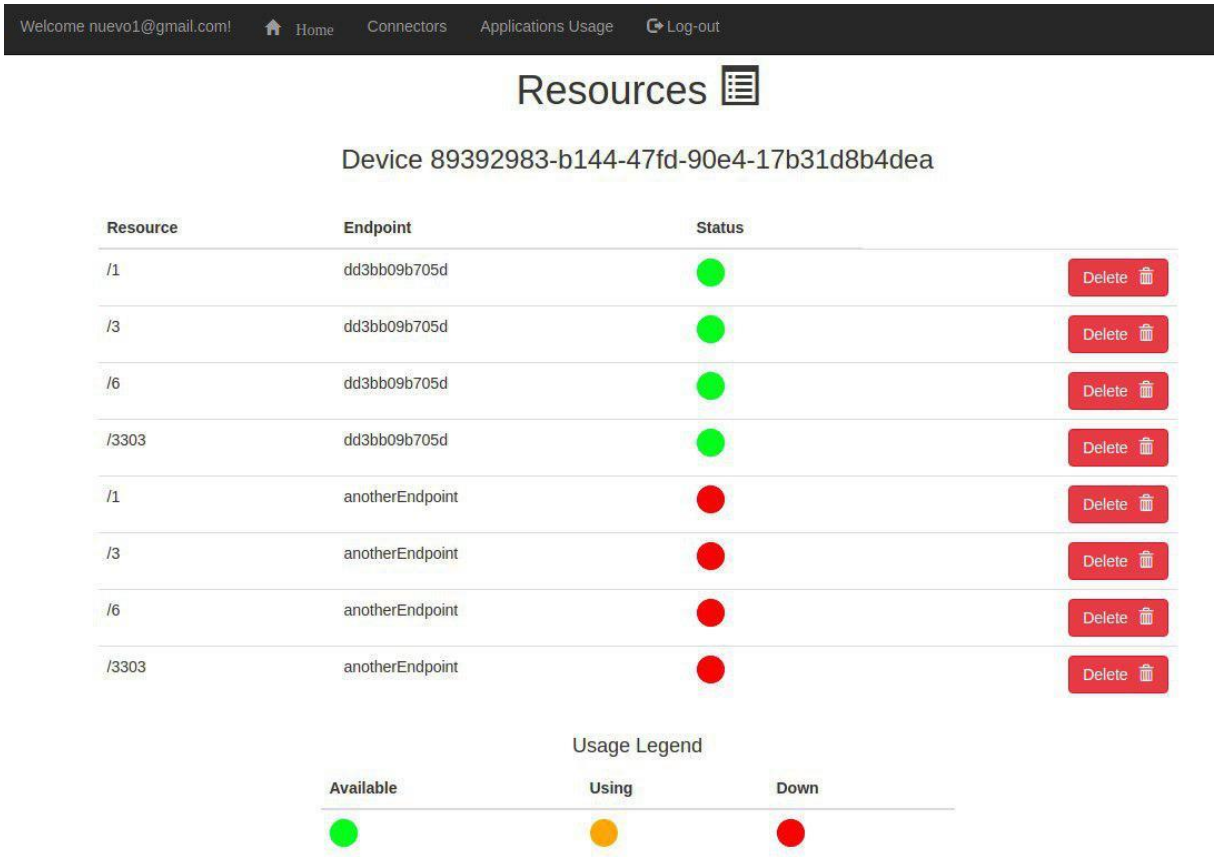


Figura MU.14: Vista recursos asociados a un dispositivo físico con leyenda de uso

También se pueden ver todos los recursos de un dispositivo Shadow, pulsando el botón **Resources**. Se muestra una página con los tipos de recursos que hay y la cantidad de cada recurso. Ver Figura MU.15.

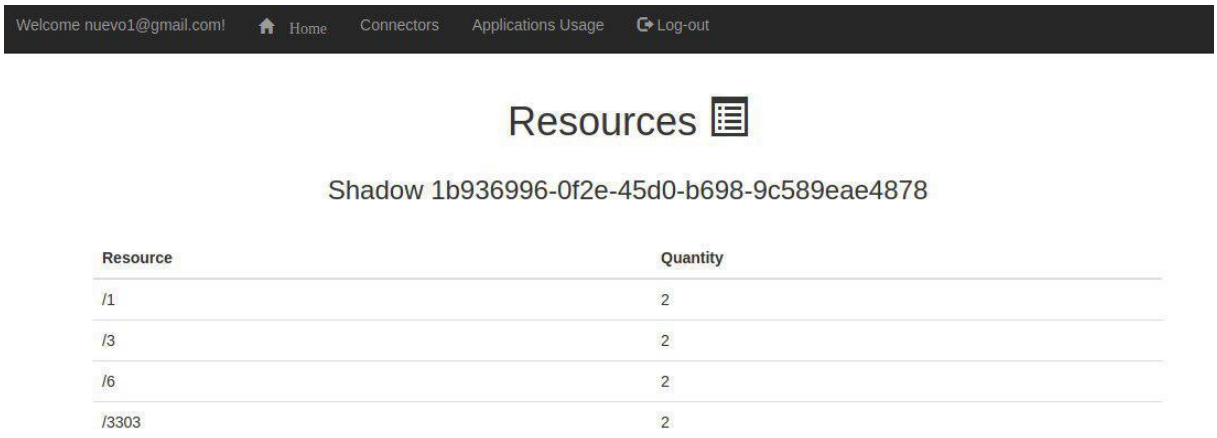


Figura MU.15: Vista con todos los recursos (disponibles y no disponibles) de un dispositivo Shadow y sus cantidades respectivas**.**

Al pulsar en la barra del menú el enlace **Applications Usage** se podrá ver una tabla con todas las aplicaciones del sistema que hayan mostrado al menos interés en algún recurso, sin haber llegado a usarlo. Se puede ver información relativa al interés como el tipo de recurso por el que se ha interesado, la respuesta del sistema (si se ha encontrado o no), tipo de operación que la aplicación deseará realizar sobre dicho recurso, etc. Ver Figura MU.16 y Figura MU.17.

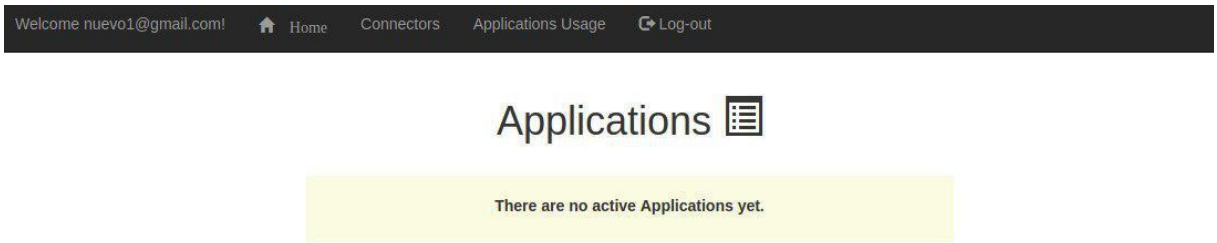


Figura MU.16: Vista aplicaciones del sistema (sin aplicaciones)

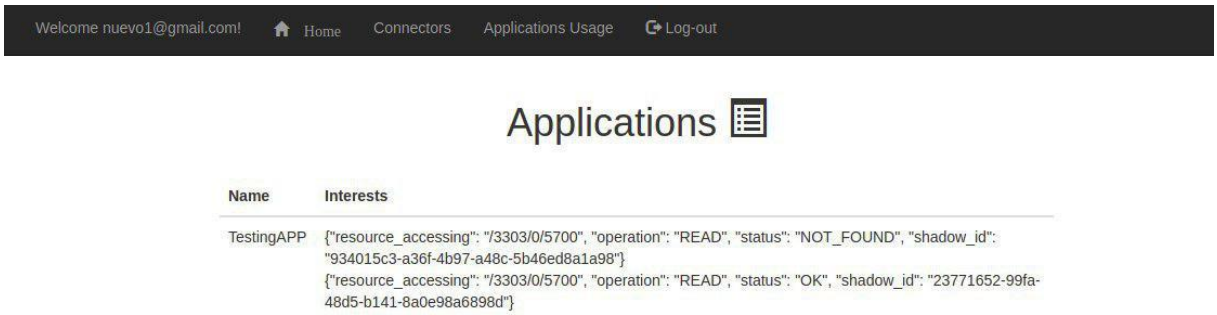


Figura MU.17: Vista aplicaciones del sistema con sus intereses respectivos mostrados en algún recurso (existente o no)