ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
GRADO EN INGENIERÍA INFORMÁTICA

DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA INTERMEDIARIO PARA INTERNET DE LAS COSAS

MIDDLEWARE DESIGN AND IMPLEMENTATION FOR THE INTERNET OF THINGS

Realizado por

Antonio García Jiménez

Tutorizado por

Mercedes Amor Pinilla

Departamento

Lenguajes y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA
MÁLAGA, JUNIO 2016

Fecha defensa:

El Secretario del Tribunal

Resumen: En esta memoria se recoge el proceso y los conocimientos necesarios para el diseño y la implementación de un sistema capaz de estar a la altura de un nuevo concepto llamado ‘Internet de las Cosas’ que pretende, mediante la conexión de objetos de nuestro día a día a internet, conseguir mejorar la calidad de vida de las personas y solucionar problemas que hasta ahora no habían sido abordados.

Este sistema, implementado en Node.js, permitirá recoger información obtenida mediante los sensores de diversos dispositivos para ser almacenados en una base de datos no relacional como lo es MongoDB. Se proporcionará para ello una API basada en el protocolo HTTP que permita a los usuarios y dispositivos un acceso sencillo y rápido.

Con este trabajo se pretende que se conecten multitud de dispositivos y que con toda la información recopilada puedan diseñarse aplicaciones de más alto nivel que utilicen estos datos para realizar tareas complejas como análisis de Big Data o respuestas automatizadas ante ciertos eventos.

La plataforma diseñada será escalable, para contar con un mayor tráfico, y extensible, para poder adaptarla en un futuro a las nuevas necesidades o tecnologías que pudieran surgir.

Palabras claves: Node.js, Internet de las Cosas, HTTP, MongoDB

Abstract: On this project, the process and knowledge needed for the design and implementation of a middleware able to live up to a new concept called ‘Internet of Things’ is addressed. This new concept, by connecting everyday objects to the Internet, aims at improving people’s life and solving some problems that have not been solved to this day.

This system, implemented in Node.js allows us to collect information obtained by various devices’ sensors and store them on a non relational database like MongoDB. It will provide an API for this purpose based on the HTTP protocol that will allow users and devices easy and quick access.

This project intent is to connect a variety of devices and use all that collected information to design higher-level applications which could use this data to perform complex tasks such as Big-Data analysis or automated responses to certain events.

This platform aims to be scalable in order to support higher traffic and extensible in order to adapt in the future to new needs or technologies that might emerge.

Keywords: Node.js, Internet of Things, HTTP, MongoDB

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

En este primer capítulo vamos a abordar las ideas y el contexto en el que se desarrolla este Trabajo Fin de Grado, poniendo el acento en la motivación y los objetivos que han impulsado este proyecto. Por último, se analizará brevemente la metodología de trabajo utilizada así como la estructura de la memoria.

1.1. Motivación

Estos últimos años hemos estado viendo un especial crecimiento en el ámbito del Internet de las Cosas, principalmente impulsado por el abaratamiento de las tecnologías requeridas para su funcionamiento, pero también por un desarrollo menos visible de otras tecnologías como pueden ser el Big Data o los nuevos estándares en telecomunicación. Por eso nos encontramos en un momento excepcional para avanzar en el desarrollo de estos sistemas.

Debemos tener en cuenta que esta tecnología no es nueva, ya que las primeras menciones datan del artículo de 2009 de Kevin Ashton, y ya entonces vio el potencial para cambiar el mundo, aunque no fue el primero, podemos datar las menciones de sistemas de este tipo, aunque con otros nombres, hasta la década de los 90. De este modo, teniendo en cuenta de que se trata de un concepto no tan moderno, podemos ver que ya ha habido algunas tecnologías que han intentado ocupar el espacio generado.[1]

Una de las principales motivaciones para este Trabajo es la falta de iniciativa para aprovechar tecnología libre y gratuita que ya se encuentra a disposición para realizar una solución que pueda satisfacer a la mayor parte de los usuarios que pretendan desarrollar sistemas de este tipo.

1.2. Objetivos

La motivación detrás de este trabajo nos lleva a un objetivo fundamental, la creación de un sistema que resuelva esta problemática. Debe tratarse de un diseño escalable, ligero y

que permita el mayor desarrollo horizontal intentando desprenderse de los dispositivos que proveen la información y también alejarse de las posibles implementaciones de aplicaciones que utilicen estos datos.

En un primer análisis, que será ampliado más adelante en la memoria, podemos encontrar diversas tecnologías que podrían permitir cumplir estos objetivos generales y para esto existen infinidad de lenguajes y de *frameworks* que nos permiten desarrollar un proyecto que cumpla nuestros objetivos, como Vert.X, Django, Node.js o JavaEE, entre otros.

Este trabajo debe contar con un sistema de aprovisionamiento de datos que sea lo más sencillo posible, permitiendo que despliegues de dispositivos de cualquier tipo puedan ser identificados dentro de nuestra estructura. Por otro lado contamos con un servicio de almacenamiento de toda esta información que debe ser fiable y rápido, además de ser muy escalable.

Para ello, se fijarán los siguientes objetivos:

- Analizar a fondo las tecnologías existentes para el desarrollo de este tipo de sistemas.
- Estudiar los diversos casos de uso que vamos a encontrar y que cubriremos en nuestro diseño.
- Diseñar e implementar los requisitos derivados de estos estudios.
- Desarrollar una pequeña prueba que permita mostrar una aplicación práctica.

1.3. Metodología

Debido a que se trata de un sistema que cuenta con unos objetivos bastante amplios, se iniciará el trabajo con un análisis exhaustivo de la tecnología actual y de las diferentes opciones con las que contamos, así como de las tecnologías que tendrán algún tipo de interacción con la plataforma para poder desarrollar un *middleware* que se ajuste a la actualidad. Una vez elegidas y justificadas las tecnologías que se utilizarán se pasará a un desarrollo que se irá complementando con pruebas para asegurar que su funcionalidad y los requisitos se ajustan a lo que se busca en el trabajo.

1.4. Estructura de la memoria

Esta memoria cuenta con 6 capítulos, un anexo con un manual de uso y despliegue y una lista de referencias bibliográficas. A continuación se muestra una breve introducción para cada capítulo:

- **Capítulo 1. Introducción.** Se trata de un capítulo dedicado a realizar una breve descripción de los objetivos y las motivaciones que han impulsado este trabajo.
- **Capítulo 2. Estado del Arte.** Es necesario para el desarrollo de este trabajo un análisis de la situación actual en este campo, haciendo hincapié en las nuevas tecnologías que están haciendo posible el avance en esta materia.

- **Capítulo 3. Especificación y diseño.** Este capítulo es clave en esta memoria ya que se explica el diseño que fundamenta el desarrollo y despliegue de la plataforma así como las características y funcionalidades que se han decidido incluir. Aquí encontraremos un esquema de toda la arquitectura del *middleware*.
- **Capítulo 4. Análisis de las tecnologías.** Una vez analizado el Estado del Arte y diseñada la plataforma, se pasará a un estudio más a fondo de las opciones destacadas que se encuentran en línea con los objetivos de este trabajo.
- **Capítulo 5. Implementación de la plataforma.** En el Capítulo 5 se describe la implementación y las pruebas de la plataforma. Se mostrará una serie de piezas de código en los que se observe con detalle las partes más importantes del desarrollo.
- **Capítulo 6. Conclusión y líneas futuras.** En este último capítulo se pretende realizar una conclusión final sobre la plataforma desarrollada y el análisis realizado además de una serie de líneas que serían interesantes para el futuro.
- **Apéndice A. Manual.** Por último, se incluye un pequeño manual de despliegue y uso que tiene como objetivo facilitar la réplica de este trabajo y su puesta en funcionamiento.

Capítulo 2

Estado del Arte

Para el segundo capítulo, es conveniente realizar un estudio del Estado del Arte que nos permitirá poner en contexto a este trabajo. Comenzaremos con una mirada en general al Internet de las Cosas para luego enfocar en los dispositivos que se encuentran en el mercado y las aplicaciones que utilizan este concepto.

2.1. El Internet de las Cosas

Como ya hemos mencionado con anterioridad, no se trata de un concepto nuevo, pero que realmente no ha sido ampliamente desarrollado hasta los últimos años. Las tecnologías que están potenciando y sustentando este avance se pueden dividir en dos grupos. Por un lado, el fuerte avance en campo de las telecomunicaciones y por otro, el abaratamiento y la llamada *democratización* de los dispositivos que conforman las *cosas* de este Internet de las Cosas. Existen diversas empresas de gran calado que han apostado por la investigación y el desarrollo en este campo, entre las que se encuentran:

- **Ericsson.** Una de las empresas pioneras en el ámbito de las telecomunicaciones y con una presencia a nivel mundial en el mercado tecnológico, tiene una fuerte apuesta por el Internet de las Cosas, financiando proyectos de investigación y con alianzas con otras empresas, como Volvo o Cisco, para realizar aplicaciones más cerca de las necesidades de la sociedad.[2]

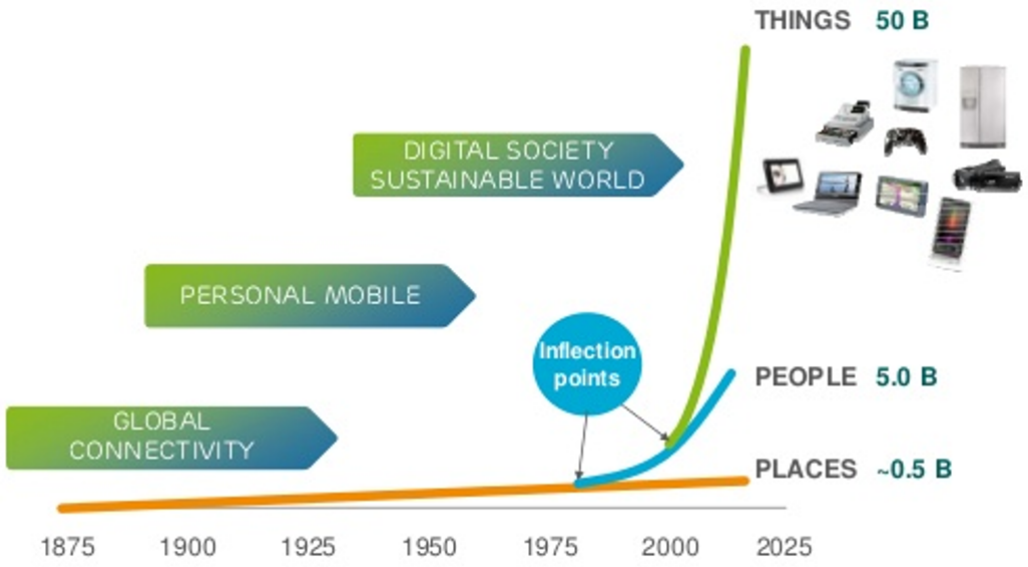

Figura 2.1: La predicción que realiza Ericsson es que antes de 2025 contaremos con 50 mil millones de dispositivos conectados.

- **Intel.** Otro gran jugador en el campo del Internet de las Cosas es el gigante Intel. Podemos encontrar un avance en el desarrollo de tecnologías que faciliten el despliegue en sectores como la industria de la energía o de la automoción. Intel cuenta además con su propio dispositivo preparado para el Internet de las Cosas, se trata del Intel Gallileo o el Intel Edison que analizaremos más adelante.[3]

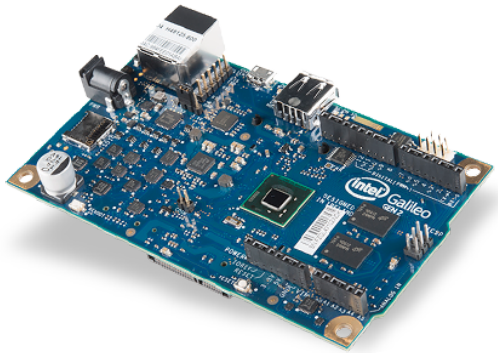

Figura 2.2: La placa Galileo desarrollada por Intel.

- **Google.** Debemos destacar tambien el trabajo de Google que ha potenciado a diversas empresas de este campo, además cuenta con una fuerte apuesta para el Internet de las Cosas dentro de su Plataforma como Servicio llamada Google Cloud Platform. [4]

2.2. Telecomunicaciones: LTE, LTE-A y WiMax

Se trata de un sector no tan visible pero crucial para el desarrollo del Internet de las Cosas. Sin un sistema de telecomunicaciones que nos permita desplegar nuestros sistemas de forma barata y con un bajo coste energético, no sería posible desarrollar el Internet de las cosas por lo que se puede considerar como un pilar fundamental.

Ciertos estudios [5] nos muestran que el crecimiento en el número de dispositivos conectados a la red está creciendo a razón de un 30 % más cada año. Podemos observar que los estándares que se están implementando y los planes de futuro para éstos cuentan con una orientación hacia la masificación de los dispositivos, potenciado principalmente por el abaratamiento de los dispositivos móviles pero también por el avance en el Internet de las Cosas.

En general las diferentes tecnologías que se están desarrollando en este campo, vienen a cubrir un espacio que hasta entonces no había estado poblado entre las tecnologías Wi-Fi y las móviles como GSM como podemos ver en el gráfico 2.3.

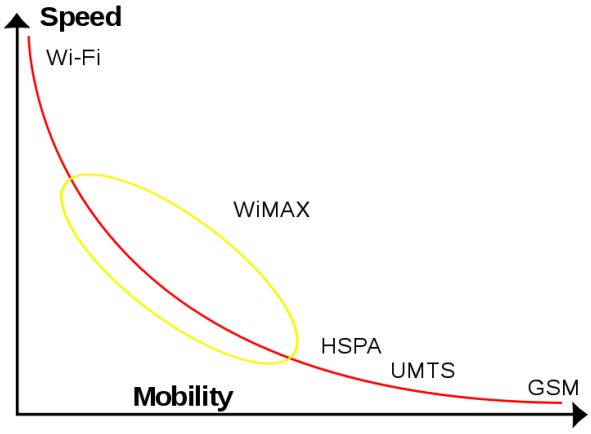

Figura 2.3: Diversas tecnologías inalámbricas comparadas en términos de velocidad y movilidad.

Dentro de estos estándares podemos destacar:

- **Long Term Evolution (LTE).** Diseñado por un consorcio de las organizaciones que controlan el sector de las telecomunicaciones a nivel supranacional, el 3GPP (siglas por su nombre en inglés, "3rd Generation Partnership Project") que se encarga, entre otros asuntos, del desarrollo y mantenimiento de los estándares de tecnologías de telecomunicaciones que son adoptadas por el mercado. La tecnología LTE cuenta con un diseño basado en anteriores tecnologías como GSM o UMTS y ofrece una capacidad y velocidad mejorada. Su implantación comenzó en 2010 y a día de hoy su cobertura se extiende por la inmensa mayoría de los países con una buena penetración como podemos observar en esta tabla 2.1 extraída de un estudio realizado en 2015. [6]

Posición	País	Penetración
1	Korea del Sur	97 %
2	Japón	90 %
3	Italia	90 %
4	Kuwait	86 %

Cuadro 2.1: Los países con mayor penetración de LTE.

- **Long Term Evolution Advanced (LTE-A).** Se trata de la evolución de la tecnología LTE y es la primera en cumplir realmente con el estándar de 4G propuesto por el 3GPP aunque tecnologías anteriores ya habían sido etiquetadas como 4G. Cuenta con un ancho de banda aún mayor que su predecesor y una de sus mayores novedades es el uso de una arquitectura basada completamente en paquetes IP. [7]

- **Worldwide Interoperability for Microwave Access (WiMax).** Creado inicialmente como una alternativa a las líneas cableadas de internet utilizando tecnología inalambrica para ello. Actualmente cuenta con uso medianamente extendido y su última versión, WirelessMAN-Advanced, cumple el estándar de 3GPP para 4G por lo que se establece como un competidor directo de LTE-A.[8]

2.3. Dispositivos: Android, Arduino y otros

La otra cara de la moneda la conforman los dispositivos que estructuran el Internet de las Cosas. Con ellos realizaremos las tareas de recopilación de datos o de información sobre el entorno y los usuarios, pero también serán usados para llevar a cabo acciones automatizadas en respuesta a esta información. Se trata de un punto crítico debido a las características de cada una de las plataformas existentes ya que debemos tener en cuenta asuntos como el consumo de energía y la potencia de procesamiento, así como la conectividad con los diversos sensores, para encontrar el mejor equilibrio que se ajuste a la situación.

En la actualidad, el mercado de dispositivos se encuentra en pleno crecimiento gracias a la apuesta de gigantes de la industria de la tecnología y la incursión de nuevos jugadores provenientes de países como China o India. Esta situación ofrece un abaratamiento importante de los precios así como una competencia que impulsa la tecnología.

Podemos destacar como líderes en el mercado las siguientes plataformas:

- **Android.** Inicialmente diseñado por una pequeña empresa que sería comprada en 2005 por Google. Actualmente se trata del sistema operativo móvil con mayor penetración en el mundo debido a que es distribuido por la compañía como software libre. Es fabricado por muchos fabricantes en todo el mundo por lo que cuenta con una amplia gama de diseños y aplicaciones. No solo un sistema operativo para dispositivos de telefonía móvil sino que está presente en muchos otros dispositivos como televisiones o tabletas. Su impresionante extensión y bajo coste, así como una

gran facilidad para realizar aplicaciones, hacen de esta plataforma una elección muy interesante. [9]

- **Arduino.** Diseñado desde el principio como dispositivo con software y hardware libres, ofrece una plataforma muy interesante que cuenta con una comunidad muy extendida y además es un buen sistema para iniciarse en el mundo de los sistemas empotrados. Su estrategia de diseño libre le permite además que distintos fabricantes puedan vender los dispositivos a precios muy asequibles. Se programan a bajo nivel y cuentan con una amplia gama de sensores y ampliaciones que van desde sensores de movimiento o contaminación hasta extensiones que permiten acceso a internet o a la red móvil. Añadir imagen de un arduino.[10]

- **Intel, Libellium y BQ.** Otros grandes fabricantes tienen sus propias apuestas preparadas para el Internet de las Cosas. Es el caso de Intel con su dispositivo Edison, Libellium con su gama de productos preparada para crear un ecosistema para desarrolladores y BQ, fabricante español, que también se ha unido a la tendencia con su propio dispositivo llamado BQ Zum.

2.4. Aplicaciones existentes

En esta sección analizaremos algunas aplicaciones que a día de hoy se aprovechan del Internet de las Cosas. De esta manera, se analizará el estado del despliegue y la amplitud que han alcanzado las aplicaciones más conocidas para poder tener una visión más global de todo el ecosistema, sobretodo desde un punto de vista más económico y práctico.

- **Aparcamiento.** El aparcamiento se ha convertido en un problema crónico en las grandes ciudades, llegando a un punto en el que la tecnología puede ser extremadamente beneficiosa para mejorar la vida de los usuarios. Es el caso de la empresa SmartSantander que ha conseguido desarrollar e implantar en la ciudad de Santander un sistema de aparcamiento inteligente mediante sensores colocados en las plazas de aparcamiento para conseguir que los usuarios encuentren una plaza de manera fácil y rápida. En la imagen 2.4 podemos ver como se despliega este sistema.[11]

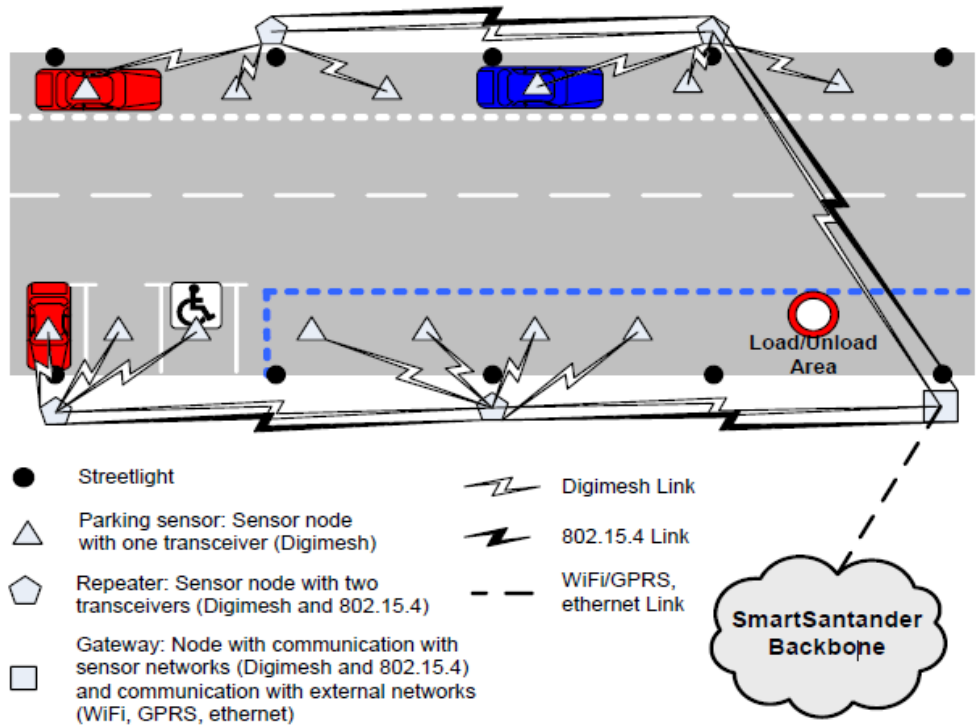

Figura 2.4: Ejemplo de uso de un sistema de SmartSantander.

- **Agricultura.** La agricultura también es un área en el que el desarrollo del Internet de las cosas ha llevado grandes avances, consiguiendo sistemas que permitan un ahorro de recursos y una mejora en la eficiencia de los cultivos. Como ejemplo cercano de esta nueva agricultura tenemos el sistema "SIEGA" que se encuentra actualmente desplegado en viñedos de Galicia ofreciendo parámetros ambientales como humedad o temperatura. Estos datos son recogidos como se puede ver en la imagen 2.5 para luego ser procesados para realizar predicciones y/o tomar acciones acordes a la situación en tiempo real del viñedo.[13]

Figura 2.5: Esquema de la arquitectura utilizada por el sistema SIEGA para viñedos.

- **Contaminación.** Con el crecimiento de las grandes ciudades, la contaminación medioambiental se ha convertido en un problema crónico. Algunas ciudades han optado por cerrar el acceso a la mayor parte del tráfico o por reducirlo, pero para cualquier solución que busque ser eficiente, un despliegue de una red de sensores ayudará a obtener información en tiempo real y con mucha más precisión, con el objetivo de adelantarse a los picos de contaminación mediante software predictivo o la rápida respuesta ante problemas no previstos. En la ciudad de Londres, una pequeña empresa ha desarrollado un sistema que utiliza palomas equipadas con un pequeño dispositivo para realizar su propio despliegue de sensores de contaminación.[14]

2.5. Conclusión del capítulo

Se puede observar que en los últimos años, las grandes empresas han dejado el terreno preparado para que los pequeños empresarios puedan desarrollar sus propias aplicaciones del Internet de las Cosas. Pero, aunque el desarrollo tecnológico podríamos observar que ya ha llegado, las aplicaciones que se han implementado no se encuentran al alcance de todos. En la mayoría de los casos se trata de aplicaciones "verticales" que se desarrollan para una situación en concreto y no con miras a un despliegue más global. Podemos concluir que el verdadero desarrollo del Internet de las Cosas llegará cuando las aplicaciones "horizontales" comiencen a implementarse y a llegar a todos los usuarios.

Capítulo 3

Especificación y diseño

Gracias a la visión que hemos obtenido en el estudio del Estado del Arte, podemos comenzar con el diseño y análisis de los requisitos para nuestra plataforma. Revisaremos los objetivos, para pasar por un proceso de diseño de la arquitectura que se adapte de la mejor forma posible. En este capítulo se intentará no profundizar en la tecnología a utilizar ya que así conseguiremos un diseño que sea independiente de la implementación. [15]

3.1. Objetivos

Es muy importante tener en cuenta los objetivos que nos hemos propuesto inicialmente para poder definir el alcance del sistema. Los revisaremos y profundizaremos en cada uno de ellos para luego pasar a definir algunos casos de uso.

- Diseño e implementación de un middleware con los servicios necesarios para la recolección, almacenamiento y provisión de datos de dispositivos del Internet de las Cosas. Se trata de nuestro principal objetivo y como tal merece la mayor atención. Debemos tener en cuenta que los datos contarán con un una jerarquía a la que pertenecen y que para ello se deberá poder almacenar información sobre, al menos, los usuarios y los dispositivos a los que pertenecen los datos recibidos. Para ello, lo mejor es diseñar el sistema de modo que los usuarios tengan acceso a dicha jerarquía y puedan modificarla a su conveniencia.

- Implementación de una página web que permita mostrar y comprobar el correcto funcionamiento del middleware. Para comprobar el correcto funcionamiento del sistema, vamos a desarrollar una rápida página web que utilice nuestro servicio de middleware para mostrar información. Se complementará con una pequeña rutina que simulará el comportamiento de un despliegue de dispositivos, mostrando la información recopilada de manera ordenada. Como se trata de un objetivo secundario,

Req.	Descripción	Prioridad
RF1	**Acceso CRUD a la información de los usuarios:**El usuario debe tener acceso a esta información ya que es básica para la configuración por parte del usuario de su propio despliegue.	Alta
RF2	**Acceso CRUD a la información de los dispositivos:**Igualmente, el usuario debe poder tener un acceso completo a sus dispositivos, para poder realizar las tareas necesarias de administración.	Alta
RF3	**Envío de datos desde un dispositivo:** La recopilación de datos es básica en nuestro sistema y por ello el usuario debe contar con un método sencillo de envío de datos.	Alta
RF4	**Lectura del último dato de un dispositivo:** Debido a que en muchos casos los datos se recibirán de manera periódica, el cliente que desee estar al tanto de los datos puede pedir el último como una primera forma de acceso a los datos.	Alta
RF5	**Petición de datos de un dispositivo con filtro:** Para que el usuario pueda recabar la información que necesite, se debe ofrecer un método de peticiones de datos mediante el uso de filtros.	Media

Cuadro 3.1: Requisitos funcionales de nuestro sistema.

no contará con un análisis tan a fondo de sus requisitos y se explicará su desarrollo en el capítulo de implementación.

3.2. Requisitos

Analizando estos objetivos, podemos proceder a listar los requisitos funcionales más importantes de nuestra aplicación, que serán una guía excelente para el posterior desarrollo. En la tabla 3.1 se muestran los requisitos, teniendo en cuenta su prioridad.

3.3. Casos de Uso

Para visualizar las diferentes situaciones con las que vamos a encontrarnos vamos a diseñar algunos casos de uso que nos serán útiles para la identificación de los requisitos funcionales y para el desarrollo de la plataforma. Vamos a dividir los casos de uso en dos apartados: acceso a los datos y acceso a la jerarquía. No habrá una diferencia tan grande en el acceso a las distintas partes desde el punto de vista del desarrollo pero nos facilitará el diseño de los casos de uso.

3.3.1. Acceso a la jerarquía

Se trata de un caso de uso básico en el que el usuario accede para leer, crear o modificar los datos sobre los usuarios del sistema o los dispositivos. Aunque no está incluido en este

Trabajo, debe tenerse en cuenta que esto debe protegerse ante amenazas externas y que es este caso de uso donde la mayor parte de la seguridad debe residir.

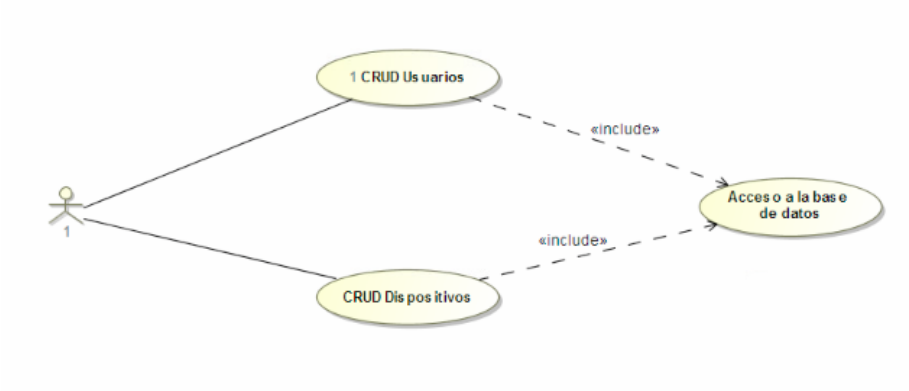

Figura 3.1: Diagrama de caso de uso para el acceso a la jerarquía.

3.3.2. Acceso a los datos

En el acceso a los datos no habrá tanta libertad como en la jerarquía. Se debe respetar su funcionamiento de tipo histórico de modo que si un dato es enviado no podrá ser eliminado mediante la API. También se ofrece al usuario el acceso a uno o más de los datos almacenados.

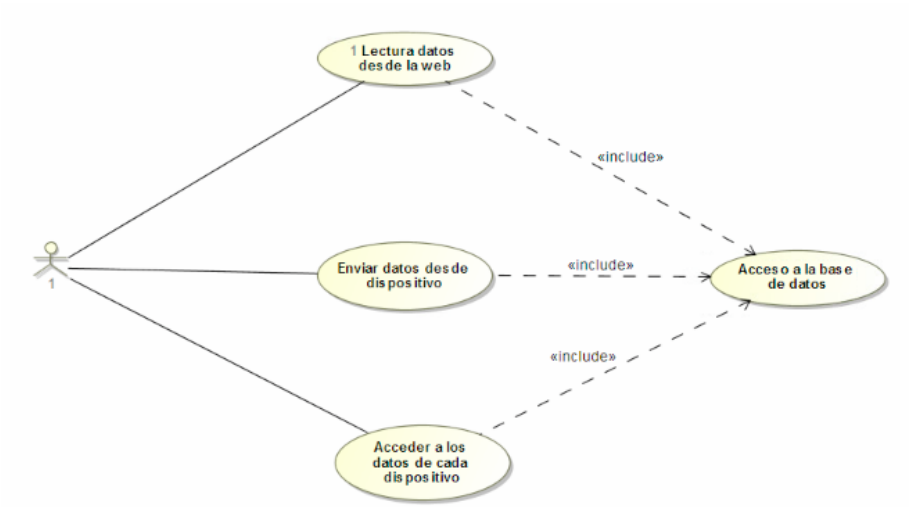

Figura 3.2: Diagrama de caso de uso para el acceso a los datos.

3.4. Diagramas de clases

A continuación es necesario realizar un análisis de las clases que va a utilizar nuestro sistema. Las clases e interfaces elegidas son las siguientes y su relación está retratada en el diagrama 3.2:

- Elemento Jerarquía. Es la interfaz de la que heredan todos las clases que conforman la jerarquía. En este proyecto se ha elegido que esté constituida por solo dos niveles, pero con esta interfaz podría ampliarse como quisiéramos.

- Usuario. Punto de entrada para la jerarquía. Cada usuario puede tener asignados tantos dispositivos como quiera y representa al propietario de los dispositivos. Contiene la información esencial sobre el usuario como dirección o número de teléfono.

- Dispositivo. Cada dispositivo pertenecerá a un solo usuario y será al que se le asignará la información que se reciba. Para cada dispositivo existirá una instancia de esta clase que contará con la información básica como modelo o localización.

- Dato. Interfaz para las clases que vayan a contener los datos recibidos. Se podrán crear implementaciones distintas de esta interfaz si se deseara tener tipos diferenciados de clases dedicadas al almacenamiento de la información.

- Bit. El bit que hemos usado en el diseño es un ejemplo, pero puede ser adaptado a cada situación para contener la información que se necesite. Es el elemento que se encarga de representar la información almacenada por los dispositivos.

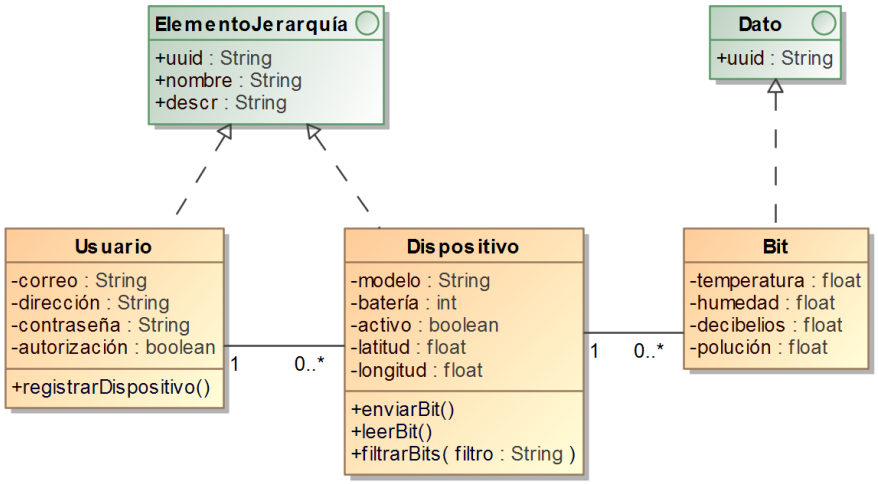

Figura 3.3: Diagrama de clases de nuestra aplicación.

3.5. Diagramas de actividad

En los diagramas 3.4 y 3.5 podemos ver los casos más representativos de las actividades que se ejecutan en el servidor. En primer lugar, un usuario registrará en la jerarquía un dispositivo, debiendo comprobar que la información del dispositivo es correcta antes de su almacenamiento y por otro lado, una petición de unos datos utilizando el filtrado que será realizado en la base de datos.

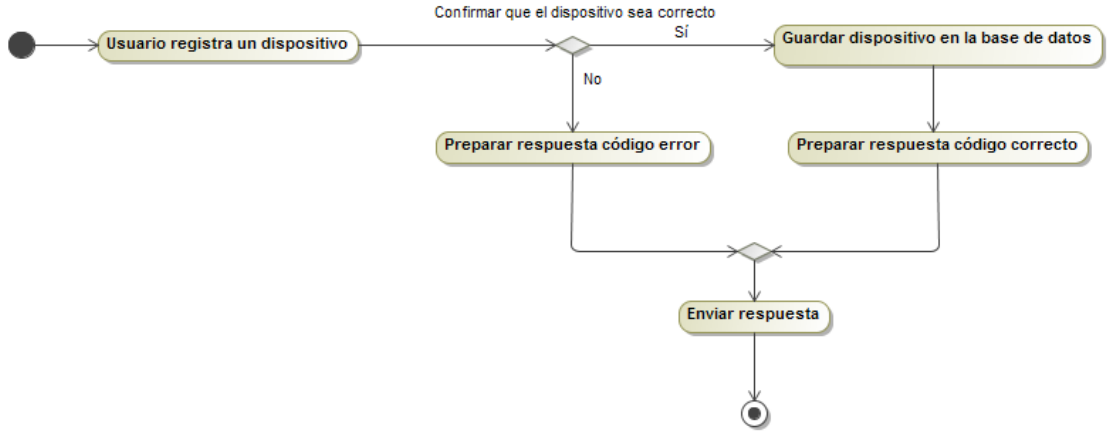

Figura 3.4: Diagrama de actividad de un registro de un nuevo dispositivo.

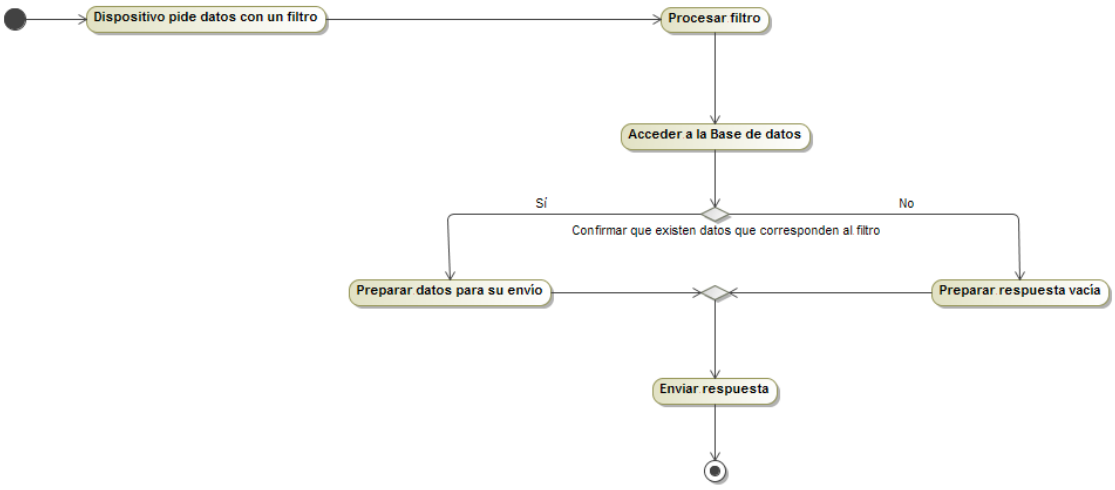

Figura 3.5: Diagrama de actividad de una petición de datos con filtro.

3.6. Arquitectura

Una vez analizado el diseño que va a tener la plataforma, podemos establecer la arquitectura que tendrá el sistema. La decisión ha sido de tomar una arquitectura cliente-servidor ya que encaja perfectamente con nuestras necesidades además de encajar con la estructura del protocolo HTTP que será la base de nuestro desarrollo. En la siguiente imagen se puede observar un diagrama que ilustra el diseño que se ha elegido.

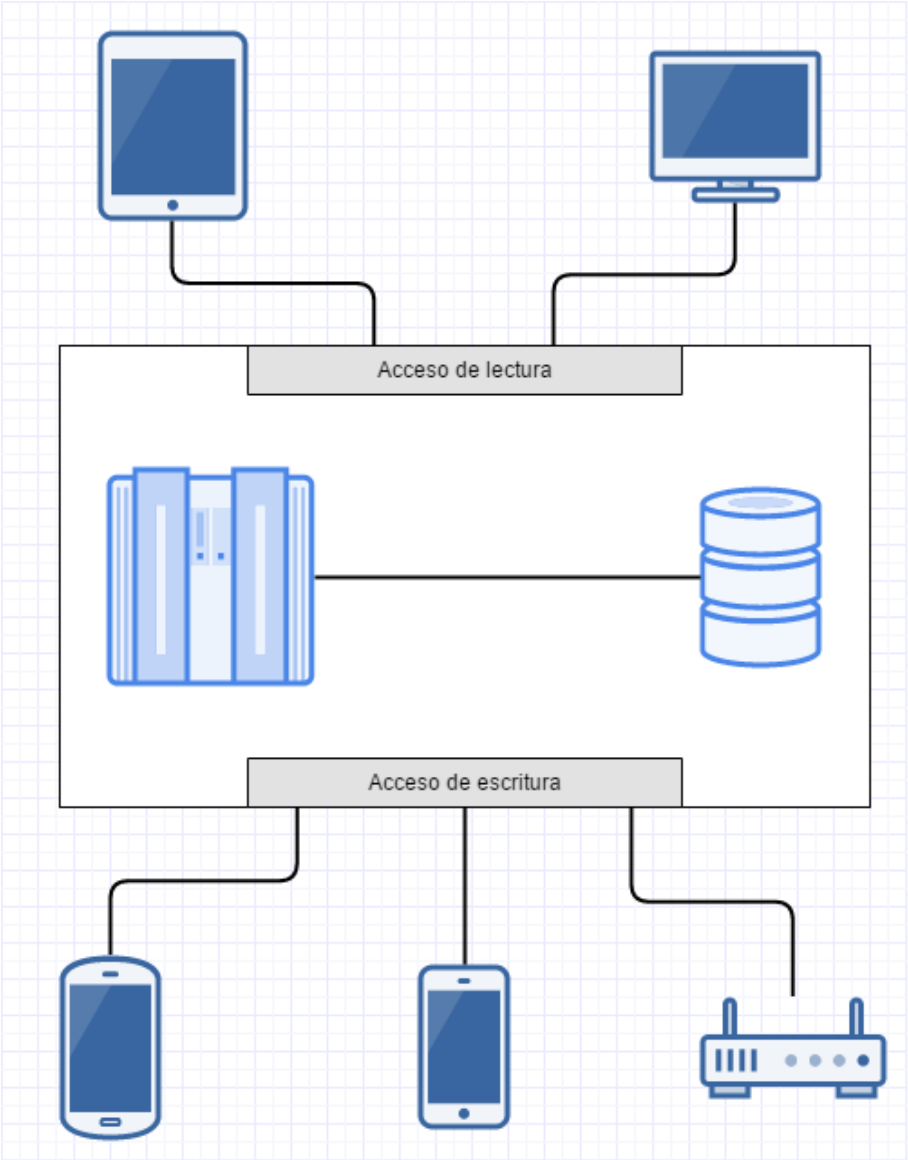

Figura 3.6: Diagrama del diseño de la arquitectura.

Según podemos ver, habrá dos tipos de accesos diferenciados y que dependerá del tipo de aplicación que esté utilizando nuestro sistema:

- Acceso de lectura. La mayoría de aplicaciones complejas que utilizarán los datos almacenados en el servidor como aplicaciones de Big Data o de procesamiento para su interpretación necesitarán realizar un gran número de peticiones de lectura a la clase Bit. Esto puede reducirse con filtros que permitan recuperar un mayor número de datos en un menor número de peticiones.

- Acceso de escritura. Contamos también con una serie de accesos de tipo escritura que se encargarán de la creación y mantenimiento de la estructura, pero también del aprovisionamiento de los datos que sean recibidos por los sensores de los dispositivos. Este acceso creará o modificará las clases de Usuario y Dispositivo, además de crear instancias de la clase Bit que es la que representa los datos.

Aunque estos dos tipos de accesos pueden contar con distintas características en cuanto a la seguridad, nuestro diseño de clases permite un acceso indiferente en los dos tipos, lo que nos ofrece la posibilidad de realizar una implementación sencilla que pueda ser ampliada en el futuro.

3.7. Conclusión del capítulo

En este capítulo hemos podido ver como podemos diseñar el sistema de manera que se ajuste a nuestras necesidades. Este análisis será la clave para que en el próximo capítulo podamos elegir las tecnologías correctas para la implementación, aunque al haber seguido este procedimiento nos hemos asegurado que la tecnología elegida será solo parte de la implementación y no parte del diseño.

Capítulo 4

Análisis de las tecnologías

En este tercer capítulo vamos a analizar brevemente las tecnologías elegidas para el desarrollo del Trabajo. Se pondrá el foco en dichas tecnologías pero también habrá espacio para estudiar algunas alternativas. Hay que tener en cuenta que estamos ante una oferta muy amplia de opciones y que la elección es clave para la implementación que vamos a realizar de nuestro diseño. Las tecnologías han sido elegidas por su facilidad de uso, su excelente soporte y su amplia difusión.

4.1. HTTP

Es la base de todo nuestro desarrollo y como tal deberemos analizarlo para tener una visión muy clara de su funcionamiento. La versión más extendida de HTTP es la 1.1 publicada en 1997 pero que ha ido recibiendo actualicaciones hasta 2014. Es por ello que en primer lugar vamos a centrarnos en esta versión más extendida para luego hacer un pequeño análisis de la nueva versión que ya está comenzando a ser implantada.[16]

4.1.1. Introducción a HTTP

HTTP es un protocolo de comunicación que se encuentra en la base de la web que conocemos hoy en día. Fue creado en 1991 por Tim Berners-Lee en el CERN donde también desarrolló HTML y la tecnología necesaria para los servidores web. Cuenta con un diseño cliente-servidor que permite realizar aplicaciones de manera muy sencilla, de modo que solo es necesario que el cliente emita una *petición* en la que se envía o se pide información y el servidor responderá con una *respuesta* que podrá ser una página HTML, un archivo o un simple código de control. Existen varios tipos de *petición* cada uno con un rol asignado tal y como se detalla en la tabla 4.1. A continuación, en la imagen 4.1, podemos ver como sería una *petición* GET típica de HTTP/1.1:

Método	Descripción
GET	Recupera la información especificada.
POST	Sirve para el envío del información al servidor.
PUT	Modifica información específica en el servidor.
DELETE	Elimina la información especificada.
TRACE	Sirve para ver la petición que se realiza al servidor.
HEAD	Con este método el servidor devuelve tan solo la cabecera.
CONNECT	No se encuentra en uso actualmente.
OPTIONS	Recupera información sobre las opciones del servidor.

Cuadro 4.1: Tipos de *petición* en HTTP/1.1.

Código	Descripción
1xx	Información variada enviada por el servidor.
2xx	Son los códigos de éxito en la petición.
3xx	Redirección.
4xx	Nos informa sobre un error en el cliente.
5xx	Información sobre un error en el servidor.

Cuadro 4.2: Códigos de *respuesta* HTTP/1.1.

```
GET /usuario3752/dispositivo2134/ HTTP/1.1
Accept: application/json
Accept-Language: es-es

User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64; Trident/7.0; rv:11.0) like Gecko
Host: www.test.es
Connection: Keep-Alive
```

Figura 4.1: *Petición* HTTP/1.1 de tipo GET.

Tras la recepción por parte del servidor de esta petición, se dispondrá a responder con una *respuesta* en la que viene incluido un código de control. Los códigos más comunes están detallados en la tabla 4.2. En este caso el servidor responderá con un archivo JSON que es el tipo que espera el cliente como se puede ver en la imagen 4.2.

```
HTTP/1.1 200 OK
Server: Apache/1.3.3.7 (Unix) (Red-Hat/Linux)
Date: Mon, 06 Jun 2016 12:04:43 GMT


Cache-Control: no-cache, no-store

Content-Type: text/json; charset=utf-8
Content-Length: 568

{ id = "2134543",...
```

Figura 4.2: *Respuesta* con código 200.

De esta sencilla manera, conseguimos un protocolo muy versátil y que permite utili-

zarse con mucha libertad. Está en manos del desarrollador utilizar su potencial y nosotros lo usaremos como la base para recibir y enviar información utilizando el formato JSON que será explicado más adelante.

4.1.2. HTTP/2

Es la nueva versión de HTTP, definida en 2015 y que a día de hoy ya se encuentra en algunos servidores web y en todos los navegadores modernos. Su objetivo no es modificar el funcionamiento del protocolo a nivel de desarrollo sino a nivel de transporte. Con esto se busca mejorar la optimización de las comunicaciones reduciendo el tiempo de espera para el usuario y disminuyendo la carga en los servidores. Una importante innovación con respecto a HTTP/1.1 es la introducción del uso de una sola conexión para la descarga de una página web completa, que conlleva una importante mejora de rendimiento.

Otro importante avance que introduce HTTP/2 es el de "server push" que se basa en la idea de que es el servidor el que origina la comunicación con el cliente. Esto es interesante en el sector del Internet de las Cosas ya que nos ofrece la posibilidad de reducir el tráfico en los casos en los que es el cliente el que tiene que esperar alguna información específica del servidor, ya que en lugar de realizar actualizaciones periódicas, el cliente puede esperar a que el servidor tenga lista la información deseada. Este método se encuentra en contraposición al usual "server pull" en el que es el cliente el que pide la información y la espera de forma síncrona.

4.2. Node.js

Node.js es un entorno de desarrollo enfocado en la creación de aplicaciones web para el servidor y fue creado en 2009 por Ryan Dahl. Se trata de una plataforma innovadora ya que combina las novedades del motor de Google "Javascript V8" con una arquitectura basada en eventos que permite una gestión asíncrona de la aplicación. En los siguientes apartados profundizamos más en estos conceptos.[17]

4.2.1. Introducción a Node.js

Node.js supuso un cambio en el paradigma del desarrollo de servidores que hasta entonces habían liderado JavaEE o PHP en el que se realizaba un procesamiento síncrono de las peticiones recibidas. Es en este punto en el que Node.js ha destacado más, llevando el desarrollo asíncrono basado en eventos al desarrollo web. A continuación vamos a definir brevemente algunos de estos conceptos para poder continuar profundizando en el desarrollo en Node.js:

- Síncrono y Asíncrono. Node.js está diseñado para ejecutarse como un único proceso, obligando al desarrollador a utilizar programación asíncrona. La diferencia entre los dos tipos de programación está en que el desarrollo síncrono suele contar con bloqueos en los que el programa espera a que termine una función para continuar

la ejecución. En cambio, con el modelo asíncrono se utilizan funciones de callback que permiten que el código se ejecute sin detenerse, efectuando varias operaciones en paralelo.

- Callback (Retrollamada). No se trata de un concepto único de Node.js pero se encuentra en prácticamente cada pieza de código del entorno. Una callback es una función que se entrega a otra como un argumento y a la que se llama cuando la primera finaliza su ejecución. En el próximo apartado se analizará un ejemplo de una función a la que se llama pasando por argumento una función de callback.

4.2.2. Desarrollo en Node.js

A continuación podemos ver un extracto de código típico de Node.js en el que se realiza una función de callback:

```
1  function haveBreakfast(food, drink, callback) {
2    console.log('Having breakfast of ' + food + ', ' + drink + '.');
3    if (callback && typeof(callback) == "function") {
4      callback();
5    }
6  }
7
8  haveBreakfast('toast', 'coffee', function() {
9    console.log('Finished breakfast. Time to go to work!');
10 });
11 console.log('Reading newspaper.')
```

Listing 4.1: Ejemplo función callback.

Procediendo a analizar con detalle este diseño, podemos observar que al realizar la llamada a la función "haveBreakfast" el código no se bloqueará esperando a que finalice, sino que cuando acabe llamará a la función anónima de callback que hemos pasado como tercer parámetro. Mientras tanto, si la función tuviera que acceder a una base de datos por ejemplo, la ejecución continuaría con el mensaje "Reading newspaper.". Esto nos permite realizar una ejecución en el servidor del código de manera paralela, descolgándonos de la espera en las llamadas a bases de datos o a otros servicios.

En cuanto a la estructura de una aplicación Node.js, se nos ofrece libertad para el desarrollo, pero para facilitarlo existen diversas librerías (llamadas módulos en Node.js) que nos ofrecen una estructura sobre la que trabajar y así ahorrarnos un trabajo que sería muy repetitivo. Para ello vamos a utilizar el módulo Express que cuenta con un soporte directo por parte de Node.js y es el más extendido.

4.2.3. Express

Express es un framework web para Node.js que aprovecha los patrones que se suelen cumplir en las aplicaciones de servidor para facilitar el desarrollo y crear aplicaciones más estables. Aunque se trata de un módulo bastante ligero, cuenta con una funcionalidad para

desarrollar una API basada en JSON que nos podrá ahorrar mucho trabajo en nuestro desarrollo.

Además, Express se encarga de realizar otras tareas por nosotros como la puesta a punto del enrutamiento de las peticiones HTTP o la gestión de vistas y modelos. En concreto, Express creará una aplicación *esqueleto* para nosotros sobre la que podremos desarrollar directamente, ahorrándonos la repetitiva tarea de crear proyectos desde cero en cada nuevo desarrollo. Aunque Express permite realizar otro tipo de organización, la estructura que crea, como podemos ver en la imagen **??**, es la siguiente:

- **node_modules**. Carpeta destinada a contener las librerías utilizadas, que en Node.js son llamadas módulos. No debemos modificarla directamente ya que Node.js cuenta con un gestor de dependencias llamado NPM que se encarga de esto y que se explicará más adelante.

- **bin**. En esta carpeta se encuentra el punto de entrada de la aplicación y por lo general no debe ser modificada.

- **public**. Aquí se almacenan los los recursos que serán accedidos por los usuarios. Generalmente se almacenan aquí los archivos estáticos como imágenes y hojas de estilo.

- **routes**. Será el punto central de nuestro desarrollo y cuenta con los archivos que gestionan las rutas de las peticiones HTTP. Puede estar organizado en diferentes archivos para facilitar su lectura. Esto nos facilitará mucho el trabajo ya que permite crear una API de tipo REST de una manera sencilla y directa.

- **views**. Las vistas serán los trozos de código que generen las páginas web que se presenten a los usuarios. Nosotros no usaremos esta característica pero cabe mencionar que Express utiliza el sistema de plantillas Jade para este cometido.

- **app.js**. Es el archivo central de nuestra aplicación y desde aquí iniciaremos las diferentes rutas necesarias para nuestro sistema así como la inicialización de las bases de datos y otros asuntos generales que no sean específicos de cada *petición*.

- **package.json**. Se trata de un archivo generado por el gestor de dependencias NPM, se irá formando automáticamente conforme vayamos desarrollando la aplicación y definirá diversos aspectos como el arranque del servidor, la gestión de los módulos o la ejecución de los tests unitarios.

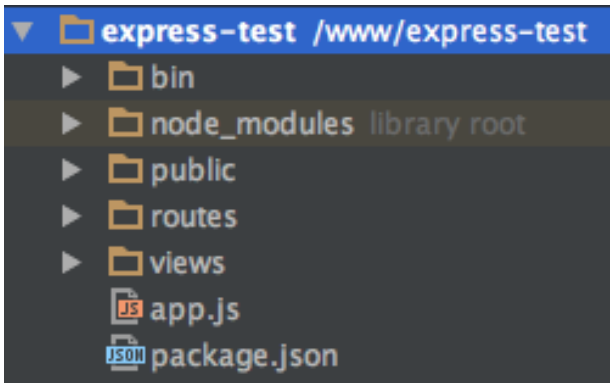

Figura 4.3: Aplicación de Node.js generada automáticamente por Express.

4.2.4. NPM, Javascript y JSON

Para finalizar con el análisis de Node.js, es necesario hacer una corta mención a estos conceptos que han surgido durante esta sección. Además serán de gran utilidad en las proximas secciones y capítulos. Los conceptos más destacables son:

- **Node Package Manager(NPM).** El entorno de Node.js cuenta desde 2011 con este gestor de dependencias que por una parte nos permite abstraernos de la tarea de gestionar las librerías y de su actualización, pero que también se ofrece como una herramienta de gestión de la aplicación con funciones de instalación en el caso de mover el código, de inicialización del servidor o de inicialización de la ejecución de tests unitarios. Además ofrece la posibilidad de crear diferentes modos de inicio, creando por ejemplo, rutinas de inicio distintas para las tareas de depuración y para la puesta en producción.

- **Javascript.** Es el lenguaje en el que se escriben el código de Node.js y por tanto es clave conocer su funcionamiento antes de comenzar a desarrollar en el entorno. Su vida se encuentra muy ligada a la de internet ya que originalmente fue concebido para crear rutinas en las páginas web que se ejecutaran en el lado del cliente. Desde entonces ha tenido un enorme crecimiento y a día de hoy se utiliza, por su excelente versatilidad, en diversos proyectos como Angular.js ó JQuery que ofrecen excelentes herramientas para el desarrollo de aplicaciones en web. Se trata siempre de un lenguaje interpretado por los motores V8 de Google o SpiderMonkey de Mozilla, entre otros.[19]

- **Javascript Object Notation (JSON).** Este estándar para la transmisión de objetos, que surgió en el seno de Javascript, consigue que la información sea legible para las personas gracias a una estructura de pares clave-valor. En la actualidad muchos otros lenguajes cuentan con soporte para JSON y está ampliamente extendido. Será nuestra base para el manejo de la información ya que como veremos más adelante, la base de datos que vamos a utilizar, está basada en esta notación.

4.3. Mongo DB

Mongo DB es un sistema gestor de bases de datos no relacional que ha tenido un importante crecimiento en los últimos años, encontrándose muy ligado a internet y a Node.js. La programación de Mongo DB se realiza en Javascript por lo que utilizaremos el mismo lenguaje en nuestro desarrollo del servidor y en la base de datos.[18]

4.3.1. Introducción a Mongo DB

Mongo DB cuenta con una distribución gratuita y con licencia libre por lo que podemos tener acceso a todo su código y utiliza una estructura no relacional para almacenar la información. Ha sido desarrollada por la empresa 10gen, ahora llamada MongoDB Inc., desde 2007 y cuenta con una comunidad y una documentación excelentes. Su desarrollo siempre ha contado con un foco muy importante en el alto rendimiento, debido a que es utilizado principalmente por servidores web, pero también ha tenido en cuenta mantener su estructura dinámica para poder adaptarse a las necesidades cambiantes de cualquier aplicación web. Cuenta igualmente con características comunes en el mercado de las bases de datos como la agregación, utilizando MapReduce, o el balanceo de carga para permitir la escalabilidad de la base de datos.

4.3.2. Estructura no relacional

Es una de las características clave de Mongo DB, que permite desprenderse de la estructura rígida de las tablas en las bases de datos relacionales para utilizar un sistema de esquemas dinámicos que Mongo DB denomina "BSON (Binary JSON)" en clara referencia a la estructura JSON. En la imagen 4.4 podemos ver como está diseñada la estructura de esta base de datos. Cada base de datos está formada por colecciones y son el equivalente a las tablas en una base de datos relacional, pudiendo contener tantas colecciones como sea necesario. Por otro lado contamos con los documentos, que serían equivalentes a las filas, y conforman la información que es almacenada en nuestra base de datos. Los documentos son objetos BSON y deben ser almacenados siempre dentro de una colección.

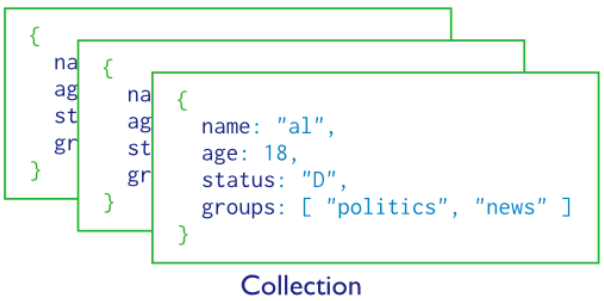

Figura 4.4: Estructura de una base de datos en Mongo DB.

Por último, a continuación se muestra un ejemplo de escritura y lectura en Mongo DB.

Donde podemos observar que se utilizan llamadas a funciones javascript para realizar estas tareas, además de utilizar el sistema de callback que definimos con anterioridad.

```
1 var collection = db.collection('test');
2 var doc = { myket:1,
3       text:"Hola Mundo." };
4
5 collection.insert(doc);
6 collection.findOne({mykey:1}, function(err, item) {});
```

Listing 4.2: Ejemplo de escritura y lectura en Mongo DB.

Capítulo 5

Implementación de la plataforma

En esté penúltimo capítulo vamos a desglosar el desarrollo de la plataforma de una manera resumida pero que tanto la funcionalidad del entorno como de la aplicación en sí queden bien claras. Comenzaremos con el entorno de desarrollo y las herramientas utilizadas para esta tarea y continuaremos con la explicación de algunas de las partes más importantes del servidor, así como un análisis de la estructura utilizada en Mongo DB.

5.1. Entorno de desarrollo

La elección y la puesta en marcha de un correcto entorno de desarrollo es una parte importante del proceso de implementación y por tanto vamos a analizar los distintos componentes que han sido claves en este proceso. Cabe destacar que se ha intentado utilizar software libre en la medida de lo posible, acudiendo en su ausencia a software propietario pero con licencias de educación o licencias gratuitas.

5.1.1. WebStorm

Es un IDE desarrollado específicamente para el desarrollo web, creado por la compañía JetBrains, y que nos ofrece un entorno integrado con soporte específico para Node.js, lo cual nos facilita mucho la tarea del desarrollo. Cuenta con función de autocompletar y con integración con gestores de versiones. Es esta integración con diferentes elementos del desarrollo lo que hace de este entorno una de las mejores apuestas para trabajar en Node.js frente a otros entornos o editores como Aptana o SublimeText.

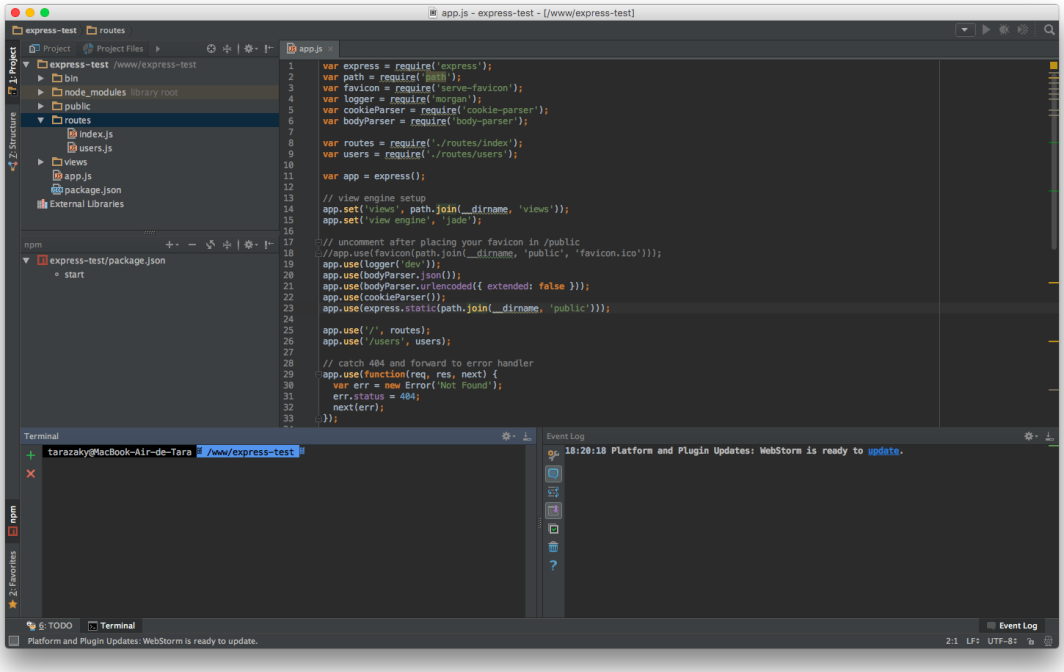

Figura 5.1: Captura de pantalla del IDE WebStorm.

5.1.2. Git

Git es un sistema de control de versiones que sigue un diseño distribuido en el que cada cliente cuenta con un repositorio completo. Fue creado por Linus Torvalds en 2005 para contribuir al desarrollo de Linux. Cuenta con una licencia de uso de software libre y su uso está tan extendido que la mayoría de los entornos de desarrollo ya traen integrado este sistema, además de muchos sistemas operativos. En los últimos años, plataformas colaborativas como Github o Bitbucket, que usan Git como la base de su diseño, han permitido una penetración aún mayor, llegando a utilizarse para otras tareas fuera de la programación.[20]

En cuanto a su funcionamiento, en la imagen 5.2 podemos ver como se estructura Git y a continuación en la imagen 5.3 se observa el llamado "gitflow workflow", flujo de trabajo de Git, que es una de las muchas formas de organizar el desarrollo sobre Git pero la más recomendada para el desarrollo de software en equipos de trabajo.

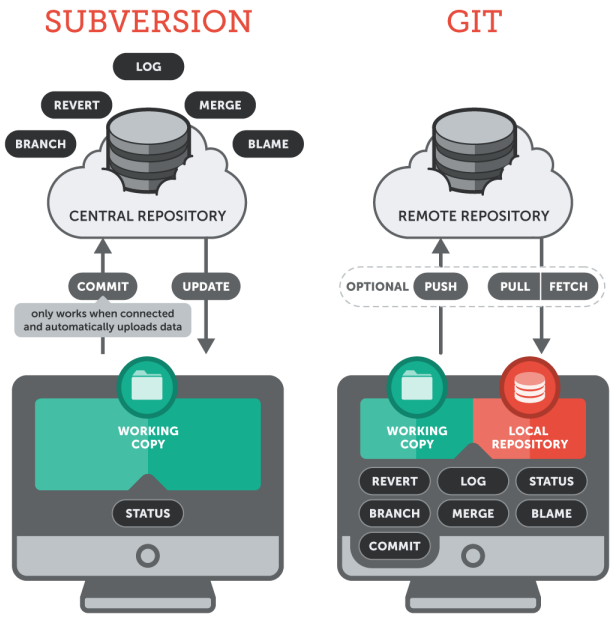

Figura 5.2: Estructura de Git frente a SVN, otro gestor de versiones.

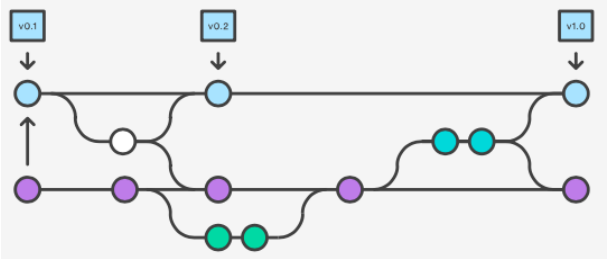

Figura 5.3: Flujo de trabajo de Gitflow.

5.1.3. Servidor de pruebas

Para nuestro servidor de pruebas se ha utilizado una máquina virtual instalada en un servidor de la Escuela. Esta máquina virtual cuenta con el sistema operativo Ubuntu 16.04 el cual cuenta con muchos de los elementos que vamos a necesitar para el desarrollo, como Git o Mongo DB. Una vez instalado el gestor de dependencias NPM, podemos clonar el servidor utilizando Git para proceder instalarlo y arrancarlo con muy poco esfuerzo. Este proceso además puede ser repetido en otras plataformas que estén basadas en Unix como Meshlium de Libelium o cualquier otra distribución de Linux. En el apéndice se incluirá una pequeña guía para instalar y arrancar el servidor de manera genérica.

5.2. Implementación de funcionalidades

A continuación, llegamos al punto clave del desarrollo, en el que se explicarán las partes más importantes de la implementación. Comenzaremos comentando la estructura que ha tomado nuestro proyecto de Node.js para continuar con el análisis de las rutas

HTTP diseñadas y la base de datos Mongo DB que se encuentra detrás. Por último, se mostrará el funcionamiento del servidor y de la página web de prueba, así como de la rutina de generación de datos.

5.2.1. Inicialización

La mayor parte de la puesta en marcha de la aplicación viene gestionada por Express de modo que en el archivo "app.js" los cambios serán mínimos.A continuación se muestra el archivo ya que es la base del servidor. Los cambios más importantes son la introducción del archivo que contiene la ruta de nuestra API.

Podemos destacar dentro de este archivo, la forma en la que en Node.js se importan las librerías, con la función import. Esto se realiza tanto para librerías como para archivos con código a los que quisiéramos acceder desde aquí. En este archivo se le especifica que utilice las rutas contenidas en "/routes" y además se diseña la función que gestionará los errores de tipo 404, información no encontrada.

```
1  var express = require('express');
2  var path = require('path');
3  var favicon = require('serve-favicon');
4  var logger = require('morgan');
5  var cookieParser = require('cookie-parser');
6  var bodyParser = require('body-parser');
7
8  var routes = require('./routes/iot');
9
10 var app = express();
11
12 app.set('views', path.join(__dirname, 'views'));
13 app.set('view engine', 'jade');
14
15 app.use(logger('dev'));
16 app.use(bodyParser.json());
17 app.use(bodyParser.urlencoded({ extended: false }));
18 app.use(cookieParser());
19 app.use(require('less-middleware')(path.join(__dirname, 'public')));
20 app.use(express.static(path.join(__dirname, 'public')));
21
22 app.use('/', routes);
23 app.use('/users', users);
24
25 // catch 404 and forward to error handler
26 app.use(function(req, res, next) {
27   var err = new Error('Not Found');
28   err.status = 404;
29   next(err);
30 });
31
32 // error handlers
33
```

```
34 // development error handler
35 // will print stacktrace
36 if (app.get('env') === 'development') {
37   app.use(function(err, req, res, next) {
38     res.status(err.status || 500);
39     res.render('error', {
40       message: err.message,
41       error: err
42     });
43   });
44 }
45
46 // production error handler
47 // no stacktraces leaked to user
48 app.use(function(err, req, res, next) {
49   res.status(err.status || 500);
50   res.render('error', {
51     message: err.message,
52     error: {}
53   });
54 });
55
56
57 module.exports = app;
```

Listing 5.1: Archivo principal de la aplicación

5.2.2. Rutas HTTP

Al tratarse de una aplicación cuyo valor se encuentra en su API, la mayor parte de la funcionalidad gira en torno a el archivo de routas, en este caso llamado "iot.js". El archivo, gracias al diseño de Node.js, cuenta con una estructura muy legible y ordenada ofreciendo un desarrollo muy rápido y muy cercano al diseño.

En este archivo se ha implementado para cada punto de acceso de la API los cuatro métodos principales de HTTP que nos permiten ofrecer un servicio de escritura, modificación, lectura y borrado de la información sobre la jerarquía. En el caso de los datos tan solo se ofrecen métodos de lectura y escritura pero con lectura por filtrado o lectura del último dato recibido.

```
1  var express = require('express');
2  var querystring = require('querystring');
3  var router = express.Router();
4
5  var mongo = require('../mongo');
6  var db;
7
8  /* GET home page. */
9  router.all('/', function (req, res, next) {
10   res.render('index', { title: 'Express' });
```

```
11 });
12
13 /* USER CRUD */
14 router.route('/:user_id')
15     .all(function (req, res, next) {
16         next();
17     })
18     .get(function (req, res, next) {
19         var user_id = req.params.user_id;
20         var collection = 'users';
21         db.collection(collection).findOne({'user_id':user_id}, function(
               err, result){
22             assert.equal(err, null);
23             res.send(result);
24         });
25     })
26     .post(function (req, res, next) {
27         var user_id = req.params.user_id;
28         var collection = 'users';
29         var json = req.query;
30         json['user_id'] = user_id;
31
32         db.collection(collection).insertOne(json, function(err, result)
               {
33             assert.equal(err, null);
34             res.send("Inserted " + JSON.stringify(json) + " into the " +
                   collection + " collection.");
35         });
36     })
37     .put(function (req, res, next) {
38         var user_id = req.params.user_id;
39         var collection = 'users';
40         var json = req.query;
41
42         db.collection(collection).replaceOne({'user_id':user_id}, json,
               function(err, result) {
43             assert.equal(err, null);
44             res.send("Modified " + JSON.stringify(json) + " from the " +
                   collection + " collection.");
45         });
46     })
47     .delete(function (req, res, next) {
48         var user_id = req.params.user_id;
49         var collection = 'users';
50         db.collection(collection).removeOne({'user_id':user_id},
               function(err, result){
51             assert.equal(err, null);
52             res.send("Deleted document from the " + collection + "
                   collection.");
53         });
54     });
```

```
55
56 router.route('/:user_id/:device_id')
57     .all(function (req, res, next) {
58         next();
59     })
60     .get(function (req, res, next) {
61         var user_id = req.params.user_id;
62         var device_id = req.params.device_id;
63         var collection = 'devices';
64
65         db.collection(collection).findOne({'user_id':user_id, 'device_id
               ':device_id}, function(err, result){
66             assert.equal(err, null);
67             res.send(result);
68         });
69     })
70     .post(function (req, res, next) {
71         var user_id = req.params.user_id;
72         var device_id = req.params.device_id;
73         var collection = 'devices';
74         var json = req.query;
75         json['user_id'] = user_id;
76         json['device_id'] = device_id;
77
78         db.collection(collection).insertOne(json, function(err, result)
               {
79             assert.equal(err, null);
80             res.send("Inserted " + JSON.stringify(json) + " into the " +
                   collection + " collection.");
81         });
82     })
83     .put(function (req, res, next) {
84         var user_id = req.params.user_id;
85         var device_id = req.params.device_id;
86         var collection = 'devices';
87         var json = req.query;
88         json['user_id'] = user_id;
89         json['device_id'] = device_id;
90
91         db.collection(collection).replaceOne({'user_id':user_id, '
               device_id':device_id},json, function(err, result) {
92             assert.equal(err, null);
93             res.send("Modified " + JSON.stringify(json) + " from the " +
                   collection + " collection.");
94         });
95     })
96     .delete(function (req, res, next) {
97         var user_id = req.params.user_id;
98         var device_id = req.params.device_id;
99         var collection = 'devices';
100
```

```
101         db.collection(collection).removeOne({'user_id':user_id, '
                device_id':device_id}, function(err, result){
102             assert.equal(err, null);
103             res.send(result);
104         });
105     });
106
107 router.route('/:user_id/:device_id/bits')
108     .all(function (req, res, next) {
109         next();
110     })
111     .get(function (req, res, next) {
112         var user_id = req.params.user_id;
113         var device_id = req.params.device_id;
114         var collection = user_id+'_'+device_id;
115         var filter = req.query;
116
117         if(isEmpty(filter)){
118             var response = db.collection(collection).find().toArray(
                    function(err, result){
119                 assert.equal(err, null);
120                 res.send(result);
121             });
122         }else{
123             var response = db.collection(collection).find(filter).
                    toArray(function(err, result){
124                 assert.equal(err, null);
125                 res.send(result);
126             });
127         }
128
129     })
130     .post(function (req, res, next) {
131         var user_id = req.params.user_id;
132         var device_id = req.params.device_id;
133         var collection = user_id+'_'+device_id;
134         var json = req.query;
135         json['user_id'] = user_id;
136         json['device_id'] = device_id;
137
138         db.collection(collection).insertOne(json, function(err, result)
                {
139             assert.equal(err, null);
140             res.send("Inserted " + JSON.stringify(json) + " into the " +
                    collection + " collection.");
141         });
142
143     });
144
145 router.route('/:user_id/:device_id/bits/last')
146     .all(function (req, res, next) {
```

```
147         next();
148     })
149     .get(function (req, res, next) {
150         var user_id = req.params.user_id;
151         var device_id = req.params.device_id;
152         var collection = user_id+'_'+device_id;
153 
154         var options = {"sort":[['_id','desc']], "limit":1};
155         db.collection(collection).find({}, options).toArray(function(err
                , result){
156             assert.equal(err, null);
157             res.send(result);
158         });
159 
160     });
161 
162 function isEmpty(obj) {
163     return Object.keys(obj).length === 0;
164 }
165 
166 module.exports = router;
```

Listing 5.2: Partes del archivo iot.js

5.2.3. Base de datos

Como ya hemos mencionado anteriormente, Mongo DB está especialmente diseñada para ser muy compatible con Node.js, entre otros, por lo que su puesta a punto es bastante sencilla y se realiza dentro de un archivo que será importando donde sea necesario. Podemos observar que el código resulta muy sencillo éste es suficiente para establecer una correcta conexión con la base de datos. La variable db es exportada y utilizada en donde sea necesario el acceso a la base de datos.

```
1  var MongoClient = require('mongodb').MongoClient;
2  var assert = require('assert');
3  var ObjectId = require('mongodb').ObjectID;
4  var url = 'mongodb://localhost:27017/iot';
5  var db;
6 
7  MongoClient.connect(url, function(err, database) {
8      if(!err){
9          console.log("We are connected");
10         db = database;
11     }else{
12         console.log("DB ERROR");
13     }
14 });
15 
16 exports.db = function () {
17     return db;
```

```
};
```

Listing 5.3: Contenido del archivo mongo.js

5.2.4. Servidor en funcionamiento

Para finalizar el desarrollo del servidor, vamos a analizar brevemente el correcto funcionamiento del servidor, por medio de la salida por pantalla y de la aplicación de consola Httpie.

```
$ http POST 127.0.0.1:8000/angajime\?name\=antonio
HTTP/1.1 200 OK
Connection: keep-alive
Content-Length: 108
Content-Type: text/html; charset=utf-8
Date: Fri, 24 Jun 2016 01:17:43 GMT
ETag: W/"6c-QWsEzN0VSMoJozS0SxtEQg"
X-Powered-By: Express

Inserted {"name":"antonio","user_id":"angajime","_id":"576
    c8a37e0ab121ff8902f61"} into the users collection.

$ http GET 127.0.0.1:8000/angajime
HTTP/1.1 200 OK
Connection: keep-alive
Content-Length: 72
Content-Type: application/json; charset=utf-8
Date: Fri, 24 Jun 2016 01:17:46 GMT
ETag: W/"48-zsynLHtVRdxJONTfpZZb6w"
X-Powered-By: Express

{
    "_id": "576c895e79233f79efbe3608",
    "name": "antonio",
    "user_id": "angajime"
}
```

Listing 5.4: Creación y lectura de un usuario

```
$ http POST 127.0.0.1:8000/angajime/arduino\?loc\=garden
HTTP/1.1 200 OK
Connection: keep-alive
Content-Length: 130
Content-Type: text/html; charset=utf-8
Date: Fri, 24 Jun 2016 01:25:35 GMT
ETag: W/"82-4ZZXh2bGg7M6naGEaqIi0w"
X-Powered-By: Express

Inserted {"loc":"garden","user_id":"angajime","device_id":"arduino","
    _id":"576c8c0fe0ab121ff8902f62"} into the devices collection.

```

```
$ http POST 127.0.0.1:8000/angajime/arduino3\?loc\=garden
HTTP/1.1 200 OK
Connection: keep-alive
Content-Length: 131
Content-Type: text/html; charset=utf-8
Date: Fri, 24 Jun 2016 01:25:49 GMT
ETag: W/"83-ORny9suj1Wl3WsIssUfV0Q"
X-Powered-By: Express

Inserted {"loc":"garden","user_id":"angajime","device_id":"arduino3","
    _id":"576c8c1de0ab121ff8902f63"} into the devices collection.
```

Listing 5.5: Creación y lectura de un dispositivo

```
$ http POST 127.0.0.1:8000/angajime/arduino/bits\?temperatura\=32\&day
    \=17
HTTP/1.1 200 OK
Connection: keep-alive
Content-Length: 154
Content-Type: text/html; charset=utf-8
Date: Fri, 24 Jun 2016 01:30:14 GMT
ETag: W/"9a-UdHSPb3GXqx0NXZisNiSyg"
X-Powered-By: Express

Inserted {"temperatura":"32","day":"17","user_id":"angajime","
    device_id":"arduino","_id":"576c8d26e0ab121ff8902f68"} into the
    angajime_arduino collection.

$ http GET 127.0.0.1:8000/angajime/arduino/bits\?day\=17
HTTP/1.1 200 OK
Connection: keep-alive
Content-Length: 217
Content-Type: application/json; charset=utf-8
Date: Fri, 24 Jun 2016 01:30:25 GMT
ETag: W/"d9-eNLI4lOwUKnsvjt1NXqKQQ"
X-Powered-By: Express

[
    {
        "_id": "576c8d1ce0ab121ff8902f67",
        "day": "17",
        "device_id": "arduino",
        "temperatura": "35",
        "user_id": "angajime"
    },
    {
        "_id": "576c8d26e0ab121ff8902f68",
        "day": "17",
        "device_id": "arduino",
        "temperatura": "32",
        "user_id": "angajime"
```

```
35      }
36  ]
```

Listing 5.6: Envío de datos y lectura con filtro.

Podemos observar que en los tres casos contamos con una API muy sencilla que hace muy legible la comunicación con el servidor. Este era uno de los objetivos originarios del Trabajo ya que una interfaz accesible para los usuarios siempre va a mejorar su interacción y ayuda a realizar una depuración más sencilla. En el caso de la lectura de datos con filtro, hemos usado uno muy sencillo, pero MongoDB nos permite utilizar filtros mucho mas complejos.

Desde el punto de vista del servidor, se nos informa de las diferentes peticiones recibidas y se muestra además el código con el que se ha respondido a estas junto con la latencia. De este modo, contamos con una visión de cliente y servidor que nos permite depurar fácilmente cualquier problema que surja durante el desarrollo.

```
1   $ npm start
2
3   > iot-server@0.0.0 start /www/iot-server
4   > node ./bin/www
5
6   We are connected
7   POST /angajime?name=antonio 200 10.236 ms - 108
8   GET /angajime 200 8.401 ms - 72
9   POST /angajime?name=antonio 200 1.559 ms - 108
10  GET /angajime 200 3.021 ms - 72
11  POST /angajime/arduino?loc=garden 200 2.826 ms - 130
12  POST /angajime/arduino3?loc=garden 200 1.077 ms - 131
13  GET /angajime/arduino 200 1.293 ms - 92
14  POST /angajime/arduino/bits?temperatura=35 200 3.556 ms - 143
15  POST /angajime/arduino/bits?temperatura=34 200 0.909 ms - 143
16  POST /angajime/arduino/bits?temperatura=36 200 0.795 ms - 143
17  GET /angajime/arduino/bits?temperatura%3E34 200 2.124 ms - 2
18  GET /angajime/arduino/bits?temperatura%3E35 200 1.494 ms - 2
19  GET /angajime/arduino/bits 200 1.408 ms - 292
20  GET /angajime/arduino/bits?temperatura=35 200 1.584 ms - 98
21  GET /angajime/arduino/bits?temperatura=35&day=17 200 0.938 ms - 2
22  GET /angajime/arduino/bits?temperatura=36&day=17 200 0.918 ms - 2
23  GET /angajime/arduino/bits 200 1.284 ms - 292
24  POST /angajime/arduino/bits?temperatura=35&day=17 200 0.762 ms - 154
25  POST /angajime/arduino/bits?temperatura=32&day=17 200 0.845 ms - 154
26  GET /angajime/arduino/bits?day=17 200 0.945 ms - 217
```

Listing 5.7: Salida del servidor.

5.2.5. Página web y rutina

Uno de los objetivos definidos en esta memoria era el de la creación de una simple página web, que acompañado de una rutina de línea de comandos, pudiera simular el

funcionamiento de un despliegue real de dispositivos para así observar, y depurar, el funcionamiento de nuestra plataforma. Para ello, en el mismo servidor se ha colgado una página web en la ruta raíz de modo que fuera fácilmente accesible. La página web utiliza las librerías Bootstrap y jQuery para facilitar el diseño y mejorar su funcionalidad. Se ha elegido un despliegue de sensores de contaminación medioambiental que, desplegados por la ciudad de Málaga, permiten mostrar información en tiempo real sobre un mapa de la ciudad. A continuación se adjunta parte del código de la web y la rutina que genera los datos.

```
<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="initial-scale=1.0, user-scalable=no">
    <meta charset="utf-8">
    <title>Simple web app</title>
    <script src="https://code.jquery.com/jquery-2.2.4.min.js" integrity=
       "sha256-BbhdlvQf/xTY9gja0Dq3HiwQF8LaCRTXxZKRutelT44=" crossorigin
       ="anonymous">
    </script>
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/
       bootstrap/3.3.6/css/bootstrap.min.css" integrity="sha384-1
       q8mTJOASx8j1Au+a5WDVnPi2lkFfwwEAa8hDDdjZlpLegxhjVME1fgjWPGmkzs7"
       crossorigin="anonymous">
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/
       bootstrap/3.3.6/css/bootstrap-theme.min.css" integrity="sha384-
       fLW2N01lMqjakBkx3l/M9EahuwpSfeNvV63J5ezn3uZzapT0u7EYsXMjQV+0En5r"
        crossorigin="anonymous">
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/js/
       bootstrap.min.js" integrity="sha384-0
       mSbJDEHialfmuBBQP6A4Qrprq5OVfW37PRR3j5ELqxss1yVqOtnepnHVP9aJ7xS"
       crossorigin="anonymous"></script>
    <style>
    html, body {
        height: 100%;
        margin: 0;
        padding: 0;
    }
    #map {
        height: 100%;
    }
    </style>
</head>
<body>
    <div id="map"></div>
    <script>
    var devices = [
        {name:'d0',lat: 36.7265818, lng: -4.4236739, marker: null},
        {name:'d1',lat: 36.7261693, lng: -4.4176548, marker: null},
        {name:'d2',lat: 36.7225686, lng: -4.4217912, marker: null},
        {name:'d3',lat: 36.7200547, lng: -4.4287248, marker: null},
```

```
31          {name:'d4',lat: 36.7265818, lng: -4.4343191, marker: null},
32          {name:'d5',lat: 36.7241312, lng: -4.4301827, marker: null},
33          {name:'d6',lat: 36.7307617, lng: -4.4172988, marker: null},
34          {name:'d7',lat: 36.7234518, lng: -4.4256733, marker: null},
35          {name:'d8',lat: 36.7281123, lng: -4.4294029, marker: null},
36          {name:'d9',lat: 36.7225142, lng: -4.417943,  marker: null}
37      ];
38
39      function createMarker(map, latlng, color, marker){
40
41          if(marker == null)
42                  marker = new google.maps.Marker({
43                      position: latlng,
44                      map: map,
45                      title: 'Device updated.',
46                      icon: color,
47                      visible: true
48                  });
49          else
50                  marker.setIcon(color);
51          marker.setMap(map);
52      }
53
54      function initMap() {
55          var myLatLng = {lat: 36.7265818, lng: -4.4236739};
56
57          var map = new google.maps.Map(document.getElementById('map'), {
58              zoom: 16,
59              center: myLatLng
60          });
61
62          var time = 500;
63          setInterval(function(){
64              for(var i=0;i<=9;i++){
65                  var url = '/angajime/'+devices[i].name+'/bits/last';
66                  $.get(url, function(data, status){
67                      var color;
68                      console.log(data);
69                      console.log(status);
70                      if(data[0].polution == 1){
71                          color = 'http://maps.google.com/mapfiles/ms/
                               icons/green-dot.png';
72                      }else if(data[0].polution == 2){
73                          color = 'http://maps.google.com/mapfiles/ms/
                               icons/yellow-dot.png'
74                      }else if(data[0].polution == 3){
75                          color = 'http://maps.google.com/mapfiles/ms/
                               icons/purple-dot.png'
76                      }else if(data[0].polution == 4){
77                          color = 'http://maps.google.com/mapfiles/ms/
                               icons/red-dot.png'
```

```
                    }
                    console.log(color);
                    var num = parseInt(data[0].device_id.substr(data[0].
                        device_id.length - 1));
                    var latlng = {lat: devices[num].lat, lng: devices[
                        num].lng};
                    var marker = devices[num].marker;
                    createMarker(map, latlng, color, marker);
                });
            }
        },time);

    }




</script>
<script async defer
src="https://maps.googleapis.com/maps/api/js?key=
    AIzaSyBM5mQ8hKgHdqMX9im8d35yftasrFjFawY&signed_in=true&callback=
    initMap"></script>
</body>
</html>
```

```
#!/bin/bash

declare -a devices=( "d0" "d1" "d2" "d3" "d4" "d5" "d6" "d7" "d8" "d9" )
echo "init"
while :
do
  #http POST 127.0.0.1:8000/angajime/arduino/bits\?temperatura\=35\&day
      \=17

  for device in "${devices[@]}"
  do
      pol=$(( ( RANDOM % 4 )  + 1 ))
      echo ${pol}
      curl 127.0.0.1:8000/angajime/${device}/bits?polution=${pol} -X
          POST
  done
  sleep 1

done
```

Capítulo 6

Conclusión y líneas futuras

Para el útlimo capítulo, vamos a hacer una pequeña retrospectiva una vez hemos finalizado el Trabajo. Para ello intentaremos sacar algunas conclusiones y establecer algunas líneas que pueden surgir a partir de este proyecto.

6.1. Conclusión

El título de este Trabajo Fin de Grado es "DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA INTERMEDIARIO PARA INTERNET DE LAS COSAS" y, como conclusión a este trabajo podemos decir que aunque el diseño y la implementación ha resultado no ser demasiado compleja, teniendo en cuenta el limitado alcance del proyecto, es la amplia cantidad de tecnologías y agentes en juego lo que fundamenta la verdadera complejidad del campo del Internet de las Cosas. Esto nos ofrece por un lado una oportunidad excelente para descubrir nuevas tecnologías, como ha sido en mi caso con Node.js y Mongo DB, pero por otro, hace que cualquier proyecto comercial alcance una gran complejidad en poco tiempo.

Los mayores problemas a los que me he enfrentado durante la realización han sido relacionados con la elección de la tecnología y con el diseño de una arquitectura que pudiera acoger el mayor número de casos posibles. Esto se trata de un problema de doble filo porque si apuntamos hacia una arquitectura más general, será a costa de ofrecer menos funcionalidades personalizadas que a ciertos usuarios les puede facilitar mucho el desarrollo.

Se debe mencionar que durante la preparación del Trabajo Fin de Grado se definió una prueba a realizar en un entorno real pero que, por motivos de tiempo, no ha podido realizarse. La idea inicial era instalar sobre un servidor Meshlium de la empresa Libelium que nos permitiera desplegar el sistema en una situación que simulara mejor la de un entorno de producción.

Uno de nuestros objetivos era la utilización de la tecnología existente, con el fin de aprovechar la investigación y el desarrollo que ya se ha realizado sobre ésta y no repetir

trabajo. Nuestro análisis inicial creo que ha dado buenos frutos ya que HTTP y Node.JS cuentan con una excelente base de información y creo que eso ha sido una de las piezas que más me ha facilitado el trabajo. Además, como hemos podido ver en el diseño, la plataforma está pensada para que pueda ser extensible, dando cabida de manera sencilla a nuevas estructuras de jerarquías, pudiendo adaptarse a otras situaciones.

Existe una gran competencia a día de hoy para tomar las riendas del Internet de las Cosas, pero la falta de un estándar claro y la pluralidad de este ecosistema suponen un reto que espero este Trabajo ayude a superar.

En definitiva, el Internet de las Cosas es un concepto que es difícil de llevar adelante, tal y como se ve en el gran desarrollo que se está dando en este campo pero en la poca penetración con la que cuenta en nuestras vidas diarias, aunque si la tendencia sigue así, en un futuro no muy lejano puede que veamos una verdadera revolución en este campo.

6.2. Líneas futuras

De este trabajo se pueden desprender algunas líneas que serían muy interesante de desarrollar parte debido a la falta de tiempo para realizar un trabajo más extenso y también por que se trata de otros campos que se encuentran relacionados con el Internet de las Cosas. Estas son las líneas más destacables y son asuntos con los que me he encontrado durante la realización del Trabajo. Los tres son caminos interesantes para avanzar y formaría un buen complemento para este proyecto.

- **Big Data.** Con el impresionante desarrollo que está teniendo la tecnología relacionada con el concepto de Big Data, que trata sobre el procesado de inmensas cantidades de datos para obtener información muy útil, el Internet de las Cosas puede servir de base para alimentar estos sistemas debido a gran generación de datos que puede crear un despliegue en una ciudad o cualquier otro sistema. Esto es una pieza esencial para el futuro del Internet de las Cosas ya que generar información útil a partir de estos datos también podrá ser de provecho para mejorar la calidad de los despliegues de sensores o crear redes de actuadores que se adelanten a los acontecimientos aprovechando la componente de predicción con la que cuenta el Big Data.

- **Seguridad.** La seguridad es un punto clave en el Internet de las Cosas aunque para evitar la sobreextensión se decidió eliminarla del alcance del proyecto. Desarrollar un buen sistema de seguridad que proteja la información de cada usuario es un punto muy importante y estos sistemas tan horizontales hacen que la seguridad sea un reto interesante. Hoy en día muchos usuarios dan una gran importancia a la seguridad de su información y cualquier cliente deseará que sus datos estén bien protegidos ya que en algunos casos podría contener información sensible.

- **Escalabilidad.** Node.js y Mongo DB son entornos que ofrecen unas posibilidades excelentes de escalabilidad para nuestras aplicaciones y, debido a la gran cantidad de

dispositivos que debe manejar un sistema intermediario como este, es muy necesario explorar este tema en profundidad y estudiar las mejores opciones de crecimiento del sistema.

Capítulo 7

Apéndice A

7.1. Guía de instalación del servidor

A continuación, se muestra un rápido manual de uso para levantar el servidor de modo que cualquiera pueda levantar el sistema. No se explicarán todos las herramientas utilizadas por separado ya que en las respectivas páginas de dichas herramientas se encuentran guías de instalación rápida que son muy sencillas de seguir. Durante la guía se proporcionarán enlaces a esta documentación adicional para facilitar su acceso.

7.1.1. Sistema operativo

En el despliegue de prueba hemos utilizado un servidor Ubuntu 16.04 que nos facilita mucho el trabajo porque ya cuenta con muchas de las herramientas necesarias para la instalación. Por otro lado, para el desarrollo y las pruebas durante la depuración se ha utilizado un sistema operativo OS X por lo que su uso es prácticamente independiente de la plataforma donde se instale.

7.1.2. Requisitos previos

Nuestro despliegue cuenta con una serie de herramientas que es necesario tener instaladas y en algunos casos deben estar ejecutándose para poder lanzar la aplicación, como es el caso del sistema gestor de bases de datos, Mongo DB. A continuación vamos a realizar un listado, comentando las diferentes herramientas:

- Git. Instalado por defecto en Ubuntu, nos permitirá clonar el proyecto del repositorio para poder contar con el código del servidor actualizado.
- Mongo DB. Se trata de nuestro sistema gestor de bases de datos y contar con una instalación local. Puede ser instalado fácilmente desde el gestor de paquetes de Ubuntu con el comando de consola: "# apt-get install mongodb".

- Secure Shell (ssh). Se trata de una herramienta opcional, con la que podremos acceder de manera remota al servidor. Instalaremos el servidor ssh con el gestor de paquetes de Ubuntu: "# apt-get install openssh-server".
- Node.js. Es el entorno con el que desarrollaremos la plataforma. Podemos instalar la versión que viene en los repositorios de Ubuntu con este comando: "# apt-get install nodejs".
- Node Package Manager (npm). Para gestionar los módulos de Node.js utilizaremos esta herramienta que se instalará con el comando: "# apt-get install npm".

Estas son las herramientas básicas para poder ejecutar nuestra plataforma. Las herramientas serán las mismas en otros sistemas operativos, aunque los comandos de instalación de dichas herramientas pueden cambiar.

7.1.3. Clonar y arrancar

Para concluir, queda como paso final clonar el repositorio, instalar los módulos del sistema usando el gestor de paquetes de Node.js, lanzar la base de datos y ejecutar el servidor. En la siguiente pieza de código podemos ver un listado con los comandos, sin sus salidas, que debemos ejecutar. Se debe tener en cuenta que es recomendable ejecutar la base de datos y el servidor en terminales distintos o en segundo plano, usando herramientas auxiliares como "screen".

```
1 ~$ mongod
2 ~$ git clone git@github.com:angajime/iot-server.git
3 ~$ cd iot-server
4 ~$ npm install
5 ~$ npm start
```

Con estos sencillos pasos el servidor estará listo para comenzar a recibir peticiones. Hay que recordar que se debe realizar un direccionamiento de puertos al servidor para el puerto 8000 para que las peticiones lleguen a la aplicación.

Bibliografía

[1] Ashton, K. (2016). That "Internet of Things" Thing. RFID Journal. Recuperado el 3 de junio de2016, desde http://www.rfidjournal.com/articles/view?4986

[2] Cisco And Ericsson Unveil Partnership (2015, 11 septiembre). Network Computing. Recuperado el 6 de junio de 2016, desde http://www.networkcomputing.com/networking/cisco-and-ericsson-unveil-partnership/830626442

[3] Internet de las cosas (IoT) empieza con Intel Inside®. (2016). Intel. Recuperado el 6 de junio de 2016, desde http://www.intel.es/content/www/es/es/internet-of-things/overview.html

[4] Internet of Things (IoT) Solutions. (2016). Google Developers. Recuperado el 6 de junio de 2016, desde https://cloud.google.com/solutions/iot/

[5] Gartner Says 6.4 Billion Connected. (2016). Gartner.com. Recuperado el 7 de junio de2016, desde http://www.gartner.com/newsroom/id/3165317

[6] The State of LTE September 2015. (2016). Opensignal.com. Recuperado el 7 de junio de2016, desde http://opensignal.com/reports/2015/09/state-of-lte-q3-2015/

[7] LTE Advanced and LTE Networks (2014). Qualcomm. Recuperado el 7 de junio de2016, desde https://www.qualcomm.com/invention/technologies/lte/advanced

[8] WiMAX Forum. (2016). Wimaxforum.org. Recuperado el 8 de junio de2016, desde http://www.wimaxforum.org/home

[9] Tomás Gironés, J. (2016). *El Gran libro de Android.* Barcelona: Marcombo.

[10] Blum, J. (2013). *Exploring Arduino.* Indianapolis, EEUU: Wiley.

[11] Smart City project in Santander to monitor Parking Free Slots. (2016). Libelium. Recuperado el 9 de junio de2016, desde http://www.libelium.com/smart_santander_parking_smart_city/

[12] Santander Facility. (2016). Smartsantander.eu. Recuperado el 10 de junio de 2016, desde `http://www.smartsantander.eu/index.php/testbeds/item/132-santander-summary`

[13] Smart Agriculture project in Galicia to monitor a vineyard with Waspmote. (2016). Libelium.com. Recuperado el 10 de junio de 2016, desde `http://www.libelium.com/smart\_agriculture\_vineyard\_sensors\_waspmote/`

[14] Pigeon Air Patrol. (2016). Pigeon Air Patrol. Recuperado el 12 de junio de 2016, desde `http://www.pigeonairpatrol.com/`

[15] Pressman, R., Campos Olguín, V., & Enríquez Brito, J. (2010). *Ingeniería del software.* México: McGraw-Hill.

[16] Gourley, D. & Totty, B. (2002). *HTTP : the definitive guide.* Sebastopol, EEUU: O'Reilly.

[17] Ornbo, G. (2013). *Node.js.* Madrid: Anaya Multimedia.

[18] Plugge, E., Membrey, P., & Hawkins, T. (2010). *The definitive guide to MongoDB.* New York, EEUU: Apress.

[19] Sriparasa, S. (2013). *JavaScript and JSON essentials.* Birmingham, Reino Unido: Packt Publishing.

[20] Chacon, S. (2009). *Pro Git.* Berkeley, EEUU: Apress.